AN OPENED DISTRIBUTED PLATFORM FOR AUTOMATIC CHAINING OF MULTIMEDIA INDEXING TOOLS

1Bassem HAIĐAR, 2Bilal CHEBARO

1Asstt Prof., Department of Computer Sciences, Faculty Of Sciences, Lebanese University, Lebanon
2Assoc. Prof., Department of Computer Sciences, Faculty Of Sciences, Lebanese University, Lebanon
E-mail: bassem.haidar@ul.edu.lb, bchebaro@ul.edu.lb

ABSTRACT

In the last decade, many research projects have focused on algorithms for segmentation and indexing of multimedia contents, many multimedia indexing tools has been developed by research laboratories. However many indexes need to be produced by a sequence of tools chained in a given order.

In this paper, we present a distributed platform for automatic multimedia indexing. This platform allows the automatic chaining of multimedia indexing tools in order to produce an expected index. We propose an algorithm for chaining multimedia indexing tools based on the input/output data type compatibility. This algorithm uses a backward recursive tracking to construct the chain(s) able to generate a given index. We propose scenarios to ensure the consistency of the automatic constructed chain(s). Distributed multimedia indexing tools are accessible via web services. A generic wrapper allows the easy integration of multimedia executable tools on the platform. The communication is based on SOAP/RPC protocol A central server plays the role of a repository and allows a centralised access to the platform.

Keywords: Distributed Multimedia Systems, Automatic Multimedia Indexing, Backward Chaining.

1. INTRODUCTION

A In the multimedia indexing domain, many indexes need to be produced by a sequence of tools chained in a given order. These tools are developed and distributed by different research teams, each of which is in general specialized on a single media. This makes the exploitation of these tools for cross-media analysis a difficult challenge and limits the potential research activities on multimedia indexing mining and multimodal indexing.

We propose a distributed platform for automatic multimedia indexing. The platform allows the easy integration of new multimedia indexing tools on the platform. These tools are modelled as processes. A chain of tool is modelled as a graph. The latter defines the order of execution of the different available processes. The scenario of the automatic multimedia indexing is the following: the user asks for an index, consequently, the platform tries to find the tool or the sequence of tools able to generate this type of index, finally, the chaining process stops when we find a multimedia document type.

The platform allows to share multimedia indexing resources (indexing tool, corpus, metadata) between several laboratories. New methods and scenarios of fusion and aggregation of multimedia indexing results will be easier to implement thanks to the platform.

Distributed multimedia indexing tools are accessible via web services. The communication is based on SOAP-RPC [1] protocol. The last has the advantage of posting data over the HTTP protocol. Therefore, it is able to get through firewalls that are a limitation for many distributed applications. We propose an easy scenario to integrate new indexing tools on the platform, this scenario allows research teams in multimedia indexing domain to share their tools with the multimedia indexing community.

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 Mediaworks project [4] used cooperation between information from text and image in order to develop a system that assists documentalists to index TV archives.

The project FERIA (Framework for Industrial Realization and Experimentation of multimedia Applications) [5] aims at developing a general and open framework allowing the easy development of dedicated industrial applications to edit interactive video documents.

The large-scale multimedia indexing project, presented in [6], proposed a distributed architecture for media acquisition, content processing, and document indexing. The main goal of the project is to exploit the parallel execution in order to distribute the workload.

[14] propose a web services framework for semantic multimedia indexing, they propose a method for structured metadata generated by different indexing algorithm in order to formulate a complex semantic query. The study consider that the developers should adapt their tools in order to be integrated on the platform, our study focus on the generation of all possible combination of tools able to generate a specific index. In this study we are not interested in the compatibility and interoperability of metadata generated by the indexing tools [15].

The Generic Factory Service [16] developed at Indiana University is a toolkit for wrapping scientific applications, that relies on the XSUL SOAP libraries [18] for Web services support. It also uses a serviceMap to generate new WSDL descriptions for every scientific application. It uses an XSUL Message Processor to intercept the SOAP calls for a particular Web service and route it to a generic class that invokes the scientific application using the information provided by the serviceMap document. The disadvantage of this implementation is that this approach is very XSUL specific.

Opal is developed as a Java based toolkit that automatically wraps any legacy applications with a Web services layer that can be integrated with Grid Security Infrastructure (GSI) based security, cluster support, and data management [18]. It allow using a configuration file to deploy an existing application as a web service provider. The platform allow to compose predefined scenarios using a workflow composition tool.

In our platform we propose a generic wrapper in order to integrate legacy indexing tools into the platform, and we focus on the dynamic chaining and composition problem of heterogeneous multimedia indexing tools, the goal is to find dynamically the workflow able to generate new multimedia indexes that can not be generated by a simple indexing tool.

3. AUTOMATIC CHAINING

In multimedia indexing domain, many index can only be produced by a sequence of tools. Take for example, the indexing of audio documents. In order to automatically transcribe a speech, we need a first tool to classify the audio content type into speech, music, noise and silence. Second, we use a tool to identify the language of a speaker in the case of a speech content. And finally, we use a third tool to transcribe the speech into text, using the right transcriptor.

The automatic multimedia indexing framework we introduce in this paper, allows, thanks to the chaining algorithm, to find the sequence of tools able to produce an expected index.

3.1 Chaining Algorithm

We propose a recursive algorithm that allows the construction of a sequence of tools. Based on a given requested index (which is in fact a descriptor asked by the user), the chaining algorithm must find a set of tools and their interactions. The algorithm is recursive and it stops when a primitive multimedia content (audio, video files) can be used as an input of the last chained indexer.

We start with a given "Index" and we try to find the unitary indexer able to generate this descriptor type. Once such a tool is found, two cases are present: i) The input type of this indexer is a multimedia content. Therefore, we stop the chaining. The given index can then, be generated
by the application of the chain to all documents corresponding to the input data type. ii) The input type of this indexer is another descriptor type corresponding to data generated by a hypothetic other tool. In this case, we repeat the same process recursively, integrating each time a new indexer at the top of the chain, until a multimedia content type is found.

### 3.2 Algorithm Implementation

We use Prolog\[7\], a declarative and relational programming language, to implement the chaining algorithm because of its ability to solve problems that involve relations between objects.

We formulate the problem as the following basic relationships:

The Service relationship \textit{Service (T1, In, Out)} represent a multimedia indexing service or tool where:

- T1 is the name or the identifier of the tool.
- \(\text{In} = [\text{I1, I2} \ldots] \) is the table representing the input data types.
- \(\text{Out} = [\text{O1, O2} \ldots] \) is the table representing the output data types.

We then, represent compatibility between tools as relationships described by the following logical rules:

\textit{Two tools T1, T2 are chained if}

\textit{There exist a certain I such that}

\textit{I is an input of T1, and}

\textit{I is an output of T2.}

In Prolog syntax, we check that data type \(I\) is an input data type of a service \(S\) if it is a member of the table \(\text{In}\) of service \(S\) with:

\(\text{contain}_\text{in}(S, I) : - \text{service}(S, \text{In}, \text{Out}), \text{member}(I, \text{In}).\)

The compatibility between two tools is then verified by:

\(\text{OR}(T1, T2, I) : - \text{contain}_\text{In}(T1, I), \text{contain}_\text{Out}(T2, I).\)

The “,” separator represent the “and” logic condition.

The \textit{OR} rules allow describing all possible chain relations between tools. A multimedia data type can be generated by more than one service. As a consequence, a service \(T\) can have multiple \textit{OR} relations for the same input data type. In Figure 1, the service \(T\) can have two \textit{OR} relations for the input data type \(I1\). But it does not have any \textit{OR} relation to connect the input data type \(I2\) to an existing service.

![Figure 1 OR relations](image-url)

A multimedia data type is represented by a special service having this data type as unique output data, and no input data:

\textit{Service (Data, [], [Data]).}

\textit{OR} relations construct graph of solutions which represent all possible compatibilities between tools. A specific solution represent a direct acyclic sub-graph, that starts with an index type as an output, and stops with one and only one data type as an input. Finding all possible solutions consists in browsing the graph recursively avoiding infinite loops.

\textit{Graph tracking}

Once the \textit{graph of solutions} has been built, all possible chains (sub-graphs) must be extracted, and filtered in order to keep only convergent ones.

A chain is considered to be convergent if it satisfies the three following conditions:

\textit{Condition A}:

The graph is not cyclic (i.e. one node can not be one of its children). To avoid infinite cycles, we define the predecessor rule between two services as:

\(\text{Predecessor}(X, Z) : - \text{Parent}(X, Y), \text{Predecessor}(Y, Z).\)

And we concatenate in one table all the output data type of all services found in our path.
Then three cases are possible:

1. If we find a service which is a predecessor of itself, and if its input data type is not yet recorded in the output table (see the B service on the Figure 2), this service is added to the solution sub-graph, and its output, I1, to the output table.

2. If we find a service which is a predecessor of itself, and if one of its input data type is already recorded in the output table (see service C on the Figure 2), we do not add this service to the solution sub-graph, in order to avoid infinite loops.

3. If we find a service which is not a predecessor of itself, and if one of its input data types is a member of the output table, (see service D on the Figure 2), we add the service to the solution sub-graph.

**Condition B:**

- All the graph nodes satisfy the AND rule. The AND(X) rule is true, if the service X has at least one OR relationship for every input data type. In other words, it is useless to add a service to the solution when one of its input data types can not be generated by any service present on the platform.

**Condition C:**

- And finally this chain converges to one and only one data source type. We consider only cases where multimedia content description can be generated from only one data source.

The major problem of automatic indexing is the consistency of the results produced by the above automatic chaining process.

Ensuring the syntactical compatibility between indexing tools is not enough to assume the consistency of resulting chains. The Figure 3 is an example of this problem, to generate the English text transcription from an audio file, a segmentation tool extracts the speech segment from other segments like music and noise segments, and then a language identification tool identifies the language of each speech segment. A Non_French_Filter extract the FrenchSegment from others speech segments, a Male/female segmentation tool separates speech segments pronounced by a male voice from speech segments pronounced by a female voice. And then a Male_Filter tool keeps only the FemaleSegments. Finally a transcription tool wlls transcript the FemaleSegments into text. This chain is syntactically correct, but inconsistent, because the EnglishSegment is filtered, and then no English text ca be generated.

We propose some mechanisms to ensure the consistency of the automatic constructed chain.

### 3.3.3 Strongly typed Input/Output

The first mechanism is to declare services with input/output data strongly typed. In this mechanism, the name of types is chosen to explain the content of the type. Figure 4 is an example of this mechanism.

The problem of such an approach is the complexity and the multiplication of data types. In fact, the platform is open; any multimedia indexing tool can be integrated to the platform, as soon as its input/output data types are declared by the developer. Without any policy to restrict the naming space of data types, several definitions maybe declared for a same content type. For example, an audio segmentation tool can be integrated with “AudioSegment” as output data type; another audio segmentation tool can be integrated with “SegmentAudio” as output data type.

The solution proposed in this mechanism consists in the intervention of a registration authority. When a new tool is integrated to the platform, and when one or more of its input/output data types is not already available on the platform, two cases are possible:
In the other case, the registration authority proposes to replace this type by another one already declared.

Using strongly typed data will limit the number of possible solutions produced by our automatic chaining algorithm as a consequence, and thus increases the consistency and the optimisation of the chain. But cost impact on the platform management is important. Let us mention here that the MPEG-7[3] typology can not be used to reduce this cost because it has not been designed for that purpose specifically.

3.3.4 Input/output typed using multimedia content hierarchical structure

In this mechanism, we propose to use more flexible type constraints on datatypes in order to allow more possible solutions of the automatic chaining process.

We consider that the set of data types is limited, and that there is a hierarchical structure such as for each data type, we can find another data type which generalizes the first one, excepted for one data type which is the most generic one (MultimediaContent in Figure 5)

All multimedia description types are sub-types of one of the multimedia content entities. These entities are structured in a multimedia content hierarchical structure, the basic multimedia entities elements and there hierarchies can be determined, here, from the Multimedia Description Schema basic entities of MPEG7 [8].

Then, a tool requiring the data type I1 as an input can be chained with a tool providing I1 or any data type which generalizes I1 as an output.

We so multiply the number of possible chains, but in the same way, we reduce the possible consistency of these chains.

To limit this problem, we suggest to develop a set of rules to ensure compatibility between identified tools (and not any longer between data types).

During integration of a new tool on the platform, an registration authority must add rules to control the chaining process between this tool and the other ones already existing on the platform. Here is an example of a possible scenario:

According to the multimedia content decomposition hierarchy illustrated by the Figure 5, a Non_French_Filter tool that extracts only French
speech segment from others speech segments requires an \texttt{AudioSegment} type as input data type, and generates an \texttt{AudioSegment} type as output data type. An \texttt{English_Transcription} tool, that extracts English text from audio file, requires an \texttt{AudioSegment} as input data type, and generates an \texttt{EnglishText} as output data type. The output of the \texttt{Non_French_Filter} tool can be chained to the input of the \texttt{English_Transcription} tool, but the constructed chain is never consistent because all English speech segments will be filtered by the \texttt{Non_French_Filter} tool. During the integration of this tool on the platform, the registration authority must specify that the \texttt{English_Transcription} tool should never be preceded by the \texttt{Non_French_Filter} tool.

We can use four types of rules to ensure so the consistency of the chain:

- **Rule 1:** \texttt{Predecessor(X)} = \texttt{Y}.
  - The tool \( X \) must have the tool \( Y \) as predecessor.
- **Rule 2:** \texttt{¬Predecessor(X)} = \texttt{Y}.
  - The tool \( X \) must not have the tool \( Y \) as predecessor.
- **Rule 3:** \texttt{Successor(X)} = \texttt{Y}.
  - The tool \( X \) must have the tool \( Y \) as successor.
- **Rule 4:** \texttt{¬Successor(X)} = \texttt{Y}.
  - The tool \( X \) must not have the tool \( Y \) as successor.

According to these rules, during the integration of the previous tools on the platform, the registration authority will apply the following rules:

If the \texttt{Non_French_Filter} tool is already added, during the integration of the \texttt{English_Transcription} tool, the following rule must be defined as:

\[¬\texttt{Predecessor(English\_Transcription)} = \texttt{Non\_French\_Filter}\]

If the \texttt{English_Transcription} tool is integrated before the \texttt{Non_French_Filter} tool, another rule must be declared by the expert:

\[¬\texttt{Successor(Non\_French\_Filter)} = \texttt{English\_Transcription}\]

In this paper, we do not argue for a better solution as far as it depends on the way the platform can be managed. We adopted the first solution because the number of different types is relatively reduced, and because it appears to be enough for concurrent indexing tool comparison.

4. **GLOBAL ARCHITECTURE**

The platform is composed of distributed services and a central server that implements three principal services: the repository service, the chaining service, and the access service (see Figure 7).

4.1 **Communication Details**

We use the SOAP [1] protocol for the RPC (Remote Procedure Call), SOAP (Simple Object Access Protocol) has been developed by Microsoft, IBM, DevelopMentor and UserLand Software. SOAP is a protocol for remote procedure calling and messaging with XML-encoded application data. It presents the advantage of posting data over the HTTP protocol. A SOAP message is written using the XML formalism and is sent over HTTP to a remote server. On the Web service side, the SOAP request is processed and the result is sent back to the client using a SOAP response.

We use Apache Axis [9] for development. Axis was developed by IBM alphaWorks. It provides SOAP support for Apache Tomcat application servers. Apache Axis use SAX for XML parsing.

We use a DataHandler [10] 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.

With the previous mechanisms, multiple files can be sent simultaneously by using a table of DataHandler objects, each of them handling the required file.

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.
4.2 Opened Platform, Tool Wrapper

A multimedia indexing 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, options needed for the execution, etc.

We have developed a wrapper, which is a package of code able to read the specification file, and then to launch the tool with the required command line which looks like:

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

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

```xml
<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>
      <input_type>Audio</input_type>
      <FileFormat>wav</FileFormat>
      <StorageInputPath>/Input_Path</StorageInputPath>
    </InputData>
  </Input>
  <Output>
    <OutputData>
      <output_type>AudioSegmentContent</output_type>
      <FileFormat>xml</FileFormat>
      <StorageOutputPath>/output_Path</StorageOutputPath>
    </OutputData>
  </Output>
</Component>
```

Figure 6 The specification file of an audio segmentation tool

This generic wrapper is independent from the type of the tool, and only the specification file is required.

Using the information thanks to a DataHandler table. An example of this table is explained in the following table:

<table>
<thead>
<tr>
<th>Row number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Table structure</td>
</tr>
<tr>
<td>2</td>
<td>serialised file DataHandler[2]</td>
</tr>
<tr>
<td>3</td>
<td>serialised file DataHandler[3]</td>
</tr>
<tr>
<td>4</td>
<td>serialised file DataHandler[4]</td>
</tr>
</tbody>
</table>

Table 1 DataHandler table

The first element in the table specifies the structure of the table. For example, according to the specification given in Table 2, the second element in Table 1 represent the first input parameter (param_in1), the third element represent the second input parameter etc…

Table 2 table structure of data exchange

<table>
<thead>
<tr>
<th>Structure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param_in1</td>
<td>2</td>
</tr>
<tr>
<td>Param_in2</td>
<td>3</td>
</tr>
<tr>
<td>Param_in3</td>
<td>4</td>
</tr>
<tr>
<td>Option_1</td>
<td>-Option1</td>
</tr>
<tr>
<td>Option_2</td>
<td>-Option2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Using the information from the specification file (number of input files, input data format, number of options,...), the wrapper read the table and save input files in order to launch the required command line able to execute the tool. Output data files are also returned using a DataHandler table.

4.3 Centralised Services

The platform implements three principal centralised services:

4.3.1 Repository service

The repository service allows the registration of new services on the platform. A new tool will be integrated on the platform by sending towards the repository service its specification file. The information contained in this file 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.

4.3.2 Chaining service

The chaining service implements the chaining algorithm (see section 3.1), we use a Logic Programming System XSB [12], that provides all the functionality of Prolog. The interprolog [13] library allows to invoke the XSB System from java class. The chaining service loads the list of available services on the platform from the repository service. And construct thanks to the chaining algorithm the graph of solutions that represents all possible compatibility between tools.

4.3.3 Access service

The access service allows a centralised access to the platform; its interface provides all the functionality of the platform:

- We can search for a given service using different parameters like its output or its input data type, or its name.
- We can search for available multimedia documents.
- We can launch a service by selecting this service and its required input data.
- We can ask for a index, the access service contact the chaining service, which construct the chain able to generate this index.

The access service is in fact a web service; its interface implements the previous functionalities. JSP (Java Server Page) is used to instantiate a client for the access web service. This client allows to dynamically loading available services on the platform (see Figure 8).

The Figure 9 explains the scenario of launching a service via the Access Service.

![Figure 6 scenario of launching a service via the access service](image)

The user asks the Access service for available services on the platform, the access service answers by displaying this list of services provided by the repository service. The user selects the desired service, and then the access service answers by displaying the available compatible documents registered in the repository service. The user selects relevant documents, and then the access service downloads the documents from the multimedia document service. Finally the access service sends the documents to the selected indexing service. This last one processes the documents and returns the results to the access service.

4.4 Distributed Services

4.4.1 Indexing service

The goal of the platform is to allow the access of distributed multimedia indexing tools, an indexing service can interface one or several indexing tools, developed by a single research team or specialized in a single media analysis like audio segmentation, or speech transcription. The indexing service must allow a user or another indexing service to use one of the handled indexing tools. 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-RPC interface that implements the service functionality.
4.4.2 Indexing tool representation
An indexing tool representation is the representation of one indexing tool of the indexing service; it is created by the central server. More precisely, the scenario of adding a new tool on the platform is the following one:

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

4.4.3 Indexing tool instance
An indexing tool may be used by different users simultaneously. So, for each user, we create a specific indexing tool instance that allows several executions of a same indexing tool by different users. An indexing tool instance has an identifier which is associated with a session number. The indexing tool instance implements the functionality of the generic wrapper (see section 4.2). It allows to launch the executable tools and to get the output results.

4.4.4 Multimedia documents service
A user can add a multimedia document access service to the platform. As a multimedia indexing service, the multimedia documents service allows to access to multimedia contents, and has a specification file that describes the type of contents. A SOAP interface is defined to transfer data from and to this service. We use DataHandler object to transfer multimedia documents.

5. CONCLUSION AND FUTURE WORKS
In this paper, we have introduced the main concepts of a distributed platform for automatic multimedia indexing, which supports an automatic policy of multimedia indexation. We have proposed a recursive algorithm for automatically chaining indexing tools, based on the compatibility between input and output data types of available tools. And we have proposed scenarios to check the consistency of the automatic constructed chain. Also, we have presented the global architecture of the platform, and the principal services. We have used this technology for the implementation of the Pidot[14] 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.

More research efforts have to be achieved in the future to solve problems related to the automatic indexing approach. The first 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 second 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.

REFERENCES:
http://www.w3.org/TR/soap/


Figure 7 The hierarchical structure of multimedia content

Figure 8 The global architecture of the platform