



## D2.2.5 SCALABLE CONTENT-BASED INDEXING AND RANKING

---

Advanced Search Services and Enhanced  
Technological Solutions for the European Digital  
Library

Grant Agreement Number: 250527

Funding schema: **Best Practice Network**

Deliverable D2.2.5 WP2.2

---

Deliverable

V.1.0 - 30 March 2012

Document. ref.: ASSETS.D2.2.5.CNR.WP2.2V1.0



BY NC SA

ASSETS Scalable Content-based indexing and ranking

D2.2.5 V1.0



## Table of Contents

<b>1. INTRODUCTION</b>	<b>2</b>
<b>2. IMAGE INDEXING AND RETRIEVAL</b>	<b>3</b>
<b>2.1 Introduction</b>	<b>3</b>
<b>2.2 Software Requirements Overview</b>	<b>3</b>
2.2.1 Requirements	3
2.2.2 Use Cases	4
Image indexing	4
Image searching	5
<b>2.3 Technical Documentation</b>	<b>8</b>
2.3.1 UML Diagrams	8
2.3.2 Service APIs: REST services	12
REST service for searching	13
REST service for inserting	14
2.3.3 Service APIs: Client API	15
Client API for inserting	15
Client API for searching	16
2.3.4 Software Packaging	17
2.3.5 Installation and configuration	17
2.3.6 Scientific foundations	18
<b>2.4 User Manual</b>	<b>19</b>
2.4.1 Text + Similarity Search	19
2.4.2 Image URL Similarity Search	20
2.4.3 File Uploading Image Similarity Search	22
<b>2.5 Concluding Remarks</b>	<b>23</b>
<b>3. 3D-MODEL INDEXING AND RETRIEVAL</b>	<b>24</b>
<b>3.1 Software Requirements Overview</b>	<b>24</b>
3.1.1 Functionality overview	24
3.1.2 Requirements	24
<b>3.2 Technical Documentation:</b>	<b>25</b>
3.2.1 3D search and retrieval use cases	25
3D model indexing use case	25
3D model search and retrieval use case	26
3.2.2 Service APIs	28
3D search and retrieval domain objects	28
3D extraction service interfaces	28
3D indexing service interfaces	30
3D retrieval service interfaces	31
3D search and retrieval client interfaces	32
3.2.3 Software Packaging	34
3.2.4 Installation and configuration	34
3.2.5 Scientific foundations	35



<b>3.3</b>	<b>User Manual</b>	<b>37</b>
3.3.1	Search by hand-drawn sketch	37
3.3.2	Search by uploaded 3d model	38
3.3.3	Search by id	40
<b>3.4</b>	<b>Concluding Remarks</b>	<b>42</b>
<b>4.</b>	<b>AUDIO INDEXING AND RETRIEVAL</b>	<b>43</b>
<b>4.1</b>	<b>Introduction</b>	<b>43</b>
<b>4.2</b>	<b>Software Requirements Overview</b>	<b>43</b>
4.2.1	Functionality overview	43
4.2.2	Requirements	43
<b>4.3</b>	<b>Technical Documentation:</b>	<b>44</b>
4.3.1	Audio Service Use Cases	44
	Indexing use case	44
	Retrieval use case	45
4.3.2	Audio Class Diagrams	47
	Audio Indexing Class Diagram	47
	Audio Searching Class Diagram	47
	Domain object	48
4.3.3	Service APIs: Rest Interfaces	48
4.3.4	Service APIs: Client Interfaces	48
	Audio Searching Service Interfaces	48
	Audio Indexing Service Interfaces	49
4.3.5	Software Packaging	50
4.3.6	Audio feature extractor installation	50
	Technical Requirements	50
	Installation Instructions for Redhat/Centos 5.X	50
	Installation instructions for debian/ubuntu	51
4.3.7	Audio Search Engine installation and configuration	52
	Technical requirements	52
	Installation Instructions	52
	Configuration	53
	Operation	55
	Troubleshooting	55
4.3.8	Scientific foundations	55
<b>4.4</b>	<b>User Manual</b>	<b>57</b>
4.4.1	Audio Search by existing track	57
4.4.2	Audio Search by uploading track	57
4.4.3	Audio Search by url	58
4.4.4	Audio Description	59
4.4.5	Audio Search by Audio Description	60
<b>4.5</b>	<b>Concluding Remarks</b>	<b>60</b>
<b>5.</b>	<b>VIDEO SUMMARIZATION, ADAPTATION, INDEXING AND RETRIEVAL</b>	<b>61</b>
<b>5.1</b>	<b>Software Requirements Overview</b>	<b>61</b>
5.1.1	Requirements	61
5.1.2	Use cases	61



Use case for video summarization	61
Use case for image similarity search	63
Use case for video similarity search	63
<b>5.2 Technical Documentation:</b>	<b>64</b>
5.2.1 UML Diagrams	64
5.2.2 Service APIs: REST interfaces	66
5.2.3 Services APIs: Client interfaces	68
5.2.4 Installation and configuration	70
Memory requirements	70
Additional software	70
<b>5.3 User Manual</b>	<b>72</b>
Video summarization	72
Video similarity search	74
<b>5.4 Bibliography</b>	<b>75</b>
<b>6. CONCLUDING REMARKS</b>	<b>76</b>

## Executive Summary

---

This is a technical document<sup>1</sup> detailing the ASSETS architecture and APIs for “*scalable content-based indexing and ranking*” components.

It introduces technical aspects of all the software services that have been defined, analyzed, implemented and tested during ASSETS WP2.2.

This document provides the following information:

- The software requirements overview;
- The technical documentation (UML diagrams, services description and API documentation, the software packaging and installation);
- The user manual.

---

<sup>1</sup> Part of the content of this deliverable already appears in Deliverable 2.0.4 “The ASSET APIs”



## 1. Introduction

---

The goal of the ASSETS WP2.2 is to enhance the usability of Europeana portal (the European Digital Library platform) with innovative services that aim at improving the existing search functionality. These services allow users to search multimedia objects based on content similarity. They are designed to be reused by any digital library.

This deliverable provides the technical documentation for the ASSETS *content-based indexing and ranking* services, needed to install, configure and use them.

The document is divided into four parts which neatly describe the four ASSETS content-based indexing and ranking services.

It introduces technical aspects of the services, such as the software requirements, the UML diagrams, the API documentation, the software installation and configuration, and the user manuals.

The Section 2 "Image indexing and retrieval" concerns the image similarity service, that allows users searching between Europeana images using an image as query.

The Section 3 "3D-model indexing and retrieval" describes the 3D Model service, which allows users to search for 3D models similar to a 3D model selected as query.

The Section 4 "Audio indexing and retrieval" refers to the audio service, that provides advanced music search and recommendation functionalities.

Finally, the section 5 "Video summarization, adaptation, indexing and retrieval" describes the video service, that aims at enhancing the functionalities of Europeana for searching, browsing, pre-visualizing and accessing video content.

## 2. Image indexing and retrieval

---

### 2.1 Introduction

Content-based image retrieval (CBIR) is becoming a popular way for searching digital libraries as the amount of available multimedia data keeps on increasing. CBIR applications are becoming popular for accessing cultural heritage information, as they represent a complement to metadata based search. In fact, in some cases, metadata associated with images do not describe the content with enough details to satisfy the user queries, or sometimes metadata are completely missing. Images containing reproductions of works of art contain a lot of implicit information that is not generally described in manually generated metadata.

Various search paradigms can be supported by means of this functionality. In the Europeana context, results of a metadata based search can be used as CBIR queries to refine the search, external documents can be used to access the Europeana content or even the Europeana content can be used to get information concerning documents owned by the users (for instance, who is the author of the painting contained in this picture?).

The objective of the Image indexing and retrieval service is to provide a system able to perform effective and efficient similarity search on image documents, including images of scanned manuscripts. The proposed system offers functionality that uses the real content of the images, rather than their metadata only, to search for other documents.

### 2.2 Software Requirements Overview

#### 2.2.1 Requirements

**Usability:** The web user interface should allow similarity search in a self explanatory and easy way. So, no particular settings, weights or parameters should be required during a search.

**Reliability:** The image similarity search engine will be based on three MPEG-7 Visual Descriptors that can be automatically extracted from images -- ScalableColor, EdgeHistogram, ColorLayout. The similarity (or dissimilarity) function used to compare any two images will be a linear combination of the distances suggested by the MPEG group for each of these features.

Given the similarity function used for comparing any two images, the index data structures will return approximate results. The relative quality of the results will be evaluated with the Recall and Error on Position measures. Expected values are 0.3 for Recall and less than 0.002 for Error on Position.

**Performance:** Should be able to answer queries within a second.

**User Interfaces:** A link for searching similar images should be added near each image returned by the Europeana in the web user interface independently from the type of query performed. In particular, the button or link for similarity searching should be displayed only for those images for which it would be possible to perform the content-based similarity search.

A text reporting the similarity score of the results images could be added in the results list of





similar images. However, this is not mandatory.

It should not be necessary to have a specific user interface for Content Providers. The standard methods and user interface used for ingestion of the images metadata and thumbnails should be used.

**Look & Feel:** All the links to the similarity services should be self-explanatory linked text.

**Licensing Requirements:** The image similarity search service is written in Java and requires the Java Runtime Environment. There are no requirements to acquire a licence for commercial third party software. Third party components are several open-source Java libraries (like LIRE).

**Applicable Standards:** The image features extraction software will accept most of the image formats used nowadays, i.e.; JPG, PNG, GIF, BMP. Please note that RAW photo camera files are not accepted.

Suggested resolution for the images is 500x500 pixels or more. The size of each image file should not exceed 10 Mega Bytes, to reduce downloading and features extraction times.

**System Documentation:** (a) Code is commented in a professional manner so that API documentation can be automatically generated, and (b) service documentation, detailing the installation, configuration, and use of the service is object of this deliverable.

## 2.2.2 Use Cases

### *Image indexing*

This use case describes the activities that are performed during the images indexing.

Actors:

- Europeana Ingestion Manager/Repos2SIP who harvests the metadata that will be used as input for the ingestion workflow (see also: Europeana - Requirements for integration of ingestion tools).
- ASSETS developers who develop the metadata enrichment services and use the ingestion workflow to insert content into the ASSETS database.
- ASSETS portal administrators who use the ingestion workflow to insert content into the ASSETS database and to visualize/analyze the ingestion logs.

Stakeholder:

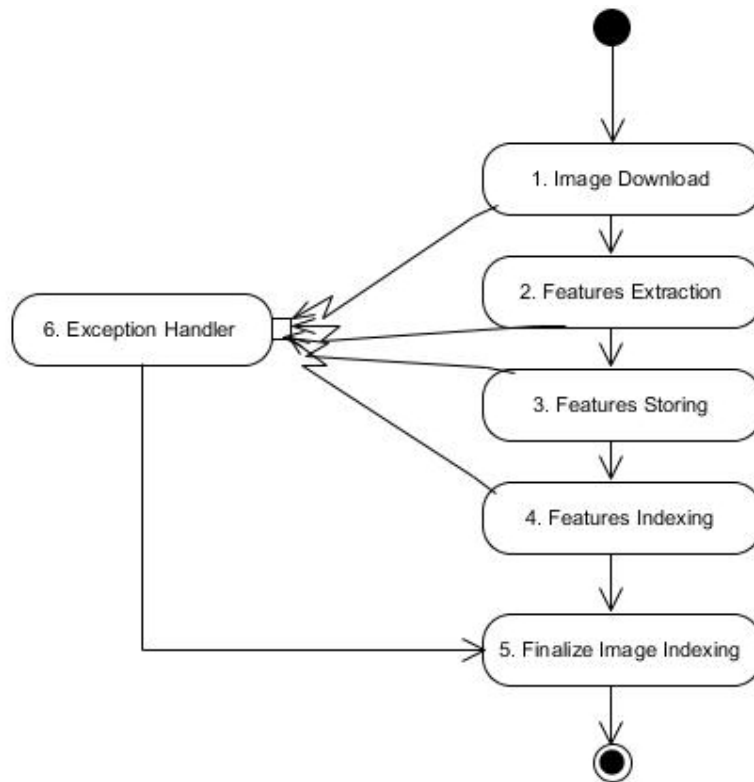
- End-user: wants to find images similar to one he/she has identified as interesting or wants to identify an image whose motif is unknown to him/her.
- Europeana: wants to offer powerful image search based on non-textual metadata and actual content.
- Content providers: are unable to develop themselves this type of search by and instead wish to offer the capability to their users via Europeana.

Preconditions:

- *Images to be indexed are available through internet by URLs*

Basic Flow of Events:





**Figure 1 - Image indexing use case**

0. **Start:** The use case begins when the *image search* system is requested to add an image to the index.
1. **Image Download:** The image is downloaded using its given URL. The image URL to be indexed is made available through the metadata of complex objects inserted in Europeana by the Content Providers.
2. **Features Extraction:** The requested content-based features are extracted. Feature extraction is a time consuming operation and is performed off-line.
3. **Features Storage:** Automatic extracted features are stored. The index will be built using the features previously stored. For this reason, in case of index rebuilding, it will not be necessary to extract the features again. However, features are necessary only for the content based image searching and indexing and it is not necessary to store them with the other metadata.
4. **Features Indexing:** Features are indexed for content based searching. An index is necessary to avoid comparison of the query image with all the images in the library. Thus, index is built for efficiency and it is typically not dynamic. Dynamically updating the index may result in rebuilding the index.
5. **Image Indexing:** Image indexing is finalized.

### **Image searching**

This use case describes the activities that are performed during the image content-based searching. The main goal of the service is to offer powerful yet easy to use image search

functionalities to the portal end-users.

*Actors:*

- Europeana Ingestion Manager/Repos2SIP who harvests the metadata that will be used as input for the ingestion workflow (see also: Europeana - Requirements for integration of ingestion tools).
- ASSETS developers who develop the metadata enrichment services ASSETS portal administrators who use the ingestion workflow to insert content into the ASSETS database and to visualize/analyze the ingestion logs.

*Stakeholder:*

- End-users: want to find images similar to one he/she has identified as interesting or wants to identify an image whose motif is unknown to him/her.
- Europeana: wants to offer powerful image search based on non-textual metadata and actual content.
- Content providers: are unable to develop themselves this type of search by and instead wish to offer the capability to their users via Europeana.

Basic Flow of Events:

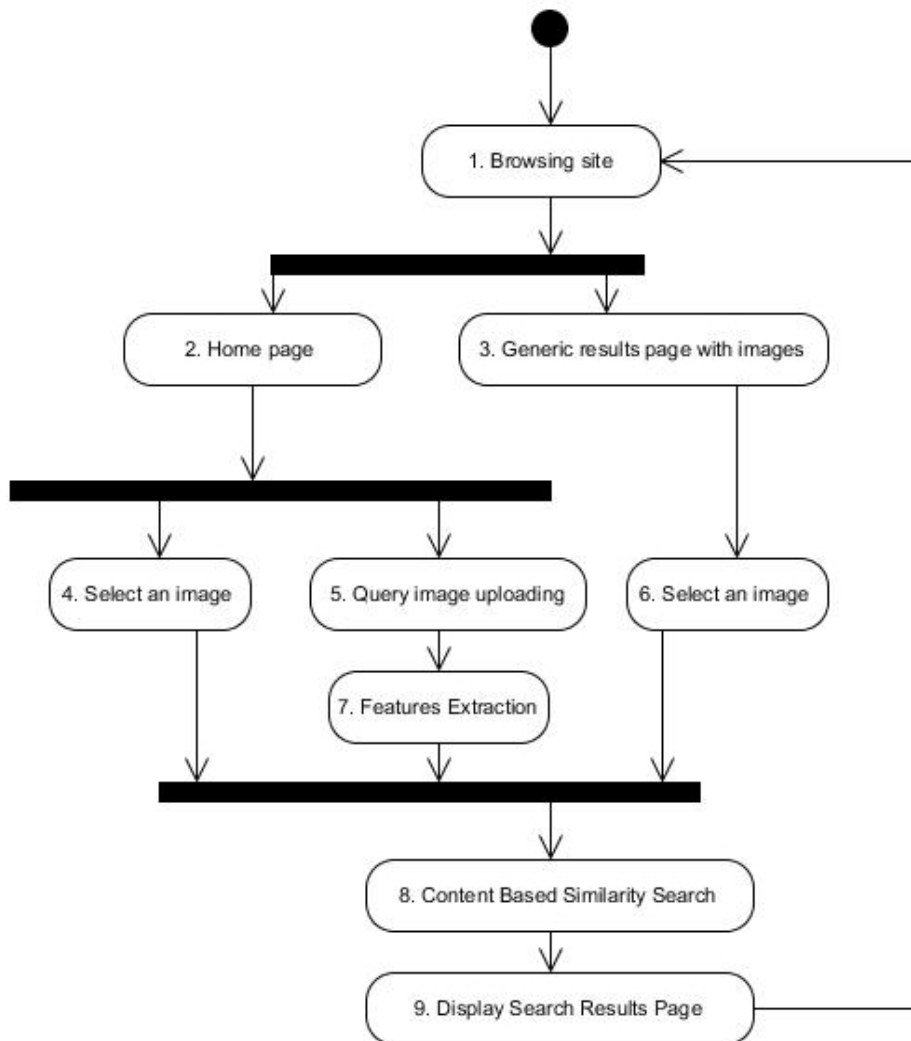


Figure 2 - Image searching use case

0. **Start:** The use case begins when the users access the site.
1. **Browsing site:** The user is browsing the site.
2. **Home Page:** The user is viewing the home page.
3. **Generic results page with images:** The user is viewing the results page of a previous generic search.
4. **Select an image:** The user selects an image for searching between the ones listed in the home page.
5. **Query image uploading:** The user selects an image between the ones in his device and uploads it in order to search for similar images.
6. **Select an image:** The user selects an image between the ones displayed in the results page.

7. **Features Extraction:** Features are extracted from the image.
8. **Content Based Similarity Search:** Content based similarity search is performed.
9. **Display Search Results Page:** Results in a web page are displayed to the users.

*Requirements for Content Provision:*

Images to be indexed must be available through URL at medium resolution (at least 500x500 pixels) and must have one of the following formats: JPG, PNG, GIF, BMP.

*Content requirements for Europeana portal:*

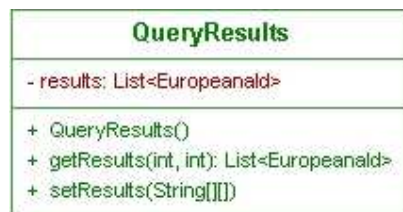
Images to be indexed must be available through URL at medium resolution (at least 500x500 pixels) and must have one of the following formats: JPG, PNG, GIF, BMP.

## 2.3 Technical Documentation

### 2.3.1 UML Diagrams

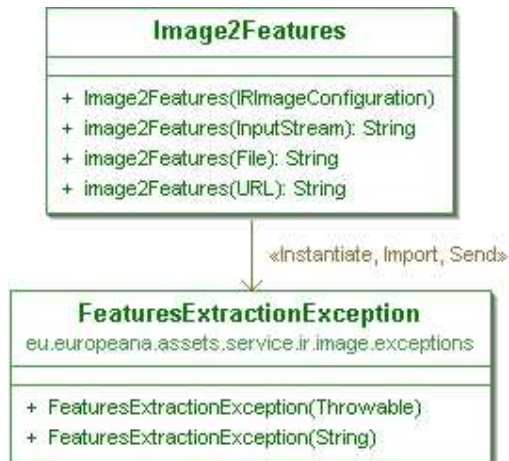
The class diagrams of the domain objects used by the service are presented in Figure 4Figure 3 and Figure 4:

- QueryResults contains the query results with the most similar images to the one the system was querying for, as a list of Europeana IDs.
- Image2Features allows extracting the visual features (necessary to perform an image similarity search) from an image. To extract them, it makes use of LIRE<sup>2</sup>, an open source Java content-based image retrieval library.



**Figure 3 - Images Indexing and Retrieval: QueryResults**

<sup>2</sup> <http://www.semanticmetadata.net/lire/>



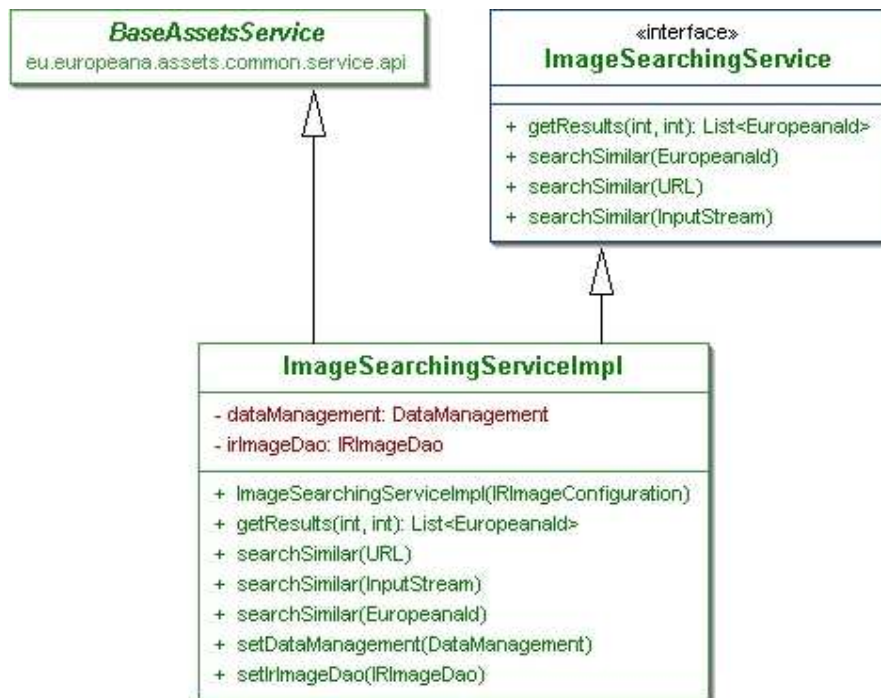
**Figure 4 - Images Indexing and Retrieval: Image2Features**

The class diagram of the Image Searching Service is presented in Figure 5.

In this diagram the available methods to perform an image similarity search are shown. It is possible to perform a search by:

- An image id (Europeanald),
- The stream of an image,
- An image URL.

Then, by calling `getResults`, the system returns the results of the query ranked by similarity.



**Figure 5 - Images Indexing and Retrieval: Searching Service Class Diagram**

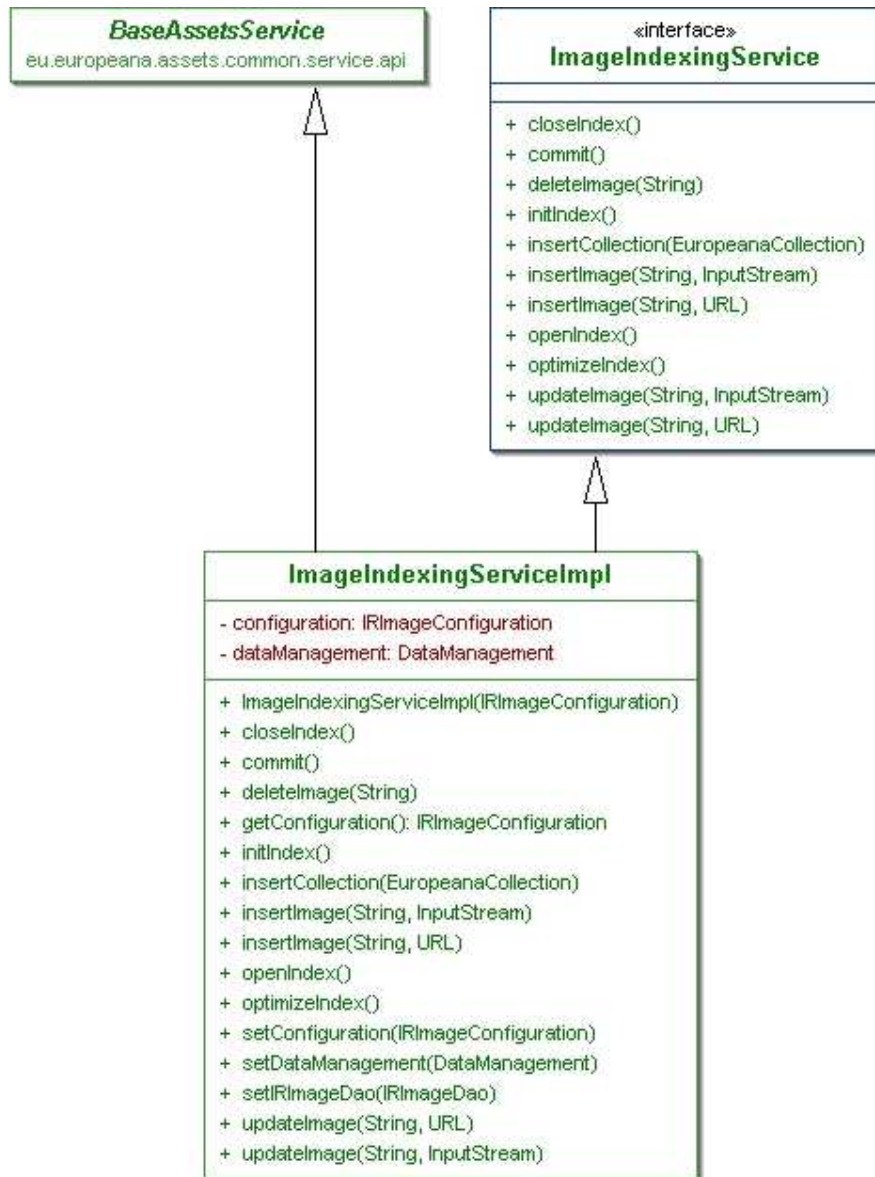
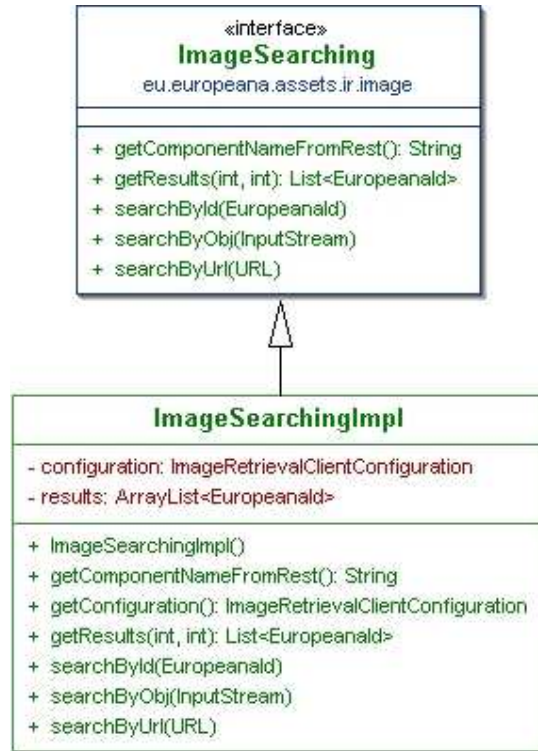


Figure 6 - Images Indexing and Retrieval: Indexing Service model

In this diagram the available methods to create and insert images into the image similarity search index are shown. The `initIndex()` method creates a new image index; it destroys the previous index (if any) and then builds a new one. By calling the `insertImage()` method the index can be populated by inserting images URLs or streams.



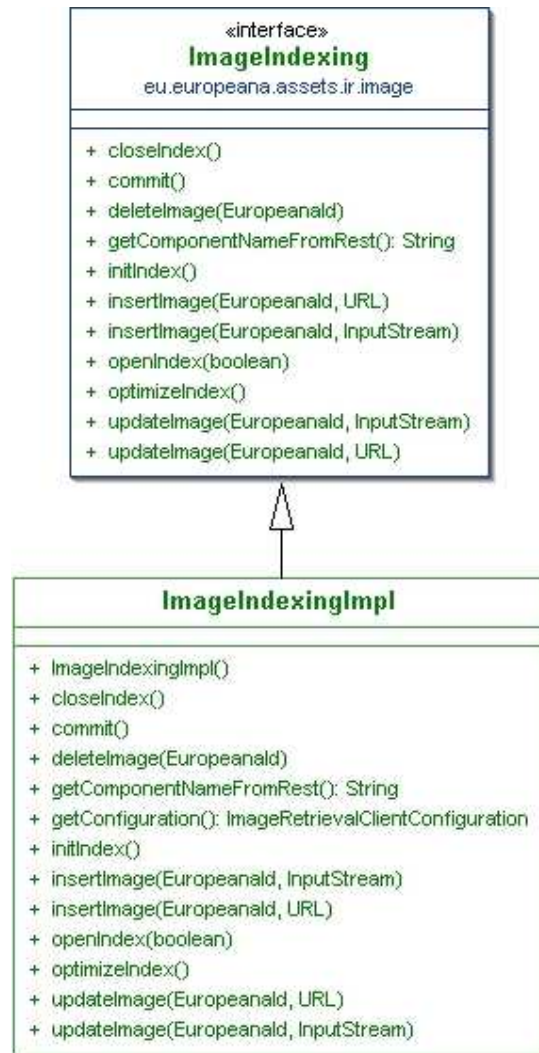
**Figure 7 - Images Indexing and Retrieval: Retrieval Client Model**

In this diagram the client side methods used to perform an image similarity search are shown. They allow performing a search by:

- An image id (Europeanald),
- The stream of an image,
- An image URL.

Then, by calling the `getResults` method the system will return the results of the query, ranked by similarity.





**Figure 8 - Images Indexing and Retrieval: Indexing Client Model**

In this diagram the client side methods used to create and insert images into the image similarity search index are shown. `initIndex()` is the method to be invoked in order to create a new image index; it destroys the previous index (if any) and then builds a new one. Then, by calling `insertImage()` the index can be populated by inserting images URLs or streams.

### 2.3.2 Service APIs: REST services

Among the REST services some are needed to build an image index and some to perform image similarity searches. Figure 9 shows a table with the available services.

IRImage Services			
IRImage			
IRImage Search Service allows users to build an image index and to performs visually image similar searches			
Method	Response type	Path	Function
GET	XML/JSON	/assets/ir-image /searching /rest/searchById	Searches similar images starting from an EuropeanId image already in the index
POST-MULTIPART	XML/JSON	/assets/ir-image /searching /rest/searchByObj	Searches similar images starting from an uploaded sample image
GET	XML/JSON	/assets/ir-image /searching /rest/searchByUrl	Searches similar images starting from an URL of a sample image
GET	-	/assets/ir-image /indexing /rest/openIndex	Opens the index for inserting
GET	-	/assets/ir-image /indexing /rest/closeIndex	Commits the indexing process
POST-MULTIPART	-	/assets/ir-image /indexing /rest/insertImageObj	Inserts an image into the index
GET	-	/assets/ir-image /indexing /rest/insertImageUrl	Inserts an image into the index

Figure 9 - IRImage REST services

### REST service for searching

The search service provides methods to perform a visual similarity search. They allow the user to perform a search by:

- An image id (EuropeanId),
- The stream of an image,
- An image URL.

The search methods return a list of EuropeanId containing the ids of the most similar images for the query.

Prefix path of the service is: **/assets/ir-image/searching/rest**

In Table the main service information needed to call it are shown.

Method	Response type	Name	Parameters	Function
GET	XML/JSON	<b>searchById</b>	<b>@imageQueryId</b> , Id of the query image <b>@numResults</b> , number of results to return. Default value: 100 (if the parameter is missing).	Searches similar images starting from an image Id already in the index

POST-MULTIPART	XML/JSON	<b>searchByObj</b>	<b>@imgFile</b> , InputStream of the query image <b>@numResults</b> , number of results to return. Default value: 100 (if the parameter is missing).	Searches similar images starting from an uploaded sample image
GET	XML/JSON	<b>searchByUrl</b>	<b>@imageQueryURL</b> , URL of the query Image <b>@numResults</b> , number of results to return. Default value: 100 (if the parameter is missing).	Searches similar images starting from a URL of a sample image

**Table 1 - REST Search methods**

**REST service for inserting**

The insert service provides the needed methods to build a new image index and/or to insert new images.

Prefix path of the service is: **/assets/ir-image/indexing/rest/**

In Table the main service information needed to call it are shown.

Method	Response type	Name	Parameters	Function
GET	-	<b>openIndex</b>	<b>@append</b> , boolean value. If false the index will be built from scratch.	Opens the index for inserting
GET	-	<b>closeIndex</b>	-	Commits the indexing process and close the index
POST-MULTIPART	-	<b>insertImageObj</b>	<b>@imageObj</b> , InputStream of the image to insert <b>@imageld</b> , id of the image to insert	Inserts an image into the index
GET	-	<b>insertImageUrl</b>	<b>@imageUrl</b> , URL of the image to insert <b>@imageld</b> , id of the image to insert	Inserts an image into the index
POST-MULTIPART	-	<b>updateImageObj</b>	<b>@imageObj</b> , InputStream of the image to update <b>@imageld</b> , id of the image to update	Updates an image into the index
GET	-	<b>updateImageUrl</b>	<b>@imageUrl</b> , URL of the image to update <b>@imageld</b> , id of the image to update	Updates an image into the index
GET	-	<b>deleteImage</b>	<b>@imageld</b> , id of the image to delete	Deletes an image from the index
GET	-	<b>optimizeIndex</b>	-	Optimizes the index
GET	-	<b>commit</b>	-	Commits the indexing process

Table 2 - REST methods for management of the image index

### 2.3.3 Service APIs: Client API

#### Client API for inserting

This API provides the needed methods to interface with the inserting service. Next table explains the available methods for inserting.

API	ImageIndexing
Responsibility	<i>Allows building and populating the image index.</i>
Provided methods	<p><b>void initIndex()</b>  <i>Creates a new image index. It destroys the previous index (if it exists) to build a new one.</i></p> <p><b>void insertImage(Europeanald imageId, InputStream imageObj)</b>  <i>Inserts an image into the index.</i>  <i>Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i>  <b>@imageObj</b> InputStream of the image to insert  <b>@imageId</b> Europeanald of the image to insert</p> <p><b>void insertImage(Europeanald imageId URL imageUrl)</b>  <i>Inserts an image into the index.</i>  <i>Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i>  <b>@imageUrl</b> URL of the image to insert  <b>@imageId</b> Europeanald of the image to insert</p> <p><b>void updateImage(Europeanald imageId, InputStream imageObj)</b>  <i>Updates an image into the index.</i>  <i>Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i>  <b>@imageObj</b> InputStream of the image to update  <b>@imageId</b> Europeanald of the image to update</p> <p><b>void updateImage(Europeanald imageId URL imageUrl)</b>  <i>Updates an image into the index.</i>  <i>Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i>  <b>@imageUrl</b> URL of the image to update  <b>@imageId</b> Europeanald of the image to update</p>



	<p><b>void deleteImage(Europeanald imageId)</b>  <i>deletes an image from the index.</i></p> <p><b>@imageId</b> Europeanald of the image to delete</p> <p><b>void commit()</b>  <i>Commits the indexing process.</i></p> <p><b>void optimizeIndex()</b>  <i>Optimizes the index.</i></p> <p><b>void openIndex(boolean append);</b>  <i>Opens the index for inserting.</i></p> <p><b>@append</b> If false the index will be built from scratch</p> <p><b>void closeIndex()</b>  <i>Commits the indexing process and close the index.</i></p>
Dependencies	ASSETS Common

### Client API for searching

This API provides the needed methods to interface with the searching service. Next table explains the available methods for searching.

API	<b>ImageSearching</b>
Responsibility	<i>Allows to perform image similarity searches.</i>
Provided methods	<p><b>void searchById(Europeanald euld)</b>  <i>Searches similar images starting from an Europeanald already in the index.</i></p> <p><b>@imageQueryId</b> Europeanald of the query image</p> <p><b>void searchByObj(InputStream imgObj)</b>  <i>Searches similar images starting from a sample image.</i>  <i>A query image should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i></p> <p><b>@imageQueryObj</b> InputStream of the query image</p> <p><b>void searchByUrl(URL imageUrl)</b>  <i>Searches similar images starting from a sample image.</i>  <i>A query image should have a size of at least 500x500 pixels and available in one of the following formats: JPG, PNG, GIF, BMP.</i></p> <p><b>@imageQueryURL</b> URL of the query Image</p>

	<p><b>List&lt;EuropeanId&gt; getResults(int startFrom, int numResults)</b>  <i>Returns the results of the query.</i></p> <p><b>@startFrom</b> index to start</p> <p><b>@numResults</b> number of results to return. If the value is set to -1, it returns all the query results.</p> <p>It returns a List of EuropeanId containing the ids of the query results.</p>
Dependencies	ASSETS Common

### 2.3.4 Software Packaging

The image index software is 100% Java code. It consists of a jar library file, some configuration files and some library dependences.

The software needs the following libraries:

- Image-index library:  
**melampo.jar**  
 This is the main image index module
  
- Dependence libraries:  
**lire.jar, messif.jar, mifile.jar, conja.jar, trove.jar, mtree.jar, jama.jar, lucene.jar, vir.jar**  
 Feature extraction, and various utilities used in the main index module
  
- Configuration files:  
**assets-ir-image.properties, assets-ir-image-fx.properties, indices.properties, LIRE\_MP7ALL.properties**

### 2.3.5 Installation and configuration

The image index installation and configuration steps are quite simple.

The software consists of a set of jar libraries and some configuration files. In order to install the image service the following configuration files will need to be set:

**assets-ir-image.properties, assets-ir-image-fx.properties, indices.properties, LIRE\_MP7ALL.properties**

**assets-ir-image.properties** sets the image index home path and the feature extraction configuration file.

It contains the following parameters:

- **image\_index\_home** = `${image.index.location}`

that is the file system path of the image index. The index folder contains the image index files.

- **fx\_config\_file** = `${image.fx.config.location}`



that is the file system path of the image features extractor configuration file.

**assets-ir-image-fx.properties** contains the settings for the image features extraction.  
**At the moment it mustn't be modified.**

**indices.properties** sets the similarity implementation to be used (currently ScalableColor, EdgeHistogram and ColorLayout MPEG-7 descriptors)

**LIRE\_MP7ALL.properties** is the main image index configuration file. This file is used to configure the index. It contains the following parameters:

#path syntax: "\*" means parent path of this configuration file (path of the image index folder)

- archive\_0 = \*/binaries/FCArchive-testset-lire.dat

Path of the binary files of the visual features.

#image index

- luceneIndexPath = \*/LuceneLire\_filtered

Path of the image index.

#Pivot file

- PivotsPath = \*/binaries/LireObjectPivots\_filtered\_10k.dat

Pivot file (supplied with the index) to use during the indexing and the searching process.

#Num of pivots

- num\_of\_pivots = 10000

Number of pivots in the pivot file.

# Top pivots to be used in the index

- maxpivsIndex = 50

Maximum number of pivots to use during the indexing process.

# Top pivots to be used in the query

- maxpivsQuery = 50

Maximum number of pivots to use during the searching process.

### 2.3.6 Scientific foundations

Amato G., Savino P., *Approximate Similarity Search from another "Perspective"*, SEBD 2008.

Bolettieri P., Falchi F., Lucchese C., Mass Y., Perego R., Rabitti F., Shmueli-Scheuer M., *Searching 100M Images by Content Similarity*, IRCDL 2009.

Gennaro C., Amato G., Bolettieri P., Savino P., *An approach to content-based image retrieval based on the Lucene search engine library*, ECDL 2010.

Amato G., Bolettieri P., Falchi F., Gennaro C., Rabitti F., *Combining local and global visual feature similarity using a text search engine*, CBMI 2011.

Gennaro C., Amato G., Bolettieri P., Savino P., *An approach to Content-Based Image Retrieval based on the Lucene search engine library (Extended Abstract)*, SEBD 2011.



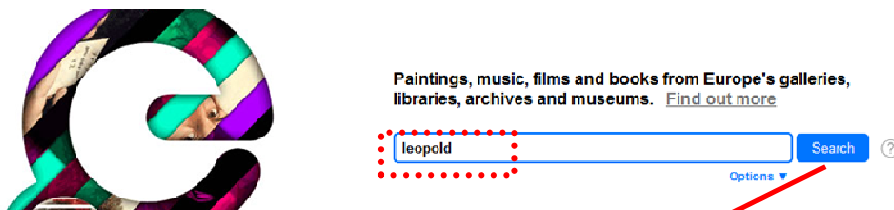
## 2.4 User Manual

In the next pages, the functionality of the image search service will be demonstrated through some search samples on the project portal (<http://assetsdemo.atc.gr/portal/>).

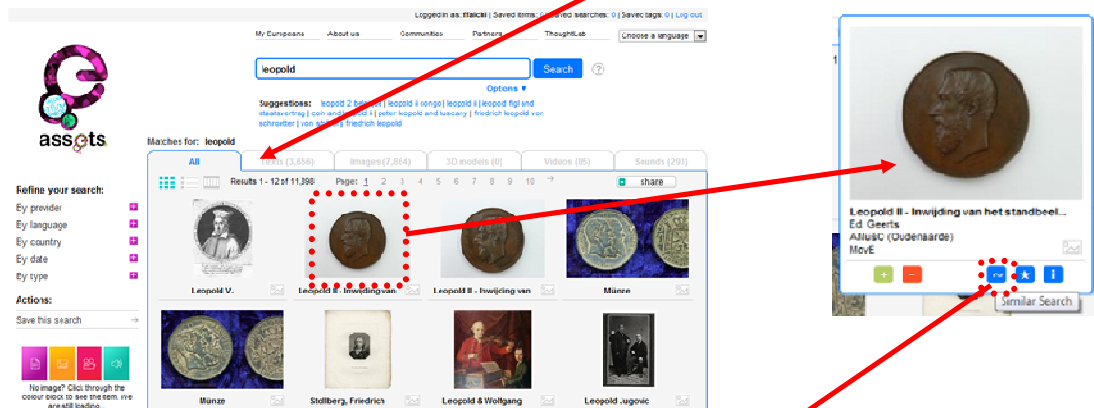
### 2.4.1 Text + Similarity Search

**Description:** the user first searches for “leopold” as text query, then select one of the results as a query for searching similar objects with the respect to the visual content.

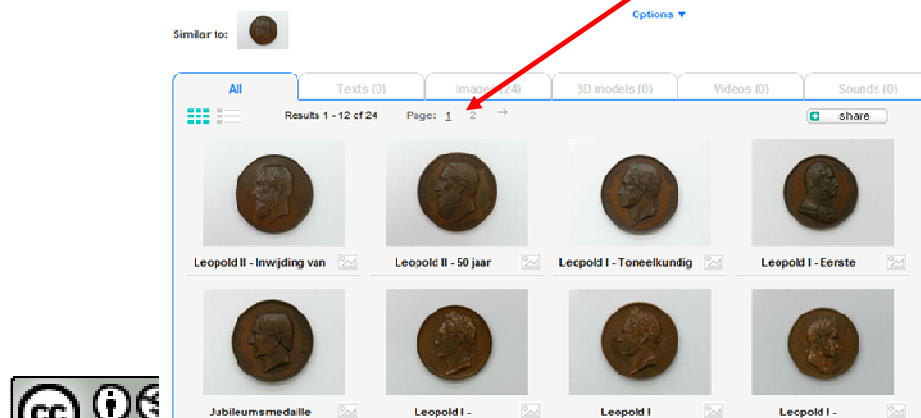
#### #1 The user searches for “leopold”



#### #2 Text search results page. The user clicks on the similar button of one of the results



#### #3 Similar images are retrieved





## 2.4.2 Image URL Similarity Search

**Description:** the user is interested in Europeana objects similar to the one he/she has found on a web page.

**#1 The user finds an image while navigating the British Library's web site**

<http://www.bl.uk/onlinegallery/onlineex/apac/other/019wdz000002060u00007000.html>

**BRITISH LIBRARY**

# ONLINE GALLERY

See 30,000 items from our collection

Search the Online Gallery

Online Gallery Home | Virtual books | Online exhibitions | Highlights tour | Personal galleries

[bl.uk](#) > [Online Gallery Home](#) > [Online exhibitions](#) > [Asia, Pacific and Africa Collections](#) > Encampment at Bishnupur (Bengal); elephants, bullock-carts, horses and an Englishman on horseback in the foreground, 29 January 1823

**Encampment at Bishnupur (Bengal); elephants, bullock-carts, horses and an Englishman on horseback in the foreground, 29 January 1823**

Artist: D'Oyly, Sir Charles (1781-1845)  
 Medium: Pen and ink on paper  
 Date: 1823

Interactive zoomable image (needs Flash)  
 Full size printable image  
 More metadata

**Search within this collection**

**Favourites**  
 (What is this?)  
 You must [log in](#) to see your favourites.

**Personal galleries**  
 (What is this?)  
 You must [log in](#) to create and edit galleries.

**#2 The user right clicks on the image and select "Copy image URL"**

Artist: D'Oyly, Sir Charles (1781-1845)

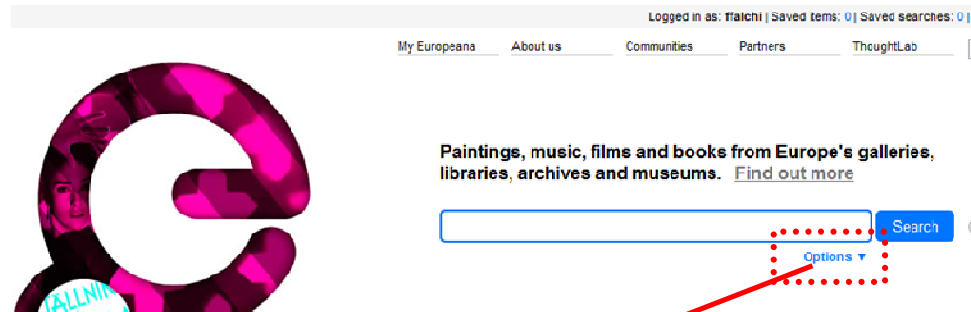
- Save image as...
- Copy image URL**
- Copy image
- Open image in new tab
- Chrome to Phone
- Search Image on TinEye
- Inspect element

Pen and ink drawing, by Sir Charles D'Oyly (1781-1845), o

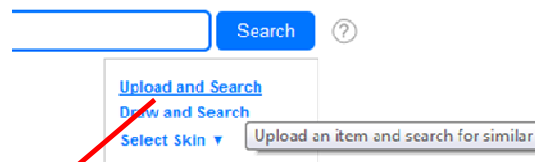
**The copied URL is**

[http://ogimages.bl.uk/images/019/019WDZ000002060U00007000\[SVC1\].jpg](http://ogimages.bl.uk/images/019/019WDZ000002060U00007000[SVC1].jpg)

**#3 The user goes to the Europeana portal and click on the "Options" menu**



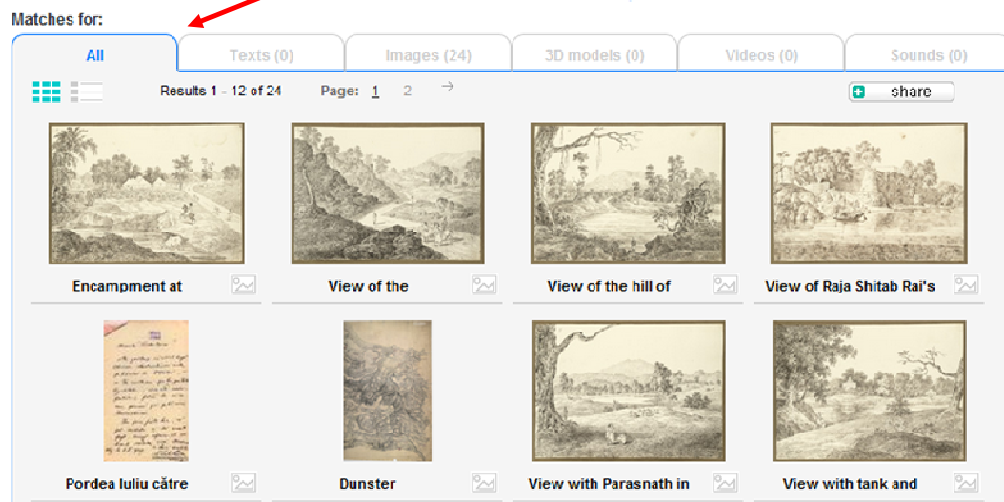
**#4 The user selects "Upload and Search"**



**#4 The user pastes the URL of the image and press the Search button**



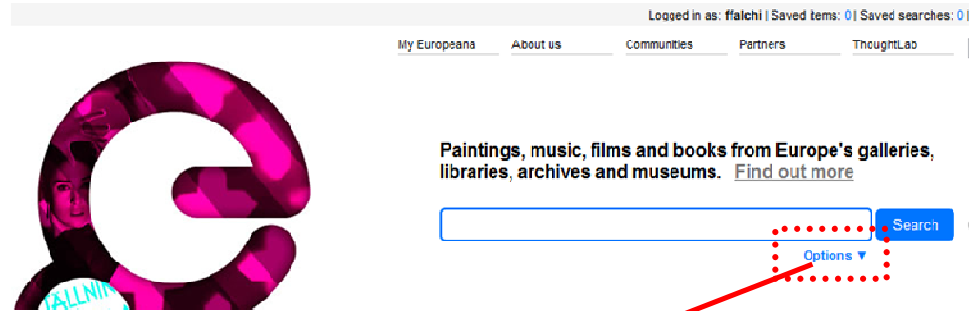
**#4 The very same object together with similar items in Europeana are retrieved**



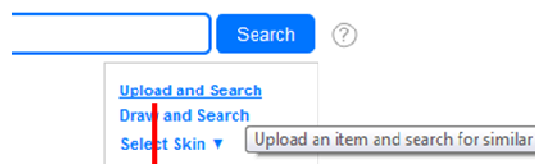
## 2.4.3 File Uploading Image Similarity Search

**Description:** the user has a photo and wants to find similar content in Europeana.

**#1 The user goes to the Europeana portal and click on the "Options" menu**



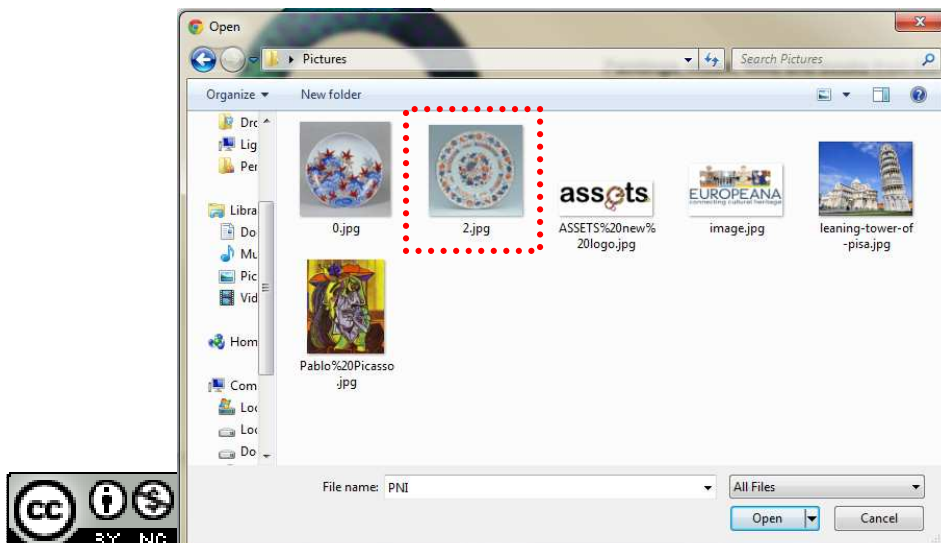
**#2 The user selects "Upload and Search":**



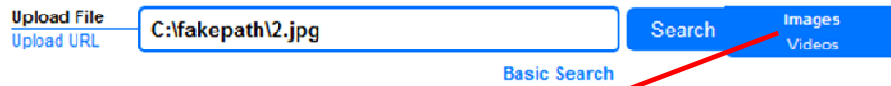
**#3 The user clicks on the box:**



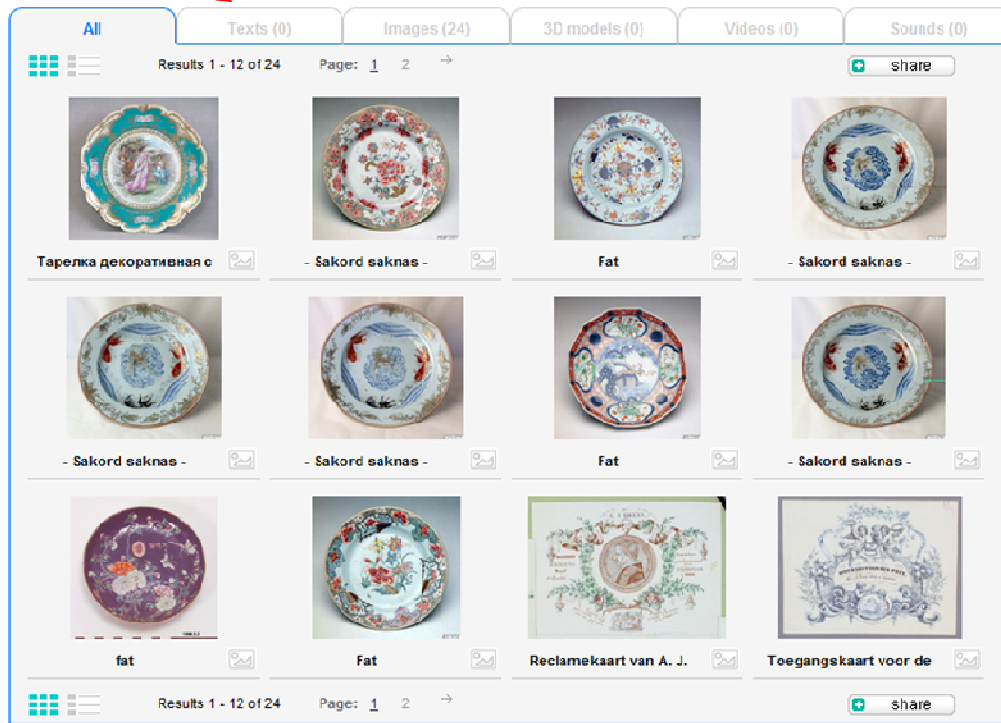
**#4 The user navigates its own computer file system eventually selecting an image**



**#5 The path is automatically inserted in the search and the user selects Search/Images**



**#6 Similar objects are finally retrieved:**



## 2.5 Concluding Remarks

The image search and retrieval index was demonstrated during the first year review and published in the Europeana ThoughtLabs:

<http://pro.europeana.eu/web/guest/thoughtlab/new-ways-of-searching-and-browsing>

## 3. 3D-model indexing and retrieval

---

### 3.1 Software Requirements Overview

Apart from the well-known kind of data, like text, images, video and sound, Europeana intends to host 3D representations of cultural objects. Thus, a technology is required which can index large amounts of 3D models and enable the fast search and retrieval of 3D models.

A user would like to search for 3D models geometrically similar to a query model. The 3D search interface will allow three types of queries: models uploaded from the user, models returned from a previous search and hand-drawn sketches.

#### 3.1.1 Functionality overview

In the following paragraph, the different types of search will be described.

##### Upload a 3D model and search for similar models

The user accesses the interface by selecting the appropriate tab. S/he uploads a 3D model by clicking a "Browse" button and selects the model file, which is located on his/her local hard disk. The model is uploaded and the search similar function returns a list of similar results.

##### Select one of the existing 3D models and search for similar models

The user accesses the interface by clicking on a "Search similar" link, while browsing through the existing models. The search similar function returns a list of similar results.

##### Create a sketch and search for similar models

The user accesses the interface by selecting the appropriate tab on the user interface. S/he draws a sketch and performs a search on the 3D content. The search similar function returns a list of similar results.

#### 3.1.2 Requirements

**Usability:** The service should be self explanatory and easy to use even from a non-expert user.

**Reliability:** The 3D search functionality should have a 24/7 availability and work every time within acceptable time limits for a user search. The returned results should be geometrically similar to the queried 3D object.

**Performance:** The feature should have no adverse effects on the performance of the Europeana platform.

**Look & Feel:** Font and colors should be in line with Europeana brand guidelines.



**Layout and Navigation:** Search similar button, upload interface, sketch interface.

**Communications Interfaces:** The interface will use stateless HTTP requests for the communication with other services or clients.

**Licensing:** The 3D search service will be implemented in Java and requires the Java Runtime Environment. There are no requirements to acquire a license for commercial third party software. Third party components are a Java virtual machine and several open-source Java libraries (like Log4J). The implementation uses well-known standards like HTTP and XML.

**Documentation:** Documentation for developers will be delivered as code documentation and as a detailed document outlining the architecture and the interaction between the used components.

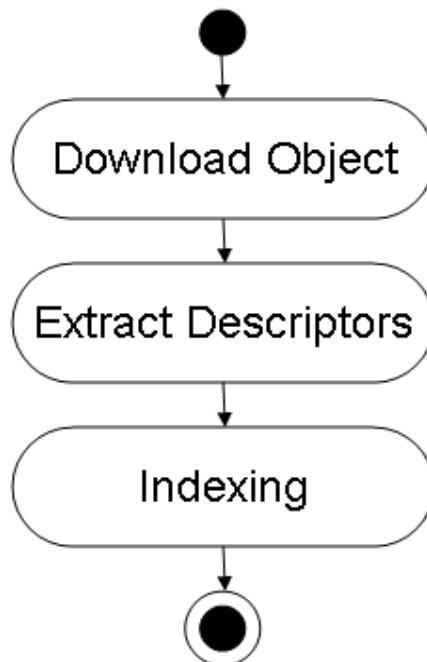
## 3.2 Technical Documentation:

### 3.2.1 3D search and retrieval use cases

Two major use cases are identified in relation to the 3D search and retrieval service: the 3D model indexing (UC1) and the 3D search and retrieval (UC2). In the following paragraph, a brief description of each use case will be given.

#### *3D model indexing use case*

The following Figure describes the basic flow of events during the 3D model indexing.

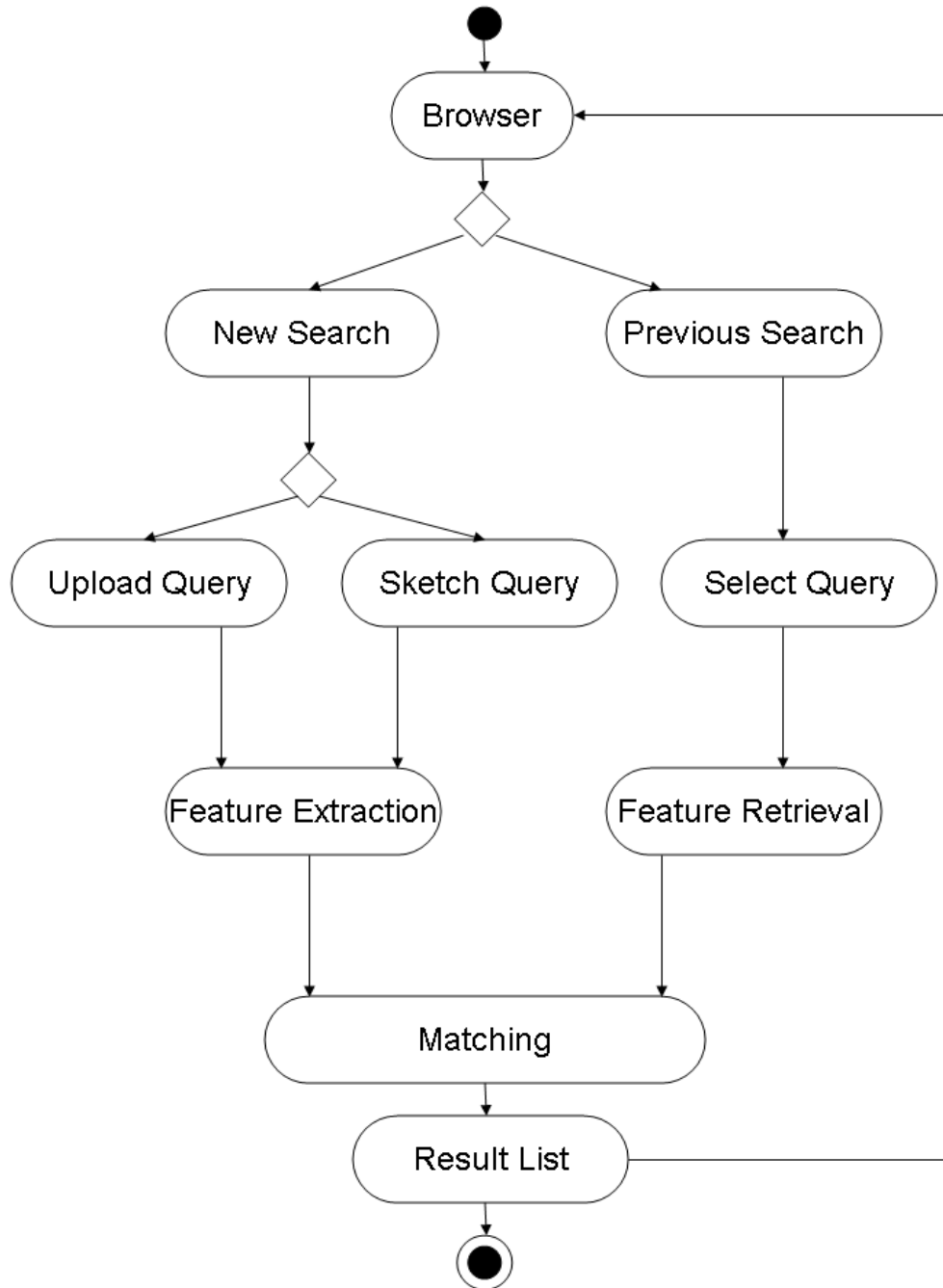


The use case begins with the Europeana Ingestion subsystem downloading the 3D model from the Content Provider. The 3D model is sent to the Feature Extraction service, which extracts the 3D descriptors and sends them back to the Ingestion subsystem. The 3D descriptors are indexed along with the rest of the metadata. Alternatively, the Feature Extraction service can download the 3D model itself, under the assumption that it is accessible. Available 3D model formats are: .wrl, .x3d, .off, .obj and .3ds.

After the use case is completed, the system has a set of 3D descriptors indexed along with the object metadata.

### ***3D model search and retrieval use case***

The following Figure describes the basic flow of events during the 3D model search and retrieval.



The use case begins with the user accessing the Europeana site. The user uploads a 3D model or draws a 2D sketch for using it as a query. The model/sketch is sent to the Feature Extraction Service, which extracts the 3D descriptors. After the matching procedure, the search results are returned to the user.

In an alternative flow, the use case begins with the user accessing the Europeana site. The user selects a 3D model, which was returned in a previous search for using it as a query. The



features are retrieved from the index. After the matching procedure, the search results are returned to the user.

After the use case is completed, the user receives a list of results in which the retrieved objects are geometrically similar to the given query object. Available object formats are: .wrl, .x3d, .off, .obj and .3ds for 3D model queries, as well as 100x100 monochrome .png images.

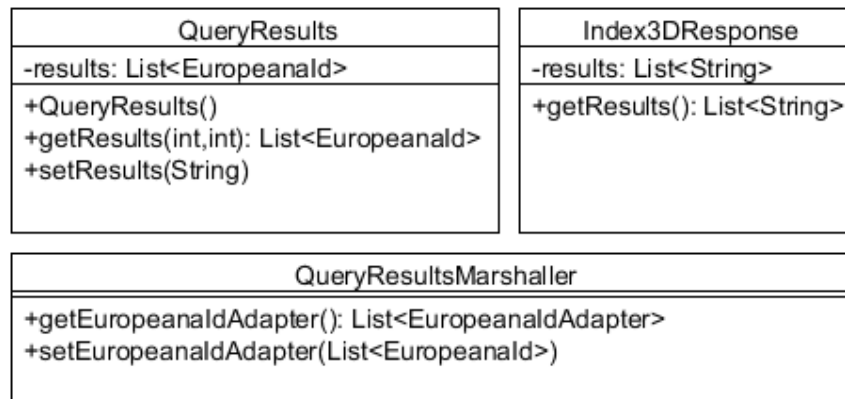
### 3.2.2 Service APIs

The Search and Retrieval framework consists of the following services:

- the **Extraction3D** service, responsible for extracting the 3D low-level features out of the 3D objects or hand-drawn sketches,
- the **Indexing3D** service, responsible for creating the 3D index,
- the **Retrieval3D** service, responsible for returning to the user results similar to a given query object.

#### 3D search and retrieval domain objects

In the following Figure, the class diagrams of the domain objects used by the 3D services are shown.



- *QueryResults* contains the query results with the most similar 3D models to the query object, as a list of Europeanalds.
- *QueryResultsMarshaller* marshals the 3D search results in a list of EuropeanaldAdapter objects (Rest Service purpose).
- *Index3DResponse* contains the 3D search results as a list of EuropeanaldUris.

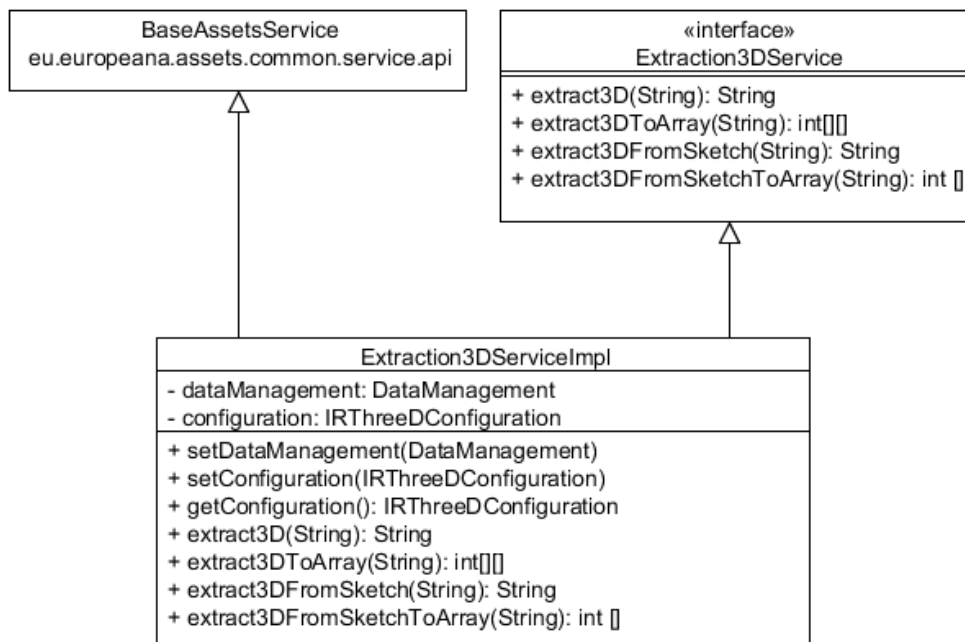
#### 3D extraction service interfaces

Service Name	Extraction3DService
--------------	---------------------



Responsibility	Extraction of the 3D low-level features out of the 3D objects or hand-drawn sketches
Provided Interfaces	<p><b>String extract3D(String url)</b>  <b>throws Extraction3DException</b></p> <p>Extracts 3D low-level feature vector from a 3D model, given by its url. The 3D model should be in one of the following formats: VRML, 3DS, OBJ, OFF, X3D. It returns an xml representation of the extracted 3D low-level descriptors.</p> <p><b>int[][] extract3DToArray(String url)</b>  <b>throws Extraction3DException</b></p> <p>Extracts 3D low-level feature vector from a 3D model, given by its url. The 3D model should be in one of the following formats: VRML, 3DS, OBJ, OFF, X3D. It returns the extracted 3D low-level descriptors in a 2-dimensional array of integers.</p> <p><b>String extract3DFromSketch(String url)</b>  <b>throws Extraction3DException</b></p> <p>Extracts 3D low-level feature vector from a hand-drawn sketch, given by its url. The sketch should be saved as a .png file. The size of the sketch should be either 100x100 or 400x400. It returns an xml representation of the extracted 3D low-level descriptors.</p> <p><b>int [] extract3DFromSketchToArray(String url)</b>  <b>throws Extraction3DException</b></p> <p>Extracts 3D low-level feature vector from a hand-drawn sketch, given by its url. The sketch should be saved as a .png file. The size of the sketch should be either 100x100 or 400x400. It returns the extracted 3D low-level descriptors in an array of integers.</p>
Dependencies	ASSETS Common

In the following Figure, the class diagram of the 3D Extraction service is shown.

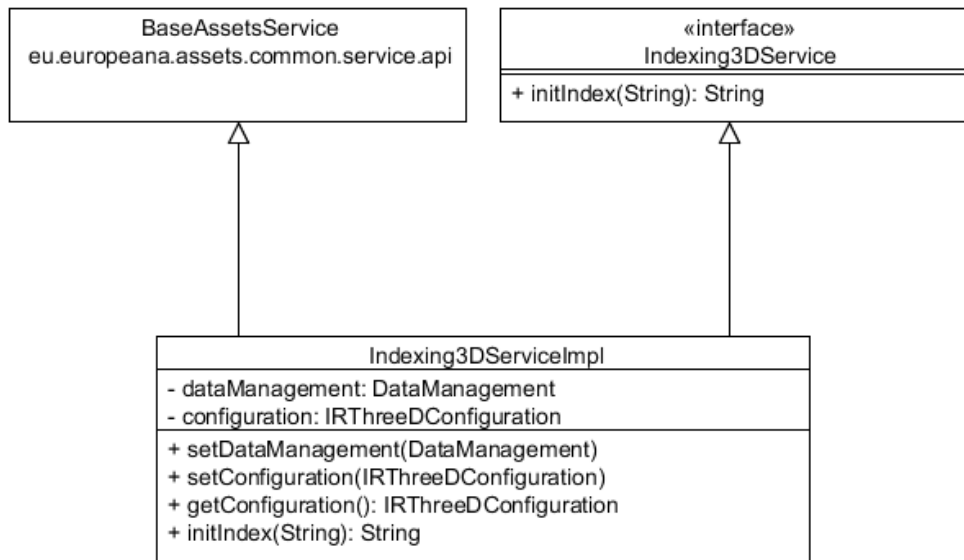


- *extract3D* passes the url of a 3D model to the extractor; the results are returned as an XML string.
- *extract3DToArray* does the same, with the difference that the results are returned as a two-dimensional int array.
- *extract3DFromSketch* passes the url of a hand-drawn sketch to the extractor; the results are returned as an XML string.
- *extract3DFromSketchToArray* does the same, with the difference that the results are returned as an int array.

### 3D indexing service interfaces

Service Name	Indexing3DService
Responsibility	Creation of the 3D index
Provided Interfaces	<b>String initIndex(String objectsFile)</b> <b>throws Indexing3DException</b> Initializes the 3D index and insert all objects that are listed along with their metadata and 3D descriptors into the index.
Dependencies	ASSETS Common

In the following Figure, the class diagram of the 3D Indexing service is shown.



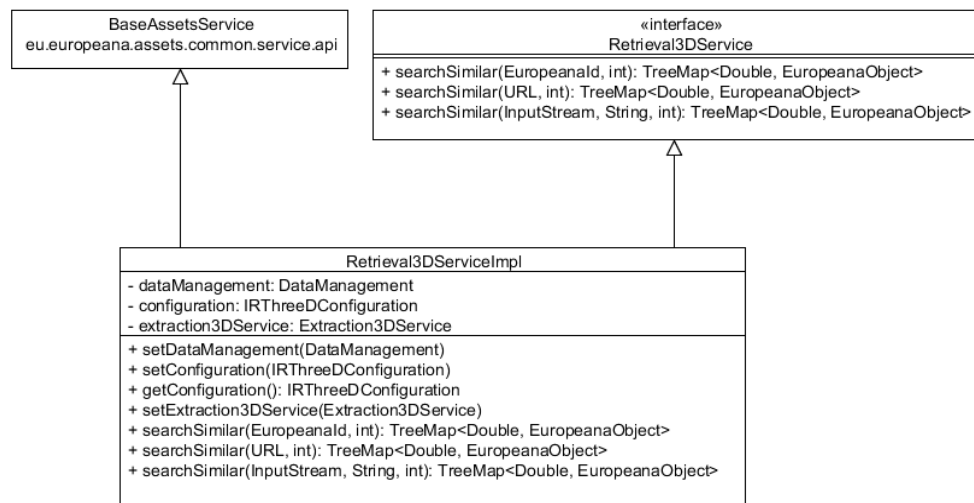
*initIndex()* creates a new 3D index by passing the URL of a file that contains the URLs of the new 3D models to be indexed.

### 3D retrieval service interfaces

Service Name	Retrieval3DService
Responsibility	Returns results, similar to a given query, to the user.
Provided Interfaces	<p><b>TreeMap&lt;Double, EuropeanaObject&gt;</b>  <b>searchSimilar(EuropeanaId queryID, int numOfResults)</b>  <b>throws Retrieval3DException,</b>  <b>Indexing3DException,</b>  <b>Extraction3DException</b></p> <p>Searches for 3D models similar to a model given by its EuropeanaId.</p> <p><b>TreeMap&lt;Double, EuropeanaObject&gt;</b>  <b>searchSimilar(URL modelURL, int numOfResults)</b>  <b>throws Retrieval3DException,</b>  <b>Indexing3DException,</b>  <b>Extraction3DException</b></p> <p>Searches for 3D models similar to a model given by its URL. If the query is a 3d model, it should be in one of the following formats: VRML, 3DS, OBJ, OFF, X3D. If the query is a hand drawn sketch, it should be a monochrome PNG file of size either 100x100 or 400x400.</p> <p><b>TreeMap&lt;Double, EuropeanaObject&gt;</b>  <b>searchSimilar(InputStream modelFile,</b></p>

	<p><b>String extension,</b> <b>int numOfResults)</b></p> <p><b>throws Retrieval3DException,</b> <b>Indexing3DException,</b> <b>Extraction3DException</b></p> <p>Searches for 3D models similar to a model given by its input stream. If the query is a 3d model, it should be in one of the following formats: VRML, 3DS, OBJ, OFF, X3D. If the query is a hand drawn sketch, it should be a monochrome PNG file of size either 100x100 or 400x400.</p>
<b>Dependencies</b>	ASSETS Common

In the following Figure, the class diagram of the 3D Retrieval service is shown.



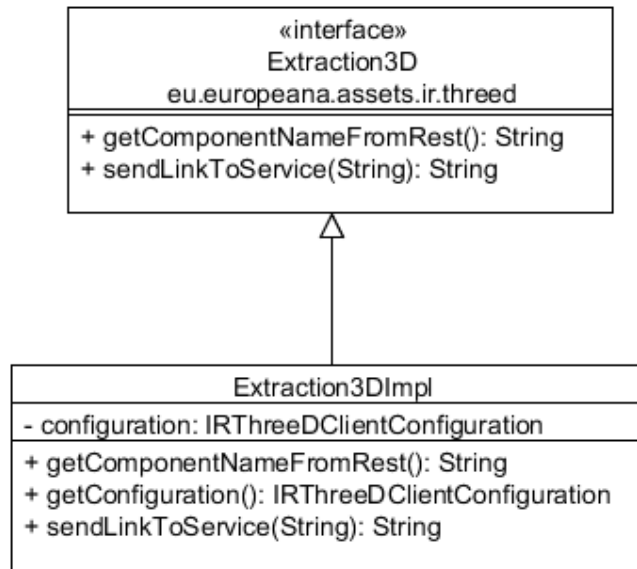
The following query forms can be used:

- a 3D model id (EuropeanaId);
- the input stream of a 3D model;
- a 3D model URL or a hand-drawn sketch URL.

The results are ranked by similarity.

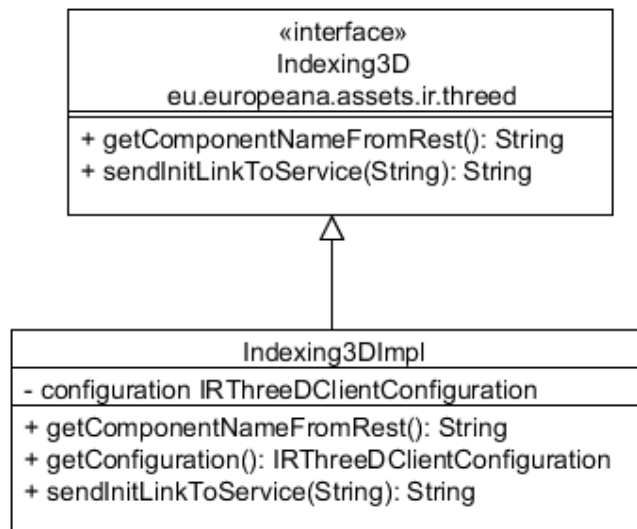
### **3D search and retrieval client interfaces**

In the following Figure, the class diagram of the 3D Extraction client is shown.



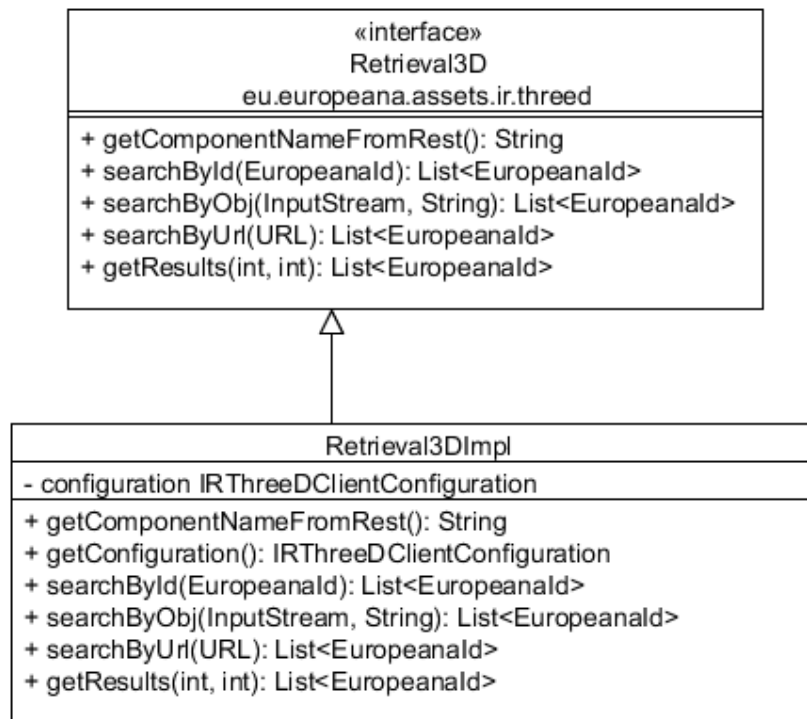
*sendLinkToService()* sends the 3D model file URL to the extractor and receives the extracted features as an XML string.

In the following Figure, the class diagram of the 3D Indexing client is shown.



*sendInitLinkToService()* sends the URL of a file containing 3D model URLs to the index creator.

In the following Figure, the class diagram of the 3D Retrieval client is shown.



The following query forms can be used:

- a 3D model id (Europeanald);
- the stream of a 3D model;
- a 3D model URL or a hand-drawn sketch URL.

By calling *getResult()* the system returns the results of the 3D search ranked by similarity.

### 3.2.3 Software Packaging

The 3D search and retrieval framework consists of two parts:

- the 3D search and retrieval services, packaged in a war file
- the 3D search and retrieval clients, packaged in a jar file

The services package also contains the 3D index.

### 3.2.4 Installation and configuration

The installation of the 3D search and retrieval framework is straightforward. The services war package needs to be deployed on a web server/servlet container like Apache Tomcat or Jetty.

Two folders need to be accessible for the service: the index folder and the temp (temporary) folder. The index folder contains the 3D index in the form of files, as they appear in the following list:





The temp folder is initially empty and serves for storing the uploaded queries and other feature extraction related files. This folder needs to be accessible both for reading and writing.

After the installation, the configuration file *assets-ir-threed.properties* needs to be updated in order to point to the correct location (paths) of the two folders.

The initial configuration may look like:

```
threed_index_path = ./services/ir-threed/src/main/resources/index/
threed_content_folder= ./services/ir-threed/src/main/resources/temp/
```

Since this package structure will not be available on the production server, the paths need to be updated, in order to point to the correct location:

```
threed_index_path = ./ir-threed/index/
threed_content_folder = ./ir-threed/
```

### 3.2.5 Scientific foundations

The description of the scientific foundations, on which the 3D search and retrieval framework relies, is out of the scope of this deliverable. The feature extraction algorithm is based on [1], while the index structure used in the framework is based on [2].

[1] P. Daras, A. Axenopoulos: A 3D Shape Retrieval Framework Supporting Multimodal



Queries. International Journal of Computer Vision, Springer, Volume 89, Issue 2, 2010-09  
[2] G. Amato, P. Savino: Approximate similarity search in metric spaces using inverted files.  
In: Proceedings of the 3rd International Conference on Scalable Information Systems  
(InfoScale 2008), pp. 1–10. ICST (2008)

### 3.3 User Manual

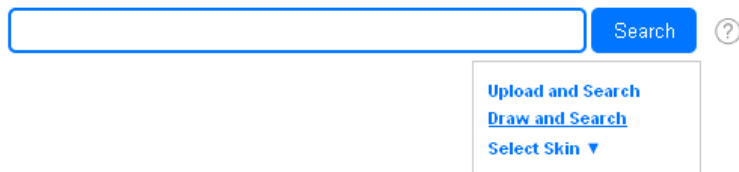
In the following sections, the functionality of the 3D search and retrieval framework will be demonstrated through an example of a search request in the Assets portal (<http://assetsdemo.atc.gr/portal/>).

#### 3.3.1 Search by hand-drawn sketch

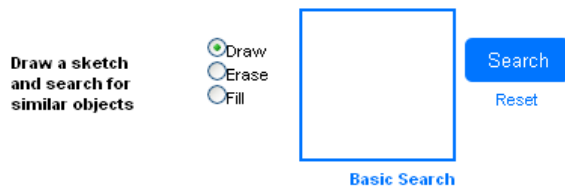
The user wishes to draw a sketch and search for similar 3d models. This can be done through the dedicated interface on the portal:

#1 Open the ASSETS portal

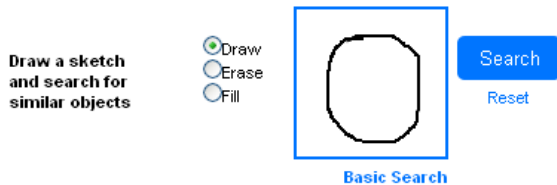
#2 Click on “Options” / “Draw and Search”



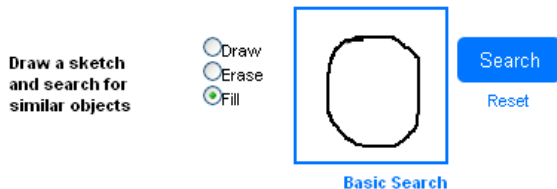
#3 The following interface appears:



#4 Draw shape



#5 Choose “Fill” option

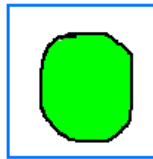


#6 Click on area to be filled and fill shape



Draw a sketch  
and search for  
similar objects

- Draw
- Erase
- Fill



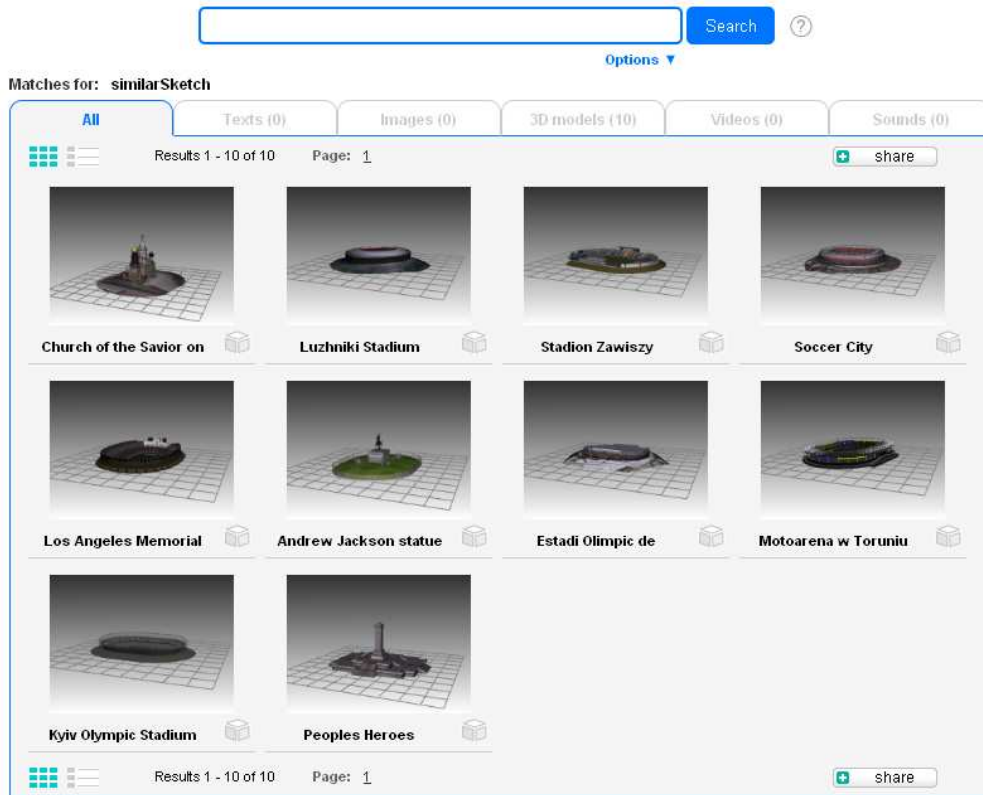
Search

Reset

Basic Search

#7 Press Search button and search for similar 3D model

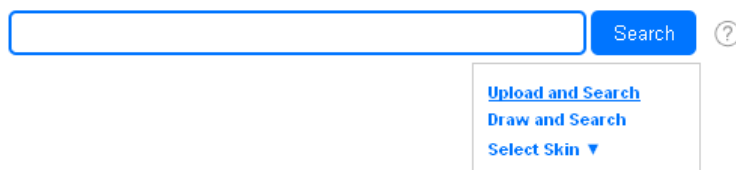
#8 The search in the Assets portal returns 10 results



### 3.3.2 Search by uploaded 3d model

#1 Open the ASSETS portal

#2 Click on "Options" / "Upload and Search"



#3 The following interface appears:



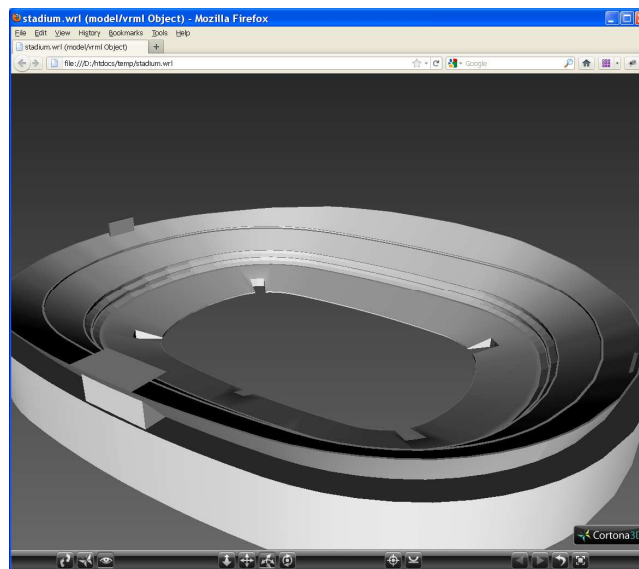
**Upload File**  
Upload URL

Basic Search

#4 Click on the “Browse” button and choose a query 3d model file from local hard-disk or click on the “Upload URL” link and give the URL of a 3d model file to be placed into the input field.

In this example, the 3d model “stadium.wrl” was used as a query, which is accessible through the following address:

<http://www.europeanlabs.eu/svn/assets/trunk/services/ir-threeed/src/test/resources/stadium.wrl>



Remark: Valid 3D model file extensions for this service are: wrl, x3d, 3ds, obj, off

#5 The search in the Assets portal returns 10 results

Matches for:

All
Texts (0)
Images (0)
3D models (10)
Videos (0)
Sounds (0)

Results 1 - 10 of 10 Page: 1 share

Ataturk Olimpiyat Stadi	King Fahd International	Kyiv Olympic Stadium	Los Angeles Memorial
Stadion Slaski/Harodowy	Olympiastadion Berlin	Stadion Narodowy /	Stadion Zawiszy
Cape Town Stadium	Motoarena w Toruniu		

Results 1 - 10 of 10 Page: 1 share

### 3.3.3 Search by id

In any of the result pages of the last two examples, the user can click on the “Similar search” button, located under all the 3d results and search for 3d models similar to that result.

As an example, we continue after step #5 of the last section:

#6 Click on the “Similar search” () of the result “Olympiastadion Berlin”

Matches for:

[All](#)
[Texts \(0\)](#)
[Images \(0\)](#)
[3D models \(10\)](#)
[Videos \(0\)](#)
[Sounds \(0\)](#)

Results 1 - 10 of 10 Page: 1 [share](#)

**Olympiastadion Berlin**  
 Google 3D warehouse content provider 2011  
 Google 3D warehouse ASSETS

[+](#)
[-](#)
[↻](#)
[★](#)
[i](#)

Results 1 - 10 of 10 Page: 1 [share](#)

#18 The search in the Assets portal returns 10 results

Similar to:

[Options ▼](#)

[All](#)
[Texts \(0\)](#)
[Images \(0\)](#)
[3D models \(10\)](#)
[Videos \(0\)](#)
[Sounds \(0\)](#)

Results 1 - 10 of 10 Page: 1 [share](#)

**Olympiastadion Berlin**
**Kaftanzoglio stadio in**
**Stadion Zawiszy**
**Kyiv Olympic Stadium**

**Estadi Olympic de**
**Kosevo Stadium in**
**Soccer City**
**Athens Olympic Stadium**

**Motoarena w Toruniu**
**King Fahd International**

Results 1 - 10 of 10 Page: 1 [share](#)



### 3.4 Concluding Remarks

The 3D search and retrieval framework was demonstrated during the first year review and published in the Europeana ThoughtLabs:

<http://pro.europeana.eu/web/guest/thoughtlab/new-ways-of-searching-and-browsing>

## 4. Audio indexing and retrieval

---

### 4.1 Introduction

The service has two main objectives:

- Providing a service for the automatic description and content enrichment of music audio data,
- Providing a service for fast indexing and retrieval of music audio content.

Music search core engine will listen to, understand and interpret music like humans do. It will describe audio in musical and perceptual terms like rhythm (bpm, percussiveness), harmony (key, mode, chords), timbre (instrumentation, production qualities), genre of the song, and even moods ('blue', 'party', 'furious', 'happy'... amongst others).

The audio description is combined with editorial information (artist, release date, country) as well as information distilled from blogs, reviews, or other external data. This supports implementation of a hybrid music search.

The search engine will then base the search criteria on metadata extracted from the song and its editorial content and will allow the users to search inside the collections for "music that sounds like Pixies" or to query music like "what are the tracks from this repository which sound like Chet Baker" or to select "I want relaxed jazzy music".

Once the user has reduced the scope of search via editorial metadata or audio descriptors search, the service will provide the means for retrieving similar tracks to a given track so to obtain more user relevant results. The search must scale to millions of audio segments and have a simple yet powerful API.

### 4.2 Software Requirements Overview

#### 4.2.1 Functionality overview

The basic functionality of the audio service is the similarity search based on either an uploaded track or an existing track in the catalog.

Please, see the comprehensive list of use cases displayed below:

1. Upload a music track or a track url and search for similar tracks,
2. Select one of the existing audio tracks and search for similar tracks,
3. Retrieve an audio description of a given track,
4. Search by audio descriptors.

#### 4.2.2 Requirements

**Usability:** The service should be self-explanatory and easy to use even from a non-expert user.





**Reliability:** The audio search functionality should have a 24/7 availability and work every time within acceptable time limits for a user search. The returned results should be musically similar to the queried track.

**Performance:** The feature should have no adverse effects on Europeana.eu's web site performance.

**Look & Feel:** Font and colors should be in line with Europeana brand guidelines.

**Layout and Navigation:** Search similar button, upload interface, audio descriptors.

**Communications Interfaces:** The interface will use stateless HTTP for the communication with other services or clients.

**Licensing:** The ir-audio uses a java library, jElla, which complies with the licence used by Europeana artifacts - EUPL. However, this java library invokes functionality of the ella service which is BMAT's copyrighted. This back-end interface is REST based and supports well-known standards like HTTP, XML and JSON.

**Documentation:** Documentation for developers will be delivered as code documentation and as a detailed document outlining the architecture and the interaction between the used components.

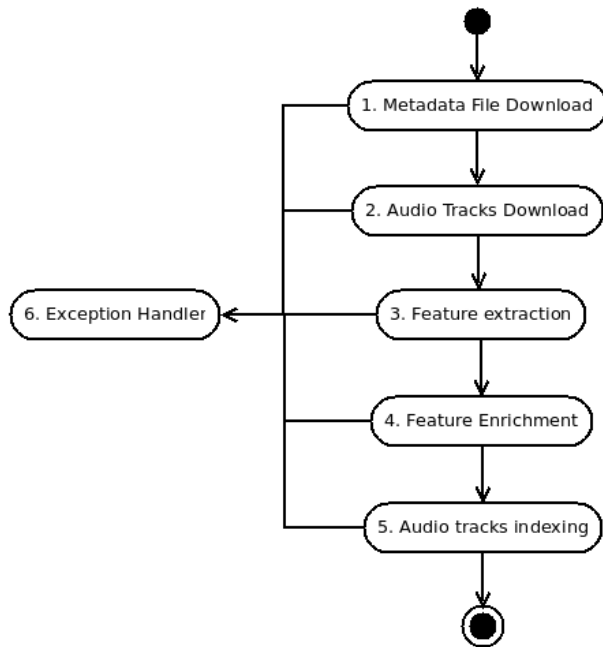
## 4.3 Technical Documentation:

### 4.3.1 Audio Service Use Cases

The main use case we considered for the audio service are: indexing and retrieval services.

#### *Indexing use case*

The indexing use case explains the indexation process an audio track undergoes when ingested on assets system. The main point here is that audio editorial metadata is enriched with data features coming from audio itself through Music Information Retrieval algorithms.



**Figure 10 Audio indexing Use case**

***Retrieval use case***

The retrieval use case explains the retrieval (search) process both when searching based on a track which is already in the assets system or based on a uploaded track sample. Results are based on music similarity to the seed track.

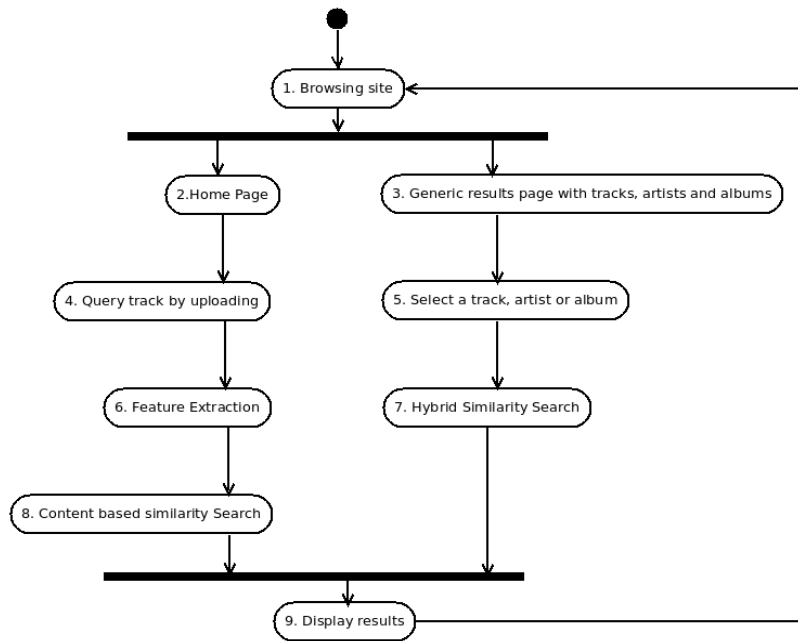


Figure 11 Audio Searching Use case

### 4.3.2 Audio Class Diagrams

#### Audio Indexing Class Diagram

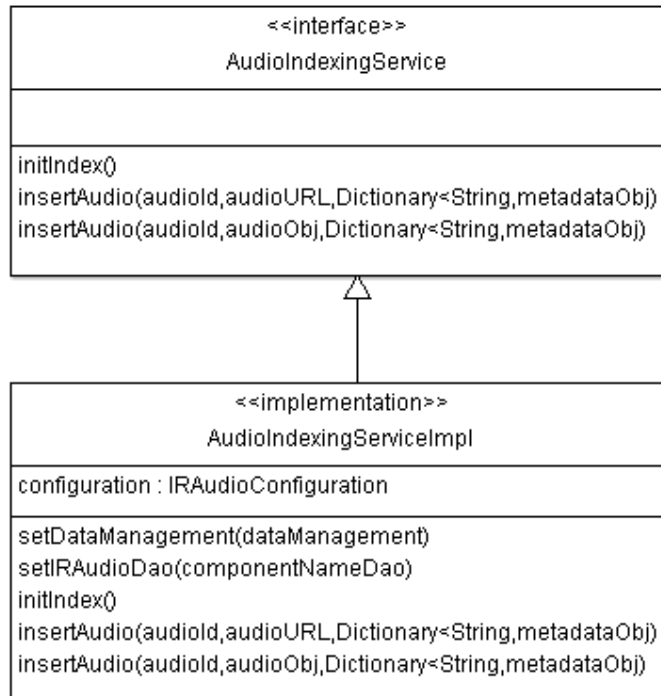


Figure 12 Audio Indexing Class Diagram

#### Audio Searching Class Diagram

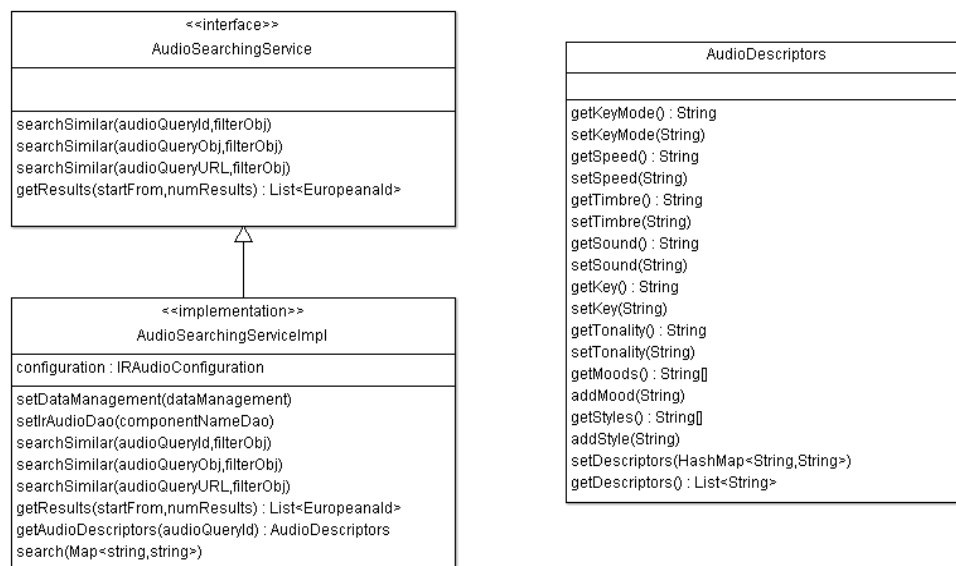


Figure 13 Audio Searching Class Diagram

**Domain object**

QueryResultsMarshaller
collectionObjects : List<EuropeanIdAdapter>
getEuropeanIdAdapter() : List<EuropeanIdAdapter> setEuropeanIdAdapter(List<EuropeanId>)

**Figure 14 Audio Domain Object**

**4.3.3 Service APIs: Rest Interfaces**

Method	Response type	Name	Parameters	Function
GET	XML/JSON	<b>Component</b>		Returns the component display name
GET	XML/JSON	<b>searchById</b>	@audioQueryId id of the audio to be searched by.	It returns a set of musically similar songs to the seed audio.
GET	XML/JSON	<b>Search</b>	@q a query string including a set of audio descriptors with the value to be searched (e.g. mood:happy)	It returns those tracks which matched the search.
GET	XML/JSON	<b>getAudioDescriptors</b>	@audioQueryId id of the audio whose audio description is to be retrieved	It returns the audio descriptors of a given audio. Typically, this are displayed in the interface (detail view).
GET	XML/JSON	<b>searchByUrl</b>	@audioQueryUrl http url of a remote audio to be search by	It returns a set of musically similar songs to the seed audio.
POST	XML/JSON	<b>searchByObj</b>	@audioFile Audio Binary data audio stream to be searched by.	It returns a set of musically similar songs to the seed audio.

**4.3.4 Service APIs: Client Interfaces**

**Audio Searching Service Interfaces**

Service Name	AudioSearchingService
Responsibility	Provides methods associated to audio similarity search as well as audio description search.



Provided Interfaces	<p><b>void initIndex() throws AudioIndexingException</b></p> <p>Index initialization</p> <p><b>void insertAudio(EuropeanId audioId, URL audioURL, Dictionary&lt;String,String&gt; metadataObj) throws AudioIndexingException</b> Inserts the audio content coming via URL for indexation .</p> <p><b>void insertAudio(EuropeanId audioId, InputStream audioObj, Dictionary&lt;String,String&gt; metadataObj) throws AudioIndexingException;</b></p> <p>Submits the audio content coming via InputStream for indexation .</p>
Dependencies	ASSETS Common

**Figure 15 Audio Searching Service Interfaces**

***Audio Indexing Service Interfaces***

Service Name	AudiIndexingService
Responsibility	Provides methods associated to audio similarity search as well as audio description search.
Provided Interfaces	<p><b>AudioDescriptors getAudioDescriptors(EuropeanId audioQueryId) throws AudioSearchingException</b></p> <p>Given an existing track id, it returns its audio descriptors.</p> <p><b>void search(Map&lt;String,String&gt; filterObj) throws AudioSearchingException</b></p> <p>Searches within the collection by audio descriptor.</p> <p><b>void searchSimilar(EuropeanId audioQueryId, Map&lt;String,String&gt; filterObj) throws AudioSearchingException</b></p> <p>Given an existing track id, it returns similar tracks.</p> <p><b>void searchSimilar(InputStream audioQueryObj, Map&lt;String,String&gt; filterObj) throws AudioSearchingException</b></p> <p>Given an existing track InputStream (usually not in the collection – Query by Example), it returns similar tracks.</p>

	<p><b>void searchSimilar(URL audioQueryURL, Map&lt;String,String&gt; filterObj) throws AudioSearchingException</b></p> <p>Given an existing track url (usually not in the collection – Query by Example), it returns similar tracks.</p>
Dependencies	ASSETS Common

**Figure 16 Audio Indexing Service Interfaces**

### 4.3.5 Software Packaging

The audio search service is provided as follows:

1. Ir-audio service is packaged in a war library.
2. Ir-audio client is packaged in a jar library.
3. Ir-audio uses a jar library implemented for ASSETS and called jElla that wraps the calls to BMAT's Audio search engine.
4. Installation instructions for audio search engine and audio feature extractor, (which are both stand-alone) are documented below.

### 4.3.6 Audio feature extractor installation

#### Technical Requirements

- Linux RedHat (Centos) 5.x , Linux Debian Lenny , Ubuntu,
- Pentium Dual Core 1.86 GHz,
- RAM Memory : 4GB\*,
- Storage space : 100GB\*.

\*Note: Storage and Memory requirements depends on the amount of tracks.

#### Installation Instructions for Redhat/Centos 5.X

1. Most dependencies will be automatically resolved by yum. Some of the packages might not be available in the default package repositories. So, rpmforge should be added to the list of repositories and the packages for tbb and fftw3f should be installed:

```
wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
sudo rpm -Uvh rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
sudo yum localinstall fftw3-3.1.2-5.el5.x86_64.rpm --nogpgcheck
sudo yum localinstall tbb-2.0-4.20070927.x86_64.rpm --nogpgcheck
```

2. Install the extractor:

```
sudo yum localinstall ella-extractor-x.y.z-1.x86_64.rpm --nogpgcheck
```

#### 4.3.6..1 Redhat/Centos 5.5 onwards



1. Most dependencies will be automatically resolved by yum. Some of the packages might not be available in the default package repositories. So, rpmforge should be added to the list of repositories and the packages for tbb and fftw3f should be installed.

```
wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
sudo rpm -Uhv rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
sudo yum localinstall fftw3-3.1.2-5.el5.x86_64.rpm --nogpgcheck
sudo yum localinstall tbb-2.2-1.20090809.x86_64.rpm --nogpgcheck
sudo yum install imlib2
sudo yum localinstall libavutil49-0.5.2-33.el5.x86_64.rpm --nogpgcheck
```

2. Install the extractor:

```
sudo yum localinstall ella-extractor-1.1.0-1.x86_64.rpm --nogpgcheck
```

3. Redhat 5.5 provides ffmpeg 0.6 but ella-extractor requires ffmpeg 0.5. Therefore, remove ffmpeg-0.6 and dependencies:

```
sudo rpm -e ffmpeg --nodeps
sudo rpm -e x264 --nodeps
sudo rpm -e libavutil49 --nodeps
```

4. And install the following packages:

```
sudo rpm -i libtheora-1.0alpha8-1.x86_64.rpm
sudo rpm -i x264-0.0.0-0.5.20090708.el5.rf.x86_64.rpm
sudo rpm -i ffmpeg-0.5.2-2.el5.rf.x86_64.rpm
```

### ***Installation instructions for debian/ubuntu***

1. You have to install the following dependencies.

```
sudo apt-get install ffmpeg libboost-filesystem1.34.1 libboost-program-options1.34.1 libcrypto++7 libcurl3 libfftw3-3 libsamplerate0 libsndfile1 libtag1c2a libtbb2 libyaml-0-1
```

2. Install the extractor:

```
sudo dpkg -i ella-extractor_x.y.z_amd64.deb
```

NOTE: The Ella extractor requires an Internet connection to validate its license in order to work. Also make sure to exchange the x.y.z with the current version of the packageOperation

Just Execute:

```
ella-extractor path/to/the/input_audiotrack path/to/the/output_signature
```





### 4.3.7 Audio Search Engine installation and configuration

#### Technical requirements

- RedHat (CentOS) 5.3+ / Debian 5 (Lenny) / Ubuntu
- Pentium Dual Core 1.86 GHz
- RAM Memory : 4GB\*
- Storage space : 100GB\*

\*Note: Storage and Memory requirements depends on the amount of tracks

#### Installation Instructions

##### Debian 5 (Lenny)

1. Install dependencies:  
sudo aptitude install curl  
sudo aptitude install python  
sudo aptitude install python-central  
sudo aptitude install python-yaml  
sudo aptitude install libglib2.0-0  
sudo dpkg -i libqtcore4\_4.5.1-2\_amd64.deb  
sudo dpkg -i libqt4-network\_4.5.1-2\_amd64.deb  
sudo dpkg -i \*gaia2\*.deb  
sudo aptitude install openjdk-6-jre  
sudo dpkg -i pylucene\_2.3.1-1.1\_amd64.deb  
sudo aptitude install python-mako  
sudo aptitude install libapache2-mod-wsgi  
sudo aptitude install python-formencode  
sudo dpkg -i python-webpy\_0.310-1\_all.deb  
sudo aptitude install python-simplejson  
sudo dpkg -i python-jsonpickle\_0.2.0.trunk.20090715\_all.deb  
sudo dpkg -i python-mimerender\_0.2.3\_all.deb  
sudo aptitude install python-sqlalchemy
2. Install Ella:  
sudo dpkg -i python-ella-core\_2.\*.deb ella-ws\_2.\*.deb

##### Ubuntu

Just like Debian. After installation, do execute the following:

```
sudo pycentral rtinstall python2.6
```

##### Red Hat Enterprise Linux 5 / CentOS 5

Note: Ella is not compatible with SELinux. Instructions for disabling it can be found at this link [[http://www.crypt.gen.nz/selinux/disable\\_selinux.html](http://www.crypt.gen.nz/selinux/disable_selinux.html)]. Also, the default security settings for Postgres are too restrictive. Password authentication must be enabled in `/var/lib/pgsql/data/pg_hba.conf` by adding the following lines right before `# "local"` is for Unix domain socket connections only:

```
# allow localhost connections to ella database
```



```
host ella ella 127.0.0.1/32 password sameuser
```

1. Add the RPMForge repository to your "yum" sources ( see <http://rpmforge.net/> for further information ):

```
wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
sudo rpm -Uhv rpmforge-release-0.3.6-1.el5.rf.x86_64.rpm
```

2. Install dependencies from official repositories:

```
sudo yum install httpd libyaml python-yaml python-simplejson curl
```

3. Install 3rd party software (packaged by BMAT):

```
sudo yum --nogpgcheck localinstall python-jsonpickle-0.2.0-1.noarch.rpm
sudo yum --nogpgcheck localinstall python-mako-0.2.4-1.noarch.rpm
sudo yum --nogpgcheck localinstall python-webpy-0.31-2.1.noarch.rpm
sudo yum --nogpgcheck localinstall PyLucene-2.3.2.1-3.x86_64.rpm
sudo yum --nogpgcheck localinstall mod_wsgi-2.1-2.el5.x86_64.rpm
sudo yum --nogpgcheck localinstall qt45-4.5.2-1.el5.pp.x86_64.rpm
sudo yum --nogpgcheck localinstall python-sqlalchemy-0.4.9-1.x86_64.rpm
sudo yum --nogpgcheck localinstall python-mimerender-0.2.2-1.noarch.rpm
sudo yum --nogpgcheck localinstall python-formencode-1.0.1-1.x86_64.rpm
sudo yum --nogpgcheck localinstall libyaml-0.1.2-3.el5.kb.x86_64.rpm
```

Note: libyaml rpm installation might fail due to newer package in the repositories. In this case, please do:

```
sudo rpm -Uhv --force libyaml-0.1.2-3.el5.kb.x86_64.rpm
```

4. Install BMATs recommendation engine:

```
sudo yum --nogpgcheck localinstall libgaia2*
```

5. Install Ella:

```
sudo yum --nogpgcheck localinstall python-ella-core-2.2*.x86_64.rpm ella-ws-2.2*.x86_64.rpm
```

NOTE: you may install it more quickly by doing `sudo yum --nogpgcheck localinstall \*.rpm` over the rpm files provided by BMAT.

## Configuration

1. The data received from BMAT is one compressed tarball for the data (containing similarity information and metadata) and one .py file (configuration) per collection. A md5sum for all files is also provided to validate data consistency.

1a. The tarballs must be uncompressed in `/var/lib/ella/collections`. The usual directory layout is the following:



```

/var/lib/ella/
`-- collections
   |-- CUSTOMER_timestamp
   |   |-- datasets
   |   |-- enriched_index
   |-- CUSTOMER -> CUSTOMER_timestamp (this is a symbolic link)
   |-- TAGS_timestamp
   |   |-- datasets
   |   |-- index
   `-- TAGS -> TAGS_timestamp (this is a symbolic link)

```

In order to create the symbolic links, you may do the following:

```

In -s /var/lib/ella/collections/CUSTOMER_timestamp
/var/lib/ella/collections/CUSTOMER
In -s /var/lib/ella/collections/TAGS_timestamp /var/lib/ella/collections/TAGS

```

1b. The provided .py configuration files must be copied to /etc/ella/collections.

NOTE: Minimal setup will comprise two collections: a "tags" collection and a "customer" collection. However, another collection called BMAT may be provided to offer recommendation based on artists/tracks not available in the "customer" collection.

NOTE: Please, comment out the database part in the config.py if you don't require User Recommendation.

```

# database
# database = dict(
#   connection = 'mysql://ella:ella@localhost/ella'
#)

```

2. [OPTIONAL: Only required if you make use of user recommendation] Setup your favorite, sqlalchemy-compatible database (tested with Postgres and MySQL), install their python connectors (python-psycopg2 for postgres and MySQL-python for mysql) and start the server.

Note: If the db server does not run on the server where ELLA is installed, then the following sql commands will have to be executed by hand on the DB:

### 2.a. Create the database

For PostgreSQL:

```

sudo -u postgres psql -c "CREATE USER ella WITH PASSWORD 'ella';"
sudo -u postgres psql -c "CREATE DATABASE ella;"
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE ella TO ella;"

```

For MySQL:

```

mysql -u root -p -e "CREATE USER 'ella'@'localhost' IDENTIFIED BY 'ella';"
mysql -u root -p -e "CREATE DATABASE ella DEFAULT CHARACTER SET utf8;"
mysql -u root -p -e "GRANT ALL PRIVILEGES ON ella.* TO ella;"

```

2.b. initialize the database model (SCHEME is 'postgres' or 'mysql')



```
python -m ella.core.db.model create 'SCHEME://ella:ella@localhost/ella'
```

2.c. set the connection string in `/etc/ella/config.py` accordingly (SCHEME is 'postgres' or 'mysql')

```
sudo vi /etc/ella/config.py (set database connection to  
'SCHEME://ella:ella@localhost/ella')
```

### Operation

It is important to install Ella on a clean server since some problems may arise if conflicting apache configurations are present.

1. Before starting up the server, please make sure that the configuration is completed.
2. Then, execute:

```
(redhat/centos) sudo /etc/init.d/httpd restart  
(debian) sudo /etc/init.d/apache2 restart
```

3. Please, note that the first call to ELLA will last because it takes sometime to load all the necessary datasets.
4. You may test it by executing the following query such as:

```
curl localhost/ella-ws/collections/tags/search?q=rock
```

NOTE 1: replace \$CUSTOMER by your collection name

NOTE 2: In order to execute the above, curl application is needed. You may install it by doing:

```
sudo yum install curl
```

4. if the previous query fails, probably there is a configuration issue. See the troubleshooting section.

### Troubleshooting

1. If the test query fails, please try the following:

```
sudo tail -f /var/log/httpd/* # or sudo tail -f /var/log/apache/* if you use debian  
[enter][enter][enter][enter]  
curl localhost/ella-ws/collections/tags/search?q=rock
```

You'll notice in the error logs what went.

2. If the problem persists, please contact [support@bmat.com](mailto:support@bmat.com) and attach the following information:

- `/var/log/httpd/*.log` (see above) or `/var/log/apache/*.log`
- `/etc/httpd/*` or `/etc/apache/*`
- `/etc/ella/*`
- `tree /var/lib/ella/collections/ > collections_tree.txt`
- Query that fails to be executed.

### 4.3.8 Scientific foundations

Here are some of the references to papers and PhD which are behind the music search provided by BMAT:

1. Cano, P., Koppenberger, M., and Wack, N. (2005). Content-based music audio



- recommendation. In ACM International Conference on Multimedia (ACMMM), pages 211{212.
2. Cano, P., Celma, O., Koppenberger, M., and Buldu, J. M. (2006). The topology of music recommendation networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*. <http://arxiv.org/abs/physics/0512266v1>.
  3. Cano, P., Koppenberger, M., and Wack, N. (2005). Content-based music audio recommendation. In ACM International Conference on Multimedia (ACMMM),
  4. Celma, O. PhD Thesis: Music Recommendation and Discovery in the Long Tail (2008)
  5. Bogdanov, D., Serra, J., Wack, N., and Herrera, P. (2010). Hybrid music similarity measure. Music Information Retrieval Evaluation eXchange (MIREX) Abstract
  6. Bogdanov, D., Serra, J., Wack, N., and Herrera, P. (2009). From low-level to high-level: Comparative study of music similarity measures. In Inter- national Workshop on Advances in Music Information Research (AdMIRe), Co-Located with the IEEE International Conference on Multimedia and Expo (ICME).
  7. Bogdanov, D., Serra, J., Wack, N., and Herrera, P. (2010). Hybrid music similarity measure. Music Information Retrieval Evaluation eXchange (MIREX) Abstract.

## 4.4 User Manual

### 4.4.1 Audio Search by existing track

The user searches for “Eurovision” and obtains several items (which include video and audio items). The icons indicate whether it is an audio, video, image or 3d model...

(for developers only) Example relative URL: [http://assetsdemo.atc.gr/portal/brief-doc.html?start=1&view=table&assets=http%3A%2F%2Fi2.ytimg.com%2Fvi%2FmOUYgPmLK0k%2Fdefault.jpg&query=similarAudio\\_http%3A%2F%2Fwww.europeana.eu%2Fresolve%2Frecord%2F92001%2FD219B15D18424E5616DB5A0B606C0AED6D200005](http://assetsdemo.atc.gr/portal/brief-doc.html?start=1&view=table&assets=http%3A%2F%2Fi2.ytimg.com%2Fvi%2FmOUYgPmLK0k%2Fdefault.jpg&query=similarAudio_http%3A%2F%2Fwww.europeana.eu%2Fresolve%2Frecord%2F92001%2FD219B15D18424E5616DB5A0B606C0AED6D200005)

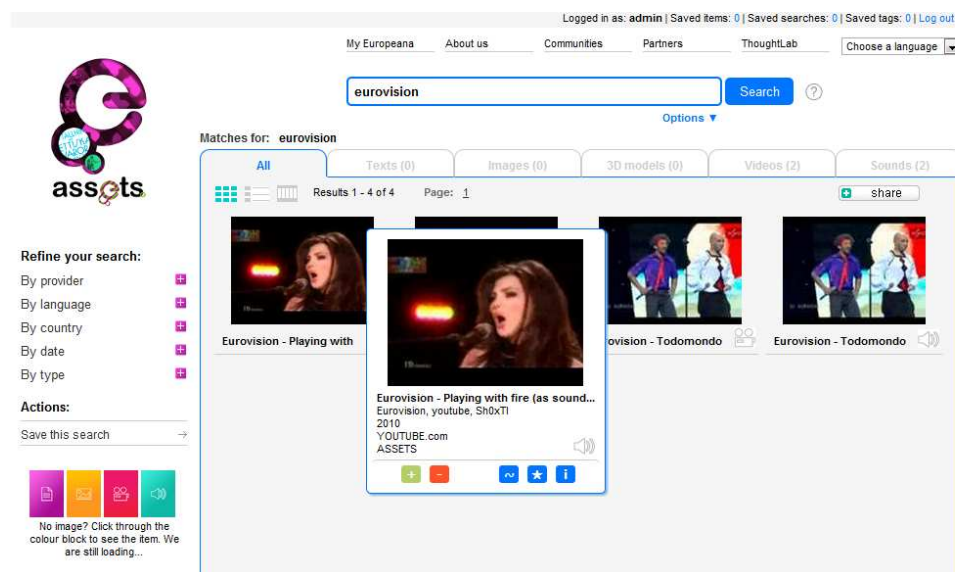



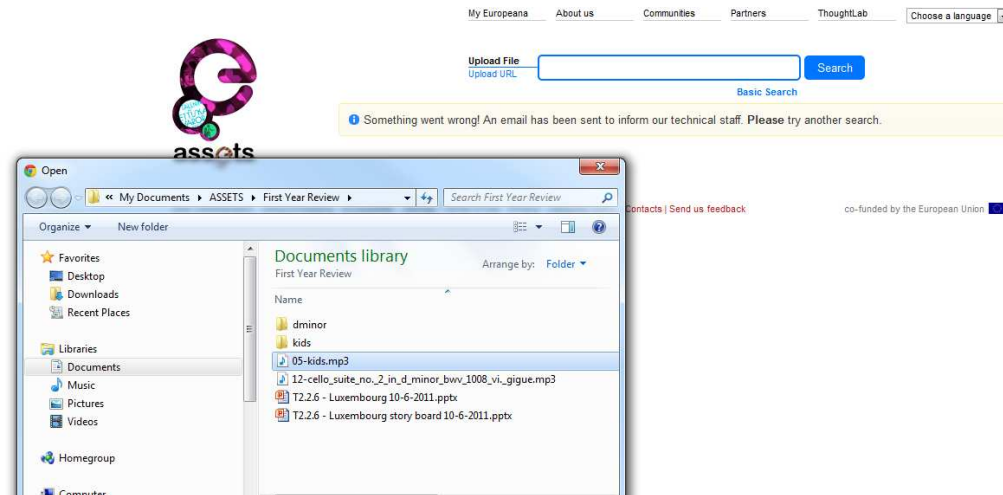
Figure 17 Audio Search by Existing Track

The icon  indicates that the retrieved content is an audio track. In order to search similar tracks, click on the search similar (i.e. ~) symbol.

### 4.4.2 Audio Search by uploading track

In order to perform an example “Upload and search” query:

- 1) mouse over the options text beneath the search box and drop down menu will be displayed.
- 2) Click on Upload and Search,
- 3) Click on “upload file” on the left hand side of the text box
- 4) Select an mp3, wav or wma file.
- 5) Click on the Search button:



**Figure 18 Audio Search by Uploading a track**

It will take some time but results will eventually appear.

#### **4.4.3 Audio Search by url**

In order to perform a search by url:

- 1) mouse over the options text beneath the search box and drop down menu will be displayed.
- 2) Click on Upload and Search,
- 3) Click on "upload url" on the left hand side of the text box
- 4) Enter (e.g.  
url )  
[http://audio.bmat.com/audio/2/b/bruce\\_springsteen/born\\_to\\_run/05-born\\_to\\_run.mp3](http://audio.bmat.com/audio/2/b/bruce_springsteen/born_to_run/05-born_to_run.mp3)
- 5) Click on the Search button:

(for developers only) Example relative URL: [http://assetsdemo.atc.gr/portal/brief-doc.html?query=uploadAudio&uploadSearchURL=http://audio.bmat.com/audio/2/b/bruce\\_springsteen/born\\_to\\_run/05-born\\_to\\_run.mp3](http://assetsdemo.atc.gr/portal/brief-doc.html?query=uploadAudio&uploadSearchURL=http://audio.bmat.com/audio/2/b/bruce_springsteen/born_to_run/05-born_to_run.mp3)

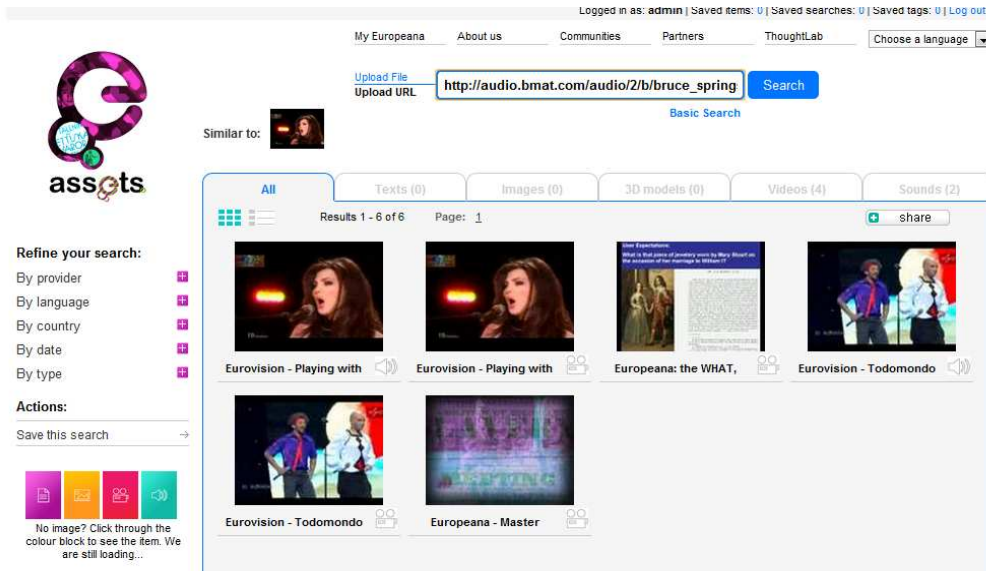


Figure 19 Audio Search by URL

#### 4.4.4 Audio Description

In order to see the audio descriptors:

- 1) Mouse over the audio item result
- 2) Click on the “i” button. It will display the complete track information, including the audio descriptors.

(for developers only) Example relative URL: [http://assetsdemo.atc.gr/portal/brief-doc.html?query=uploadAudio&uploadSearchURL=http://audio.bmat.com/audio/2/b/bruce\\_springsteen/born\\_to\\_run/05-born\\_to\\_run.mp3](http://assetsdemo.atc.gr/portal/brief-doc.html?query=uploadAudio&uploadSearchURL=http://audio.bmat.com/audio/2/b/bruce_springsteen/born_to_run/05-born_to_run.mp3)

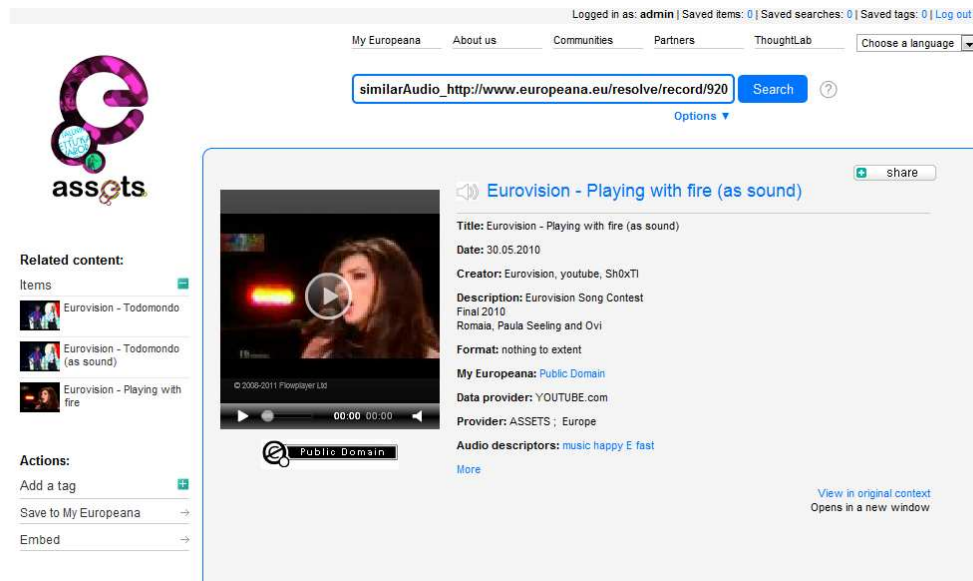


Figure 20 Audio Description





#### 4.4.5 Audio Search by Audio Description

In order to search by audio descriptors:

- 1) Mouse over the audio item result
- 2) Click on the “i” button. It will display the complete track information, including the audio descriptors.
- 3) Click on any of the audio descriptors in order to search by one of these (see figure above). In the example below, we clicked on “happy”:

(for developers only) Example relative URL: [http://assetsdemo.atc.gr/portal/brief-doc.html?start=1&view=table&assets=&query=audioDescriptorSearch\\_mood\\_happy](http://assetsdemo.atc.gr/portal/brief-doc.html?start=1&view=table&assets=&query=audioDescriptorSearch_mood_happy)

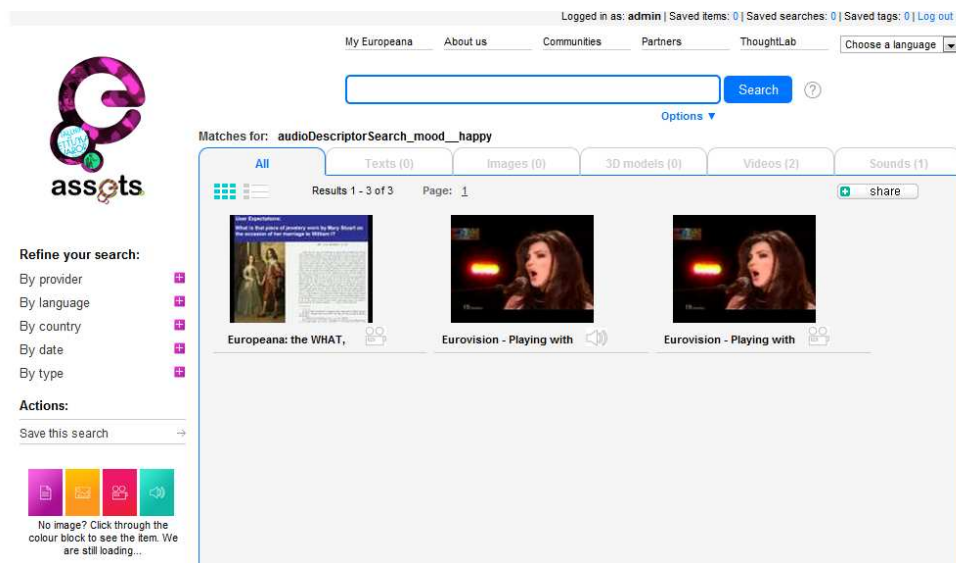


Figure 21 Audio Search by Audio Description

## 4.5 Concluding Remarks

The audio search and retrieval service was demonstrated on the first year review of the ASSETS project and was tested for usability within the user evaluation.

## 5. Video summarization, adaptation, indexing and retrieval

---

### 5.1 Software Requirements Overview

#### 5.1.1 Requirements

**Usability:** The video services are provided through a Web User Interface, which should be self-explanatory and easy to use, even for basic users.

**Reliability:** The video services should be deployed in a high availability server with, at least, 4 GB of principal memory. The user should recognize the summarized videos and find results as conceptually similar to the original videos.

**Performance:** Video summarization and indexing are time consuming operations and are executed in background. Video search is a faster operation that can be executed online.

**Look & Feel:** Web pages layout should be in line with the Europeana web application guidelines.

**Applicable standards:** The video service is able to process MPEG-2 and MPEG-4 video profiles, including Flash Videos. The video search service is able to process also images in JPEG, PNG, GIF and BMP format.

**Documentation:** The interfaces of the services include standard Javadoc comments. This document includes additional conceptual-level comments.

#### 5.1.2 Use cases

##### *Use case for video summarization*

The actors are:

- User.
- Europeana Web Interface.
- Europeana Video Service.

The flow of events is as follows (see Figure 22):

1. The Europeana Video Service summarizes videos in background.
2. The User searches for a Europeana video item.
3. The Europeana Web Interface shows a list of videos that match the search criteria.
4. The User indicates that s/he wants to see a summary of a video item.
5. The Europeana Web Interface retrieves the selected video from the Europeana Video Service along with the keyframes of the summarized video.
6. The Europeana Web Interface shows the video summary to the User.
7. The User selects a keyframe.
8. The Europeana Web Interface skips the video to the position of this keyframe.

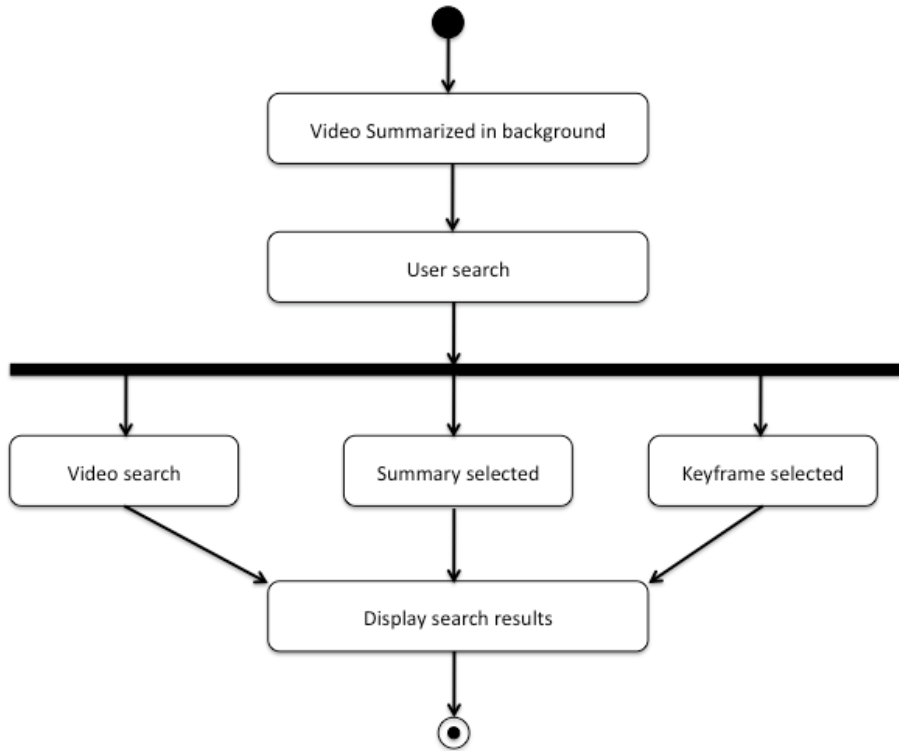


Figure 22: Video summarization use case

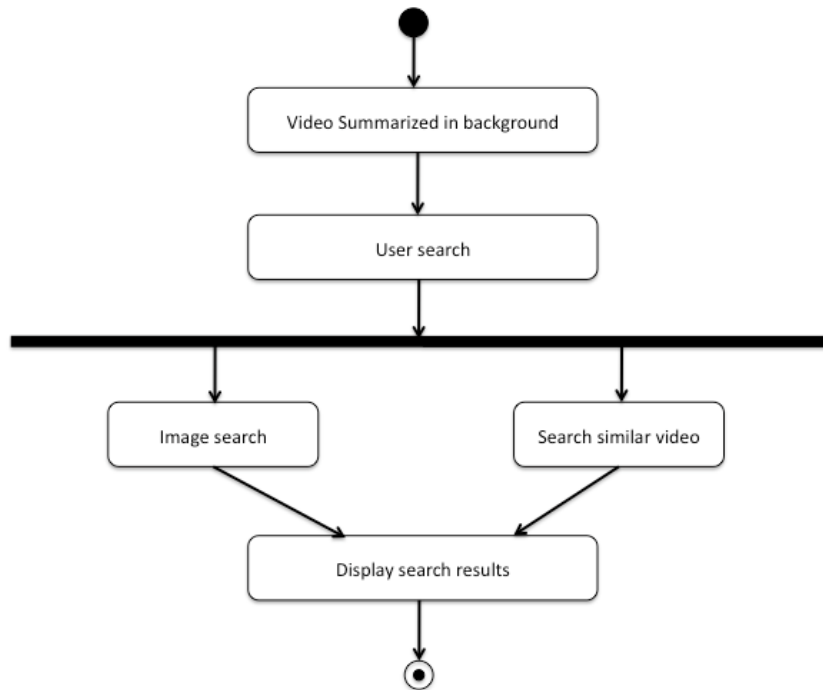


Figure 23: Image similarity search use case

### ***Use case for image similarity search***

The actors are:

- User.
- Europeana Web Interface.
- Europeana Video Service.

The flow of events is as follows (see Figure 23):

1. The Europeana Video Service indexes videos in background.
2. The User searches for a Europeana image.
3. The Europeana Web Interface shows a list of videos that match the search criteria.
4. The User indicates that s/he wants to search for videos similar to a specific Europeana image.
5. The Europeana Web Interface transfers the query to the Europeana Video Service.
6. The Europeana Video Service returns a list of similar videos ordered by similarity.
7. The Europeana Web Interface shows the User a list of videos that are similar to the provided image.

### ***Use case for video similarity search***

The actors are:

- User.
- Europeana Web Interface.
- Europeana Video Service.

The flow of events is as follows:

1. The Europeana Video Service indexes videos in background.
2. The User searches for a Europeana video.
3. The Europeana Web Interface shows a list of videos that match the search criteria.
4. The User indicates that s/he want to search for videos similar to a specific video item.
5. The Europeana Web Interface transfers the query to the Europeana Video Service.
6. The Europeana Video Service returns a list of similar videos ordered by similarity.
7. The Europeana Web Interface shows the User a list of videos that are similar to the provided video.

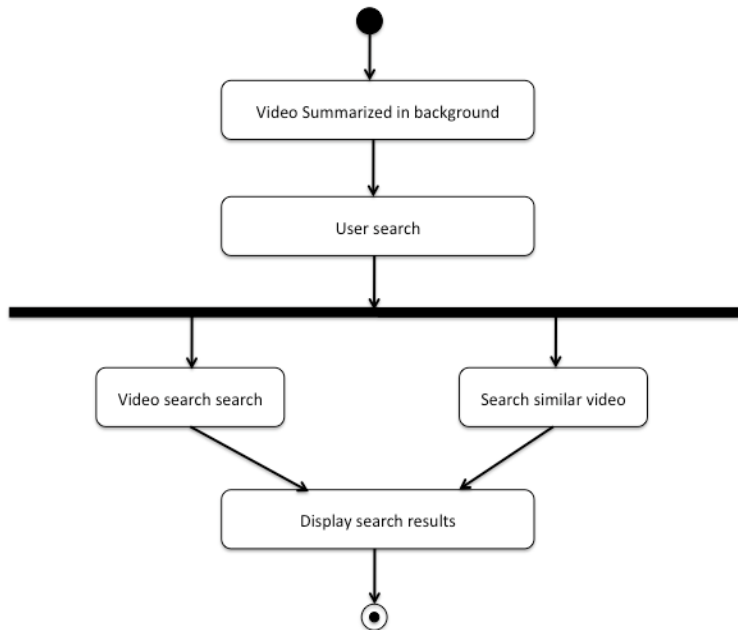


Figure 24: Video similarity search

## 5.2 Technical Documentation:

### 5.2.1 UML Diagrams

#### Service description

Figure 25 shows the UML class diagram of the domain object for the summarization service. The method *createSummarizedVideo()* of the *VideoSummarizationService* class is intended to index the video and return a corresponding *VideoData* object. The same original video can be summarized several times with different values in the percentage parameter. Each instance of the *VideoData* class represents a summarization request. This data is associated to the *Europeanald* object by means of the *getOriginalVideo()* method. The *getVideoSummary()* and *getStoryboard()* methods do not create new *Europeanald* instances but a URL or a collection of *KeyFrame* objects, respectively. The *KeyFrame* class contains information for the timestamp of the keyframe in the original video. The timestamp units are milliseconds and 0 means the beginning of the video. The *KeyFrame* class also stores the JPEG copy of the keyframe.

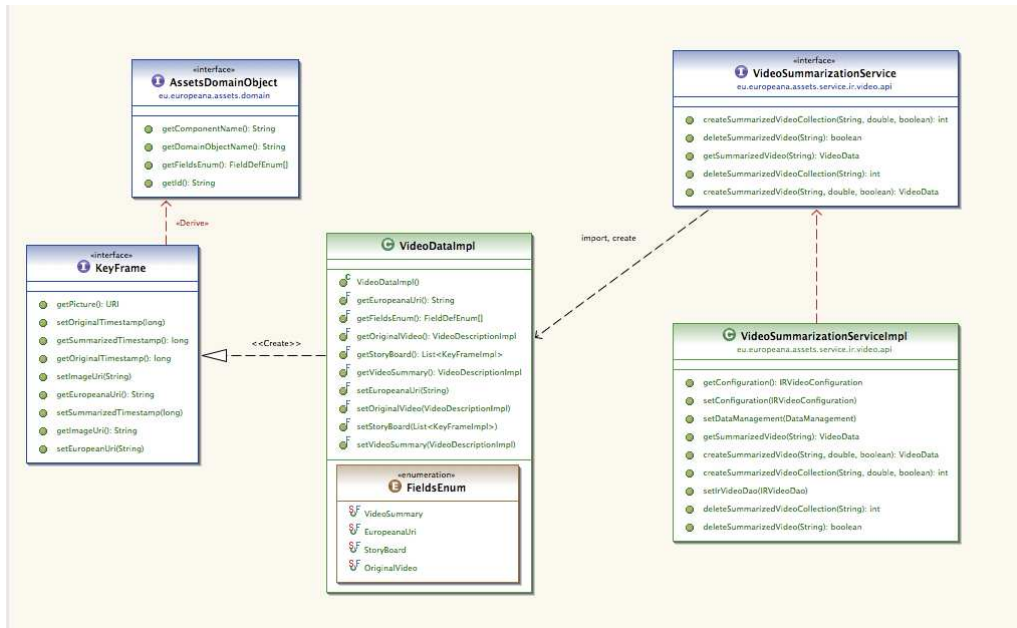


Figure 25: Video summarization client model

Figure 26 shows the UML classes diagram for the indexing and retrieval service. The *getVideosSimilarToXXX()* methods of the *IRVideoService* class are intended to obtain a collection of visually similar videos (represented as instances of the *SimilarityRanking* class). The operation receives as a parameter a *EuropeanId* object that can be either an image or a video. The visually similar videos the operation returns are obtained from those ones that the module has previously analyzed and indexed (i.e., the videos that were received through the *indexVideo()* method).



Figure 26: Similarity search client model

### 5.2.2 Service APIs: REST interfaces

The following subsections list the REST interfaces for video indexing and retrieval. Below we provide an example of these REST interfaces invocation:

GET <http://localhost:8983/assets/ir-video/summarization/rest/summarizedvideo?europeanaUriStr=http://www.europeana.eu/resolve/record/07794/337F0F2113547F5F8BB28B41432A93F324B312B0>

In this example, "GET" is the posting method, "/summarizedvideo" is the path or the service and "europeanaUriStr=..." is a query parameter.

In addition to using the unit tests, these services can be tested using the service's index.html page (see Figure 27).

Video indexing and retrieval service			
Method	Response type	Path	Function
<b>Video Summarization</b>			
GET	XML/JSON	<a href="#">/assets/ir-video/summarization/rest/component</a>	Display the name of the current component
GET	XML/JSON	<a href="#">/assets/ir-video/summarization/rest/summarizedvideo</a>	This operation retrieves a previously summarized video from the MongoDB. If the summary is not found, it returns null
POST	XML/JSON	<input type="button" value="POST"/>	This operation creates a summarized video that is stored in the MongoDB. If the video summary already exists, the method recreates the video summary.
<b>Video Indexing and Retrieval</b>			
GET	XML/JSON	<a href="#">/assets/ir-video/ir/rest/component</a>	Display the name of the current component
POST	XML/JSON	<input type="button" value="POST"/>	This operation indexes the provided video.
GET	XML/JSON	<a href="#">/assets/ir-video/ir/rest/similarvideotourimage</a>	Search videos similar to provided image
GET	XML/JSON	<a href="#">/assets/ir-video/ir/rest/similarvideotoeuropeanaimage</a>	Search videos similar to provided europeana image
GET	XML/JSON	<a href="#">/assets/ir-video/ir/rest/similarvideotoeuropeanavideo</a>	Search videos similar to provided europeana video

Figure 27: Video service's index.html page

The summarization service provides methods to summarize a video item and retrieve a previously summarized video.

Table 3 gathers the information needed to call the video REST summarization services. Prefix path of the service is: **/assets/ir-video/summarization/rest**.

Method	Response type	Name	Parameters	Function
GET	XML/JSON	<b>getComponentName</b>		Returns the component name
GET	XML/JSON	<b>getSummarizedVideo</b>	<b>@europeanaUriStr</b> , The video to be retrieved	This method retrieves a previously summarized video from the MongoDB.



POST	XML/JSON	<b>createSummarizeVideo</b>	<b>@europeanaUriStr</b> , The video to be summarized	This method summarizes a video and produces a video summary that is stored in the MongoDB
------	----------	-----------------------------	---	---

**Table 3: REST video summarization services**

Table 4 gathers the information needed to call the video REST indexing services. Prefix path of the service is: **/assets/ir-video/ir/rest**.

Method	Response type	Name	Parameters	Function
GET	XML/JSON	<b>getComponentName</b>		Returns the component name
POST	XML/JSON	<b>indexVideo</b>	<b>@europeanaUriStr</b> , The video to be indexed	The method calculates the visual similarity indexes of the video so that it can be compared to the indexes of other videos (for search purposes). MPEG-1, MPEG-2 and FLV videos are currently supported
GET	XML/JSON	<b>getVideoSimilarToURLImage</b>	<b>@url</b> , An image URL	This operation retrieves a list of videos that are similar to the image file indicated in the url parameter. The visually similar videos that this operation returns are obtained from those that our module has previously analyzed and indexed  The visually similar videos that this operation returns are obtained from those that our module has previously analyzed and indexed (using <code>indexVideo()</code> ).
GET	XML/JSON	<b>getVideoSimilarToEuropeanImage</b>	<b>@europeanaUriStr</b> , The image to be used as search pattern.	This operation retrieves a list of videos that are similar to the EuropeanId image indicated in the europeanaUriStr parameter. The visually similar videos that this operation returns are obtained from those that our module has previously analyzed and indexed. The visually similar videos that this operation



				returns are obtained from those that our module has previously analyzed and indexed (using <code>indexVideo()</code> ).
GET	XML/JSON	<b>getVideoSimilarToEuropeanVideo</b>	<b>eupeanaUriStr</b> , The video to be used as search pattern.	This operation retrieves a list of videos that are similar to the European video indicated in the <code>eupeanaUriStr</code> parameter. The visually similar videos that this operation returns are the result of aggregating the visual similarity of the keyframes of the video indicated in the <code>eupeanaUriStrVideo</code> parameter. The visually similar videos that this operation returns are obtained from those that our module has previously analyzed and indexed (using <code>indexVideo()</code> ).

Table 4: REST video indexing services

### 5.2.3 Services APIs: Client interfaces

The ir-video module provides the following services through the corresponding interfaces.

<b>Service Name</b>	Video Summarization
<b>Responsibility</b>	The generation of reduced length versions of original videos and extraction of representative keyframes.
<b>Provided Interfaces</b>	<ul style="list-style-type: none"> <li>▪ <i>VideoSummarizationService</i></li> <li>▪ <i>VideoData</i></li> </ul>
<b>Dependencies</b>	ASSETS Common

<b>Interface Name</b>	<i>VideoSummarizationService</i>
<b>Key Concepts</b>	<i>EuropeanId</i>
<b>Operations</b>	<ul style="list-style-type: none"> <li>▪ <i>createSummarizedVideo()</i> summarizes a video and produces a video summary that is stored in the MongoDB. The caller can optionally indicate the desired percentage for the video summary.</li> <li>▪ <i>createSummarizedVideoCollection()</i> summarizes a video collection and produces the video summaries that are stored in the MongoDB.</li> </ul>



	<ul style="list-style-type: none"> <li>▪ <i>deleteSummarizedVideo()</i> deletes the video from the MongoDB.</li> <li>▪ <i>deleteSummarizedVideoCollection()</i> deletes the summarized video collection.</li> <li>▪ <i>getSummarizedVideo()</i> retrieves a previously summarized video from the MongoDB.</li> </ul>
--	--

<b>Interface Name</b>	<i>VideoData</i>
<b>Key Concepts</b>	<i>URL, KeyFrame</i>
<b>Operations</b>	<ul style="list-style-type: none"> <li>▪ <i>getOriginalVideo()</i> gets the description of the original video.</li> <li>▪ <i>getVideoSummary()</i> gets the description of the summarized video.</li> <li>▪ <i>getStoryboard()</i> Returns a collection of <i>KeyFrame</i> objects according to the percentage that was indicated for the video summary. Each <i>KeyFrame</i> instance is a representative picture of the video along with its timestamp in the original video.</li> </ul>

<b>Service Name</b>	Video Indexing and Retrieval
<b>Responsibility</b>	Indexing and searching for videos based on the visual similarity.
<b>Provided Interfaces</b>	<ul style="list-style-type: none"> <li>▪ <i>IRVideoService</i></li> </ul>
<b>Dependencies</b>	ASSETS Common

<b>Interface Name</b>	<i>IRVideoService</i>
<b>Key Concepts</b>	<i>EuropeanId</i>
<b>Operations</b>	<ul style="list-style-type: none"> <li>▪ <i>indexVideo()</i> calculates the visual similarity indexes of the video so that it can be compared to the indexes of other videos (for search purposes). The method does not return any information but stores the <i>EuropeanId</i> parameter and the indexes in the internal database.</li> <li>▪ <i>indexVideoCollection()</i> indexes the indicated video collection</li> <li>▪ <i>deleteIndexedVideo()</i> deletes the indicated indexed video.</li> <li>▪ <i>deleteAllIndexes()</i> deletes all video indexes.</li> <li>▪ <i>getVideoSimilarToURLImage()</i> retrieves a ranked list of videos that are similar to a given example. The visually similar videos that this operation returns are obtained from those that our module has previously analyzed and indexed (using <i>indexVideo()</i>).</li> <li>▪ <i>getVideoSimilarToEuropeanImage()</i> retrieves a ranked list of videos that are similar to a given example.</li> <li>▪ <i>getVideoSimilarToEuropeanImageStream()</i> retrieves a ranked list of videos that are similar to a given example.</li> <li>▪ <i>getVideoSimilarToEuropeanVideo()</i> retrieves a ranked list of videos that are similar to a given example.</li> </ul>

## 5.2.4 Installation and configuration

### Memory requirements

The video summarization module requires certain memory. In our internal tests it exceeds 2 GB of memory. Windows 32 allocates a maximum of 2GB for the user space. This means that 32 bit JVMs are not eligible for executing our module. You need a 64 bit JVM.

### Additional software

For video summarization, the Xuggler library should be installed. The following instructions are for the official Ubuntu Linux environment of the ASSETS project. The installation process could differ for other environments. Section 0 references sources with additional installation instructions.

### Prerequisites

This section assumes that an instance of Apache Tomcat is installed. We also assume that CATALINA\_HOME environment variable points to the Tomcat directory.

### Installation Instructions

1.- Download and install Xuggler from <http://www.xuggler.com/xuggler/>. The installation script describes the environment variables to set up. In particular, be sure to setup the following environment variables:

- Configure XUGGLE\_HOME pointing to the Xuggler installation directory.
- Include XUGGLE\_HOME in the LD\_LIBRARY\_PATH environment variable.
- Include XUGGLE\_HOME in the PATH environment variable.

All of these variables are not required in all cases, but they will maximize the possibilities of making Xuggler work in different situations and environments.

2.- Copy the content of XUGGLE\_HOME/lib to CATALINA\_HOME/shared/lib folder. Create the destination folder, if necessary. The copied content contains all the shared libraries required by Xuggler

3.- Specify to Apache Tomcat where to look for shared libraries. Edit the CATALINA\_HOME/conf/catalina.properties configuration file and add the following lines (or append the specified folders if the lines already exist):

```
shared.loader=${catalina.home}/shared/lib,${catalina.home}/shared/lib/*.jar
```

```
java.library.path=/usr/local/xuggler/lib # Or wherever Xuggler libraries are
```

4.- Ensure that the environment variable LD\_LIBRARY\_PATH is 'visible' from Apache Tomcat. If not (or not sure), it can be done by adding the following lines in the startup.sh script for starting Apache Tomcat in Linux:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CATALINA_HOME/lib:$CATALINA_HOME/shared/lib
```

```
export LD_LIBRARY_PATH
```

6.- Be sure to include the required Xuggler's .jar files in the /WEB-INF/lib/ subfolder of your application when deploying it in Apache Tomcat.



## Additional notes

The first important thing to note about using JNI under Tomcat is that we cannot place the native libraries or their JNI interfaces under the WEB-INF/lib or WEB-INF/classes directories of a web application and expect to be able to reload the *webapp* without restarting the server. The class that calls *System.loadLibrary(String)* must be loaded by a *classloader* that is not affected by reloading the web application itself.

Then the Xuggler jars should be placed in the \$CATALINA\_HOME/shared/lib directory. Note that in Tomcat the \$CATALINA\_HOME/shared/lib directory may not exist. In this case, it has to be created .

The second important thing is that Tomcat does not take into account the \$CLASSPATH or \$DYLD\_LIBRARY\_PATH of the console. So you will need to add this to the \$CATALINA\_HOME/conf/catalina.properties:

```
shared.loader=${catalina.home}/shared/classes,${catalina.home}/shared/lib/*.jar
java.library.path=/usr/local/xuggler/lib
```

## Video content and summaries

In order to point out where the video indexing and retrieval module should search for video content, you have to configure the *Spring* configuration file named *assets-ir-video.properties*.

Usually this value is:

```
portal.content.folder=${content.folder}
```

which means that it has to search for content in the standard portal content folder.

In addition, the indexed videos are generated in a subfolder of the content folder named "generated".

## Video indexes

Video indexes are stored in two different places:

1.- In a folder that has to be specified in the *assets-ir-video.properties Spring* configuration file.

Usually this value is:

```
global.video.index.path=${video.globalindex.path}
```

2.- You need to create an additional PostgreSQL database for video indexes. For this purpose the following script has to be executed:

```
cd assets/services/ir-video/src/main/resources
make clean create populate
```

The make command can receive different options that configure its behavior:

```
make clean # Crop the DB
```

```
make create #Create the DB
```

```
make populate # Populate the tables of the existing DB
```

```
make dump # Dump the DB content into a file named dumped_test_data.sql
```

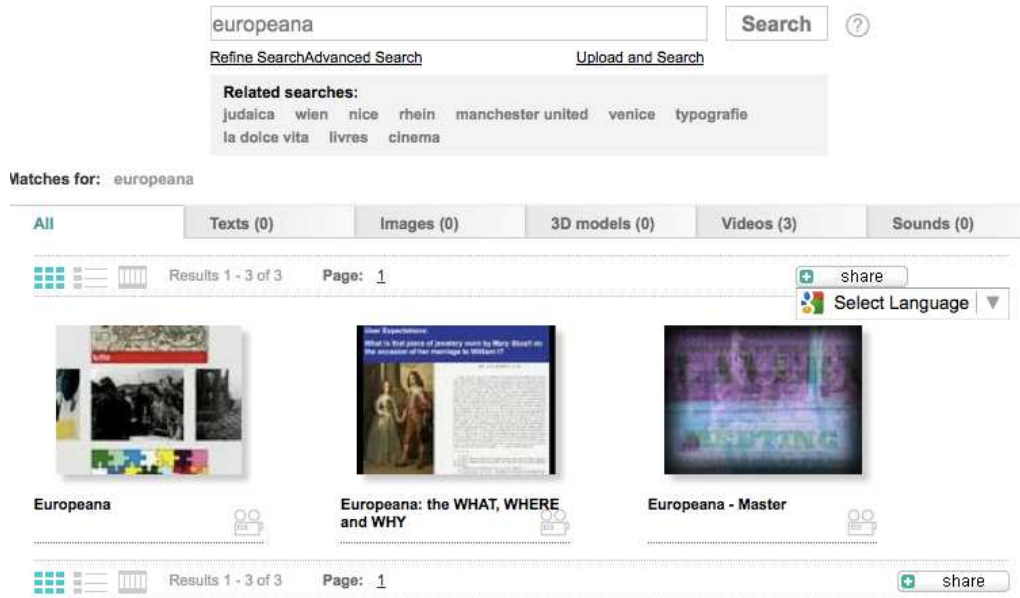


### 5.3 User Manual

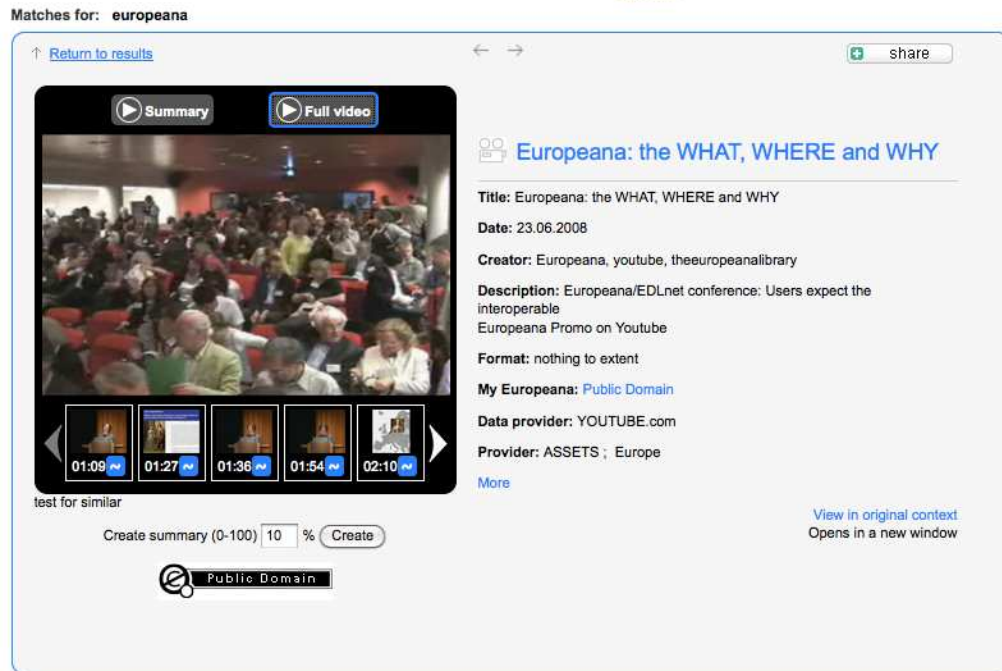
This section highlights the integrated video summarization, indexing and searching features, and demonstrates the usability improvements from the end-user point of view.

#### Video summarization

1) The user searches for "europeana" and obtains several video items. There is an icon at the bottom right of each item indicating that these items contain video. The user can also switch to the video tab, in which only video items are displayed.

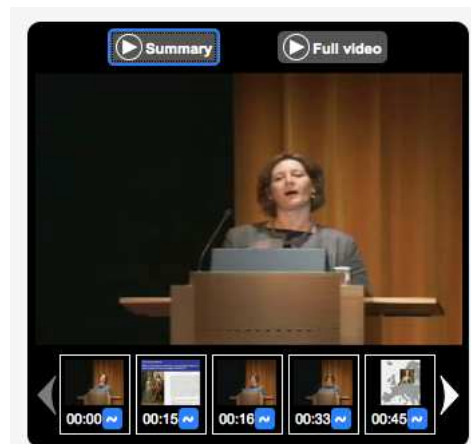


2) The user selects an item, in example we used the "Europeana: the WHAT, WHERE and WHY" object. He/she obtains the original video along with a storyboard to navigate the video. The storyboard contains the more significant parts of the video according to our summarization algorithm.



3) The user clicks on any keyframe and the video starts playing from that position.

4) The user presses the "Summary" button to obtain a summary of the original video. Our algorithm has removed redundant information in order to speed up video browsing.



### Video similarity search

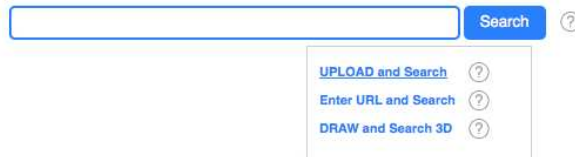
Once the video has been indexed, the user can search for videos in the portal whose content is similar to an example image.

1) The keyframes of the video have a "⋈" small button to search for videos similar to the selected keyframe.

In addition, the user can search for videos similar to an image that is not in the portal. In this case:



1) The user selects the "Upload and Search" option, which is below the main search box of the portal.



2) The user uploads an example image (.jpg, .jpeg, .gif, .png) from his/her local disk and moves the cursor over the "Search" button. Figure 19 shows the image used in this example.

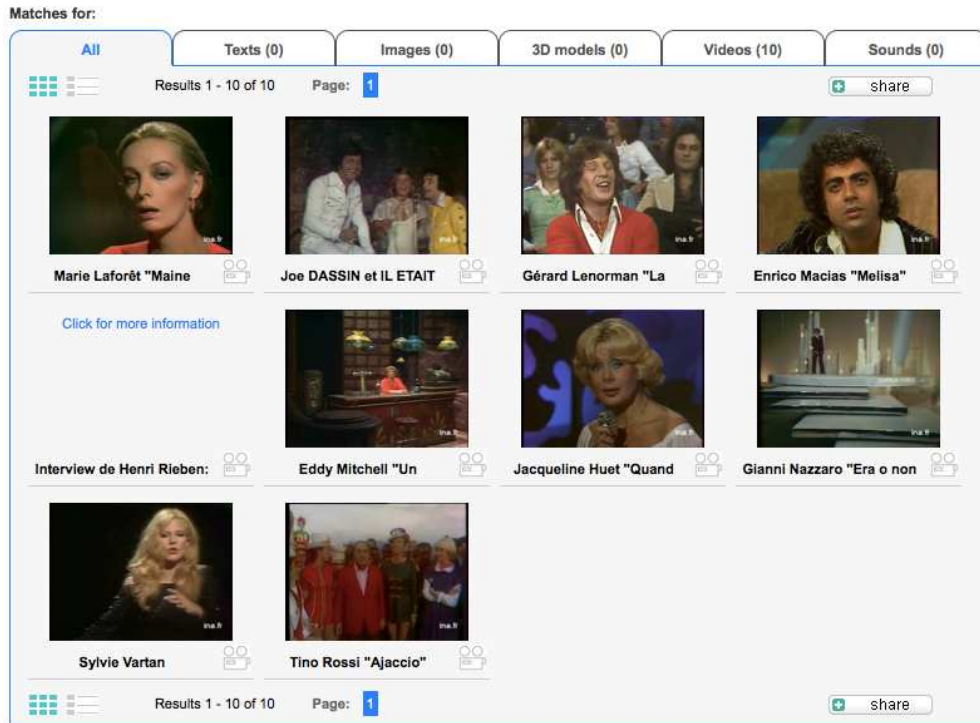


3) The user selects the "on videos" pop-up option.

4) The portal returns a list of videos that are similar to the example image.



Figure 28: Image used for the search



5) The user can also perform the operation using the "Upload URL" option. In this case an Internet URL of the example image has to be provided.



For instance, you can use the following URL:

<http://www.thebowesmuseum.org.uk/uploads/collections/fullsize/b-m-868.JPG>

## 5.4 Bibliography

[1] Apache Tomcat: <http://tomcat.apache.org>

[2] Xuggler: <http://www.xuggler.com/xuggler/>

[3] Additional information in the xuggle wiki:

[http://wiki.xuggle.com/Frequently\\_Asked\\_Questions#I\\_get\\_an\\_22UnsatisfiedLinkError.22\\_when\\_I\\_run\\_Xuggler-based\\_Applications\\_in\\_Tomcat](http://wiki.xuggle.com/Frequently_Asked_Questions#I_get_an_22UnsatisfiedLinkError.22_when_I_run_Xuggler-based_Applications_in_Tomcat)

[4] Additional information about Apache Tomcat and JNI:

[http://wiki.apache.org/tomcat/HowTo#I.27m\\_encountering\\_classloader\\_problems\\_when\\_using\\_JNI\\_under\\_Tomcat](http://wiki.apache.org/tomcat/HowTo#I.27m_encountering_classloader_problems_when_using_JNI_under_Tomcat)





## 6. Concluding Remarks

---

In this deliverable we have described the ASSETS services for the content-based indexing and ranking components, implemented and tested in ASSETS WP2.2.

The technical aspects of the following components have been explained in detail:

- Image indexing and retrieval service,
- 3D-model indexing and retrieval service,
- Audio indexing and retrieval service,
- Video summarization, adaptation, indexing and retrieval service.

The software requirements, the technical documentation (UML diagrams, services description and API documentation, the software packaging and installation), and the user manual have been provided for each service in order to allow developers to understand how to use these services, and to know the steps to follow during their installation and configuration process.