



UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA TRIENNALE IN INGEGNERIA
INFORMATICA

**Continuous Evaluation of Digital
Libraries:
A Software for Automatically
Assessing Performances**

Relatore:

Ch.mo Dr. Ing. Ferro Nicola

Laureando:

Bandiera Giuseppe

Padova, 28 Settembre 2012

Anno accademico 2011/2012

Ringraziamenti

Ringrazio i miei parenti più cari: in particolare mia sorella, mia madre e mio padre per il continuo sostegno nei miei confronti in questo periodo.

Ringrazio gli amici di Conegliano e di Padova per i momenti di svago passati assieme.

Infine desidero ringraziare la mepublisher per il rispetto del mio impegno in questo elaborato.

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
2 Evaluation of Digital Libraries Systems	3
2.1 Information Retrieval Technology	3
2.2 Actors Involved in Evaluation Campaigns	4
2.3 Stages of an Evaluation Campaign	5
2.4 The Purpose of the Thesis	7
2.5 Case Study: Europeana	7
3 Background	9
3.1 Network Architecture, with a Focus on HTTP	9
3.2 eXtensible Markup Language (XML)	11
3.3 Java 5 Concurrency: Callable and Future	13
4 Architecture and Functionalities	15
4.1 The Import of Topics	15
4.2 Launching an Experiment	15
5 Demonstration of the Functioning	19
6 The Source Code	25
6.1 An Overview	25
6.2 Main Package	26
6.2.1 Constructable	26

CONTENTS

6.2.2	Parsable	27
6.3	Europeana package	28
6.3.1	EuroQueryConstructor	28
6.3.2	EuroXMLParser	29
6.3.3	IDChanger	33
6.4	Application Package	34
6.4.1	Launcher	34
6.5	Util Package	42
6.5.1	HTTP	42
6.5.2	Item	44
6.5.3	Topic	45
6.5.4	TopicParser	46
6.5.5	DoubleHelper	49
6.6	Extension to Other DLS	50
6.6.1	Adding Support to Other Application-Layer Network Protocols	51
6.6.2	Expanding HTTP management class	51
6.6.3	Adding support to other Digital Library Systems	53
7	Conclusions	55

List of Figures

2.1	The logo of Europeana.	7
3.1	An example of XML tree.	13
4.1	Sequence diagram for the import of topics.	16
4.2	Sequence diagram for the launch of an experiment.	16
5.1	The main graphic interface of the program.	20
5.2	The import of topics.	20
5.3	Select the file with topics	21
5.4	The GUI updated after the import of topics.	21
5.5	Compiling fields.	22
5.6	The experiment is ready to be launched.	22
5.7	The experiment is being taken.	23
5.8	Experiment completed.	23
5.9	The output file.	24
6.1	The organization of the packages.	26

LIST OF FIGURES

List of Tables

3.1	The TCP/IP reference model.	9
3.2	Examples of protocols for each layer.	10
3.3	Some HTTP methods.	10
3.4	Some HTTP message headers.	11
3.5	XML node types.	12
5.1	The pattern of the output file.	19

LIST OF TABLES

1

Introduction

Information retrieval is the group of techniques used for selectively recovering electronic information, guaranteeing access to large corpora of unstructured data. It is the technology behind Web search engines, being then used by many Web users everyday. Therefore, Information Retrieval Systems have to effectively and efficiently retrieve information from large stores of data.

In this context, the need for assessing performances of Information Retrieval Systems arises.

Several workshop series have then been designed to build the infrastructure necessary for the large-scale evaluation of information retrieval technology; some examples of them are TREC and CLEF Initiative.

The purpose of this thesis is to develop a software for optimizing the process of evaluation of an IRS.

The thesis is organized as follows.

Section 1 contains a deepening on the context, with a focus on how a evaluation campaign goes on and on how the software developed during the thesis would help during its progress.

Section 2 focuses on the background: it contains a deepening on technical aspects that helped developing the software.

Section 3 presents the architecture and the functionality of the software.

Section 4 contains a presentation of how to use the application, showing the functions offered to the user.

1. INTRODUCTION

Section 5 will then show how to source code is structured and will show how to expand its functionality.

Finally, Section 6 will present the summary of the work done and the opportunities of future development on it.

2

Evaluation of Digital Libraries Systems

2.1 Information Retrieval Technology

Information retrieval technology deals with a very familiar problem: finding relevant information in large stores of electronic documents [1]. It is an old problem, being faced since the first research conference devoted to this subject held in 1958 [2]. Since then the problem has continued to grow, as more people gain electronic access to increasingly information created in electronic form. The advent of the World Wide Web definitely increased the need for effective retrieval technology. In the last 20 years, large-scale evaluation campaigns, such as Text REtrieval Conference (TREC)¹ in the United States and the Conference and Labs of the Evaluation Forum (CLEF Initiative, formerly known as Cross-Language Evaluation Forum)² in Europe, have conducted cooperative evaluation efforts involving hundreds of research groups and industries, producing a huge amount of valuable data to be analyzed, mined and understood [3]. In particular, the CLEF Initiative has been launched in 2000 with the following objectives: “to develop and maintain an infrastructure for the testing and evaluation of information retrieval systems operating on European languages, in both monolingual and cross-language contexts, and to create test-suites of reusable data that can be employed by system developers for benchmarking purposes” [4].

¹<http://trec.nist.gov/>

²<http://www.clef-initiative.eu/>

2. EVALUATION OF DIGITAL LIBRARIES SYSTEMS

CLEF has recently organized its 13th edition, in Rome. Over its editions, CLEF has set its focus on different kinds of text retrieval across languages and different kinds of media, to encourage the development of next generation multilingual and multimedia IR systems.

Since 2005, Distributed Information Retrieval Evaluation Campaign Tool (DIRECT)¹ has been adopted in the CLEF campaigns: it is a Digital Library System (DLS) for managing the scientific data produced during an evaluation campaign and it has been developed with the following goals [5]:

- To be cross-platform and easily deployable to end-users;
- To be as modular as possible, clearly separating the application logic from the interface logic;
- To be intuitive and capable of providing support for the various user tasks, such as experiment submission, consultation of metrics and plots about experiment performances, relevance assessment, and so on;
- To support different types of users, i.e. participants, assessors, organizers, and visitors, who need to have access to different kinds of features and capabilities;
- To support internationalization and localization: the application needs to be able to adapt to the language of the user and his country or culturally dependent data, such as dates and currencies.

Giving DIRECT a new functionality is the purpose of this thesis.

Evaluation campaigns involve huge amounts of participants and are made of several stages.

2.2 Actors Involved in Evaluation Campaigns

Different types of actors are involved in an evaluation campaign [6]:

- The *participant* submits his experiments in a forum, that is used also for sharing and discussing new proposals of algorithms and techniques. His experiments need to be validated, then the participant can receive measurements about the

¹<http://direct.dei.unipd.it/>

performance of his experiments and overall indicators that allow his experiments and results to be compared with those submitted by other participants.

- The *assessor* helps in the creation of the experimental collections by proposing the topics and assessing the relevance of the documents with respect to those topics.
- The *visitor* consults all the information resources produced during the course of an evaluation campaign.
- The *organizer* manages several aspects of an evaluation forum: he prepares the documents that will be used as experimental collections and oversees the creation of topics and the relevance assessments; he provides the framework that will be used by participants and computes the different measures for assessing the performances of the submitted experiments; he also provides the visitors with the means for accessing all the information resources they are looking for.

These actors interact together in various ways during the course of an evaluation campaign.

2.3 Stages of an Evaluation Campaign

Evaluation campaigns proceeds through several stages [7]:

- *Acquisition and preparation of documents*: the organizers acquire and prepare the set of documents that will be released to participants.
- *Creation of topics*: the organizers and the assessors cooperate to create the topics for the test collection. Topics are the information need statements; indeed, they represent the user requests. Topics are created by inspecting the documents.
- *Experiment submission*: the participants submit their experiments, which are built using the documents and the topics created in the previous stages. The result of each experiment is a list of retrieved documents (in decreasing order of relevance) for each topic and represents the output of the execution of the information retrieval system (IRS) developed by the participant. Participants need an interface for uploading their experiments into the DLS and for describing them.

2. EVALUATION OF DIGITAL LIBRARIES SYSTEMS

- *Creation of pools*: the organizers, using some appropriate sampling technique, select a subset of retrieved documents by participants' experiments to be manually assessed to determine their actual relevance. Pools consist of this list of documents selected.
- *Relevance assessment*: the organizers and the assessors cooperate for determining whether or not each document in the pool is relevant for the given topic.
- *Measures and statistics*: the organizers exploit the relevance assessments to compute the performance measures and plots about each experiment submitted by a participant. These measurements are then employed for conducting statistical analyses and tests on the submitted experiments.
- *Scientific production*: organizers and participants prepare reports; the former provide an overview for the evaluation campaign, the latter explain their experiments and the techniques adopted. The output of this stage may correspond to theories, models, algorithms, techniques and observations, which are usually communicated by means of papers, talks and seminars.

DIRECT provides a central platform for participants in order to submit their experiments and for visitors to access the data produced during the evaluation campaign. This implies the advantage of having a single platform for accessing the history of ten years of CLEF data.

Since 2011 [5], the Participative Research labOratory for Multimedia and Multilingual Information Systems Evaluation (PROMISE)¹ evaluation infrastructure has been implemented in DIRECT. PROMISE is a Network of Excellence that aims at [15]:

- Managing and providing access to the scientific data produced during evaluation activities
- Supporting the organization and running of evaluation campaigns
- Increasing automation in the evaluation process
- Providing component-based evaluation
- Fostering the usage of the managed scientific data

¹<http://www.promise-noe.eu/>



Figure 2.1: The logo of Europeana.

An early prototype of PROMISE has firstly been used during the CLEF 2011 campaign.

2.4 The Purpose of the Thesis

The purpose of the thesis is to develop a software for automating the process that allows participants to conduct experiments and to build their results, so that these can be stored in DIRECT platform. Actually, indeed, participants have to manually conduct their experiments: that means they have to manually store the information about the documents retrieved querying topics through their Information Retrieval System. The software developed allows to automatically generate the file containing the results of an experiment, making continuous the process of evaluation. This file can then be submitted into DIRECT platform.

The component developed will be added in the list of functionality offered by PROMISE during an evaluation campaign.

2.5 Case Study: Europeana

Europeana¹ is a European Digital Library System, co-funded by the European Union, that provides users direct access to tens of millions of books, films, paintings, museum objects and archival records that have been digitised throughout Europe. It was inaugurated on 20 November 2008 by Viviane Reding, European Commissioner for Information Society and Media, in Brussels [16].

¹<http://www.europeana.eu/>

2. EVALUATION OF DIGITAL LIBRARIES SYSTEMS

The content on Europeana is cross-domain, but its metadata is mapped to a single data model - currently Europeana's Semantic Elements (ESE)¹ [17]. This content is made available to third parts through Europeana API² (Application Programming Interface) services.

The software developed for the thesis will use this API to automatically conduct experiments on Europeana Information Retrieval System.

¹<http://pro.europeana.eu/technical-requirements>

²<http://pro.europeana.eu/web/guest/api>

3

Background

3.1 Network Architecture, with a Focus on HTTP

It is useful to organize networks as a stack of layers, each one built upon the one below it. The purpose of each layer is to offer certain services to the higher layers while shielding those layers from the details of how these offered services are actually implemented [8]. When layer n on one machine carries on a conversation with layer n on another machine, the rules and conventions used in this conversation are collectively known as the layer n protocol. Basically, a protocol is an agreement between the communicating parties on how communication is to proceed. The most widely used architecture of network is the TCP/IP Reference Model (after its two primary protocols); it was first described by Cerf V. and Kahn R. [9] and later refined and defined as a standard in the Internet Community [10].

The link layer is the lowest layer in the model and it describes what links (such as Ethernet) must do to meet the needs of this connectionless internet layer. The internet layer permits host to inject packets into any network and have them travel

Application Layer
Transport Layer
Internet Layer
Link Layer

Table 3.1: The TCP/IP reference model.

3. BACKGROUND

Application Layer	HTTP, SMTP, RTP, DNS
Transport Layer	TCP, UDP
Internet Layer	IP, ICMP
Link Layer	DLS, SONET, 802.11, ETHERNET

Table 3.2: Examples of protocols for each layer.

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
PUT	Store a Web page
DELETE	Remove a Web page
CONNECT	Connect through a proxy

Table 3.3: Some HTTP methods.

independently to the destination. The transport layer is designed to allow peer entities on the source and destination hosts to carry on a conversation. The application layer contains all the higher-level protocols.

As you see in Table 3.2, HTTP (HyperText Transfer Protocol) is a protocol of the application layer. It is a simple request-response protocol that usually runs over TCP and is closely associated with the Web. The protocol specifies the types of messages clients can send to servers and what responses they get back in return; the contents of requests and responses are given in a MIME-like format. Since version 1.1 [11], HTTP has supported persistent connections: with them, it is possible to establish a TCP connection, send a request and get a response, and then send additional requests and get additional responses. This is very useful when you get a Web page that incorporates large numbers of embedded links, for content such as icons. HTTP supports methods other than just requesting a Web page, as shown in Table 3.3. The request line can be followed by additional lines, which form the request header (the same may happen with the response; in that case additional lines would form the response header).

Some examples of MIME (Multipurpose Internet Mail Extensions) types are *text/html*, *application/xml*, *application/json*, *image/png*.

In add to this, HTTP has built-in support to caching, that allows the user to reuse a page (if cached) without repeating the transfer; this can slightly improve the

3.2 eXtensible Markup Language (XML)

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Date	Both	Date and time the message was sent

Table 3.4: Some HTTP message headers.

performance by reducing both network traffic and latency.

3.2 eXtensible Markup Language (XML)

XML is a markup language for specifying structured content. Its use is strictly connected to the need for separating structured content from its presentation. Unlike HTML, there are no defined tags for XML: each user can define and define his own tags. The first proof of XML was produced on November 1996; on February 1998 XML 1.0 became a standard of W3C [19] and on February 2004 even XML 1.1 became a standard of W3C [12]. Since XML 1.1 is not fully compatible with XML 1.0, many users have continued using revised editions of XML 1.0 [18]. The latest edition of XML 1.0 it's the fifth one [20].

Rather than being a single language, XML should be considered a common notation which permits to build markup languages [18]. On its definition, there is nothing concerning the semantics of tags. XML is also platform-independent and all of its documents are written with Unicode.

Conceptually, XML documents form a tree structure that starts at "the root" and branches to "the leaves". The *root node*, the beginning node of every XML tree, represents the entire document. Every arch of the figure represents a child relationship. An element can have many or no children; the *leaves* correspond to those elements with no children.

Every node must belong to one of the categories showed on Table 3.5.

The Figure 3.1 shows an example of XML tree.

Element nodes are specified by opening and closing tags; the text between the tags is the content of the element and may consist to the children of the node. Opening and

3. BACKGROUND

Node Type	Description	Children
Document	Represents the entire document (the root-node of the tree)	Element (one at most), DocumentType, ProcessingInstruction, Comment
DocumentFragment	Represents a Document object, which can hold a portion of a document	CDATASection, Element, ProcessingInstruction, Comment, Text, EntityReference
DocumentType	Provides an interface to the entities defined for the document	None
ProcessingInstruction	Represents a processing instruction	None
EntityReference	Represents an entity reference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	Represents an element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Represents an attribute	Text, EntityReference
Text	Represents textual content in an element or attribute	None
CDATASection	Represents a CDATA section in a document (text that will NOT be parsed by a parser)	None
Comment	Represents a comment	None
Entity	Represents an entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	Represents a notation declared in the DTD	None

Table 3.5: XML node types.

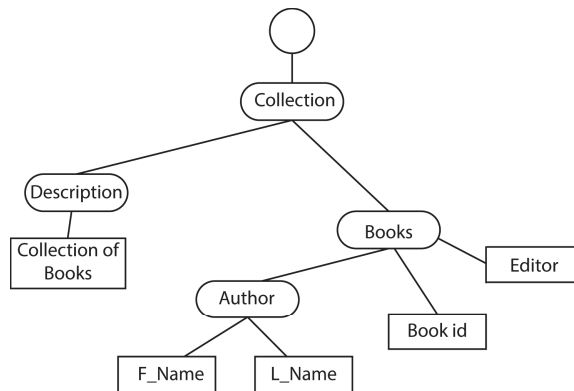


Figure 3.1: An example of XML tree.

closing tags must correspond.

An XML document usually starts with

```
<?xml version = "1.0" encoding="UTF-8"?>
```

Generally, XML languages are classifiable in four groups:

- Data-Oriented Languages: used for describing information, that usually is stored on databases. XML documents belonging to this group have generally a tree structure not deep.
- Document-Oriented Languages: text documents containing some tags for describing their structure. XHTML is a language belonging to this category.
- Protocols and programming languages: complex structures. XMLSchema, XSLT and WSDL belong to this group.
- Hybrid languages: a hybrid between the previous languages; in particular, it is common to find hybrid languages between Data-Oriented and Document-Oriented ones.

3.3 Java 5 Concurrency: Callable and Future

Concurrency is the ability to run several parts of a program in parallel. Till Java 1.4, the only way to obtain it was by implementing the Runnable interface or extending the class Thread. The problem is that they provide a simple method *run()* that cannot

3. BACKGROUND

return values, neither it can throw exceptions. Java 2 Platform Standard Edition (J2SE) 5.0 introduced the `Callable`¹ interface [13, 14], that allows user to return values from a thread. Its only method

```
V call() throws Exception
```

computes a results (as an instance of `V`), or throws an exception if unable to do so. You cannot pass a `Callable` into a `Thread` to execute: you have to use the `ExecutorService`² to execute the `Callable` object, through the `submit(Callable c)` method:

```
ExecutorService pool = Executors.newFixedThreadPool(NUMBER_OF_TASKS_VALUE);
Future<V> future = executor.submit(Callable<V> task);
```

In this example, a Fixed Thread Pool is created, but there several others implementations of `Executor service`:

- Single Thread Executor: a thread pool with only one thread, so that all submitted tasks will be executed sequentially
- Cached Thread Pool: a thread pool that creates as many threads it needs to execute the task in parallel
- Fixed Thread Pool: a thread pool with a fixed number of threads
- Scheduled Thread Pool: a thread pool made to schedule future tasks
- Single Thread Scheduled Pool: a thread pool with only one thread to schedule future tasks

As the last line of the example shows, submitting a `Callable` object to the `ExecutorService` returns a `Future` object. Finally use the `get()` method of `Future` to retrieved the value of the result, as it follows:

```
V tmp = future.get();
```

Using `get()` will block the computation if the result hasn't been computed yet. Finally, you can close the executor calling the `shutdown()` method: the executor will not shut down immediately; instead, it will no longer accept new tasks and once all the threads have finished current tasks, the executor will shut down.

¹`Callable` is available in the package `java.util.concurrent`

²`ExecutorService` is available in the package `java.util.concurrent`

4

Architecture and Functionalities

The software allows the user to import a list of topics from a file, to choose the path and the name of the file where the results of the experiment will be stored, and to select the system to query and the protocol to use, and finally to launch the experiment itself. The flow of the program is controlled by its main class, *Launcher*.

4.1 The Import of Topics

This sequence involves three classes: *Launcher*, *TopicParser* and *Topic*. The flow of this sequence starts with *Launcher* passing to *TopicParser* the file containing the topics. This file has already been selected by the user. Then, *TopicParser* starts parsing the file: for each topic found there, it initializes a new *Topic* object and saves it on a list of *Topic*. Once the parsing is finished, this list is returned to the *Launcher*, which will query these topics on the DLS.

4.2 Launching an Experiment

This process involves several stages: for each topic of the list, the launcher first obtains the string to be queried structured in a form consistent to the one used by the search engine; then, it makes an HTTP (or through other protocols, once developed an oportune management class¹) request to the DLS, that returns all the results of the query to the launcher. These results must be interpreted through a parser to be correctly stored. Finally the launcher writes the list of results on the output file. This

¹See paragraph 6.6.1

4. ARCHITECTURE AND FUNCTIONALITIES

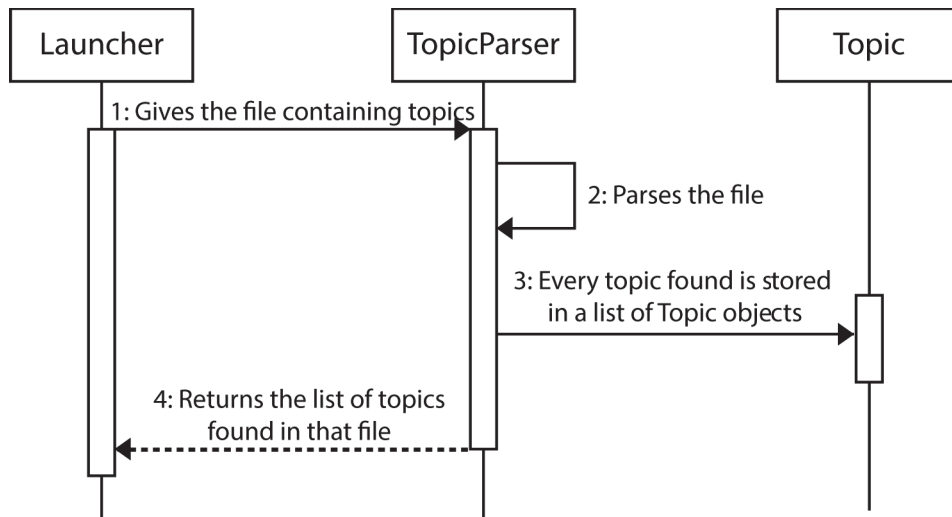


Figure 4.1: Sequence diagram for the import of topics.

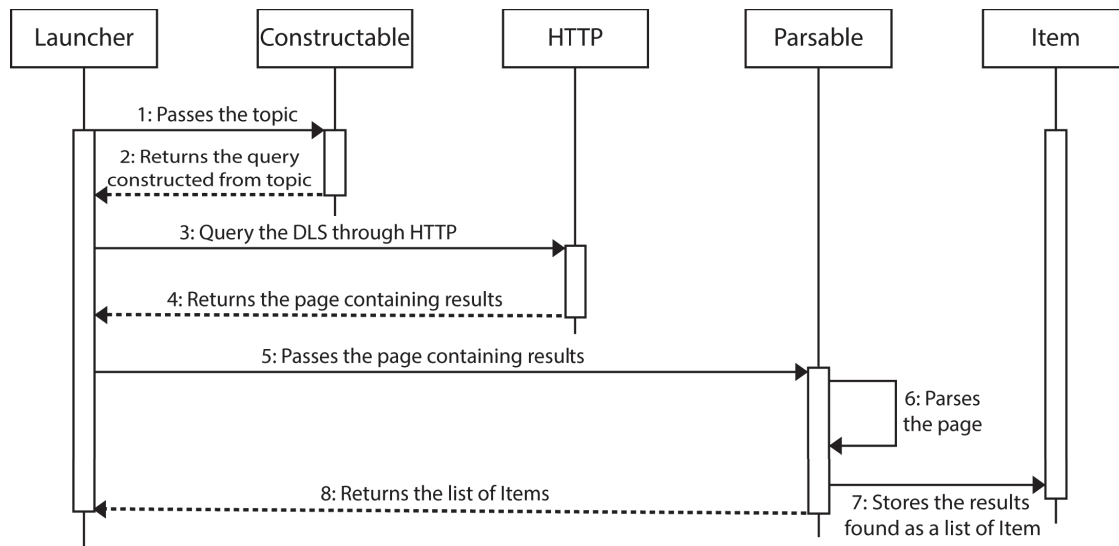


Figure 4.2: Sequence diagram for the launch of an experiment.

4.2 Launching an Experiment

process is followed for each topic. Parsable and Constructable are abstract classes, and they must be extended by specific classes for each DLS¹.

¹See paragraph 6.2

4. ARCHITECTURE AND FUNCTIONALITIES

5

Demonstration of the Functioning

To import topics, the user needs to import topics, by clicking on “File” and then “Import topics”.

Clicking on “Import Topics” will launch a File Chooser. Once chosen the file containing topics¹, the graphic user interface will be updated; user can then compile the fields regarding the output file that will be created.

The user then chooses the system to query and the protocol to use for obtaining responses from that system, so he can finally launch the experiment by pressing “Start Experiment!”.

After pressing the button for starting the experiment you might have to wait some seconds in order to let the experiment finishing its execution: indeed, all the topics of the file imported are being queried to the DLS, and the results of the query of each topic is being written on the output file.

¹See paragraph 5.5.4 to see what type of files is correctly loaded by the program

Topic ID	Q0	Link of the result	Rank of the result	Score of the result	Name of the experiment
----------	----	--------------------	--------------------	---------------------	------------------------

Table 5.1: The pattern of the output file.

5. DEMONSTRATION OF THE FUNCTIONING

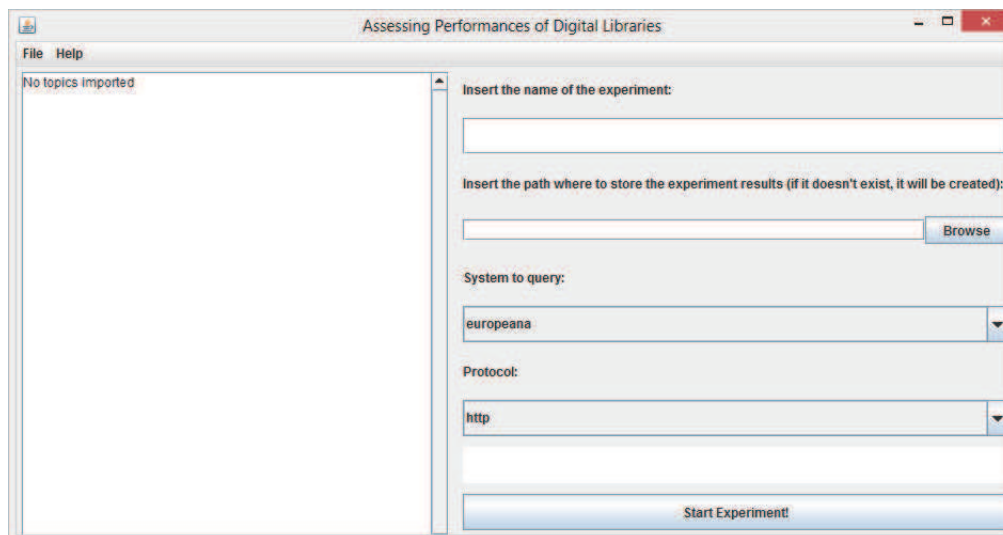


Figure 5.1: The main graphic interface of the program.

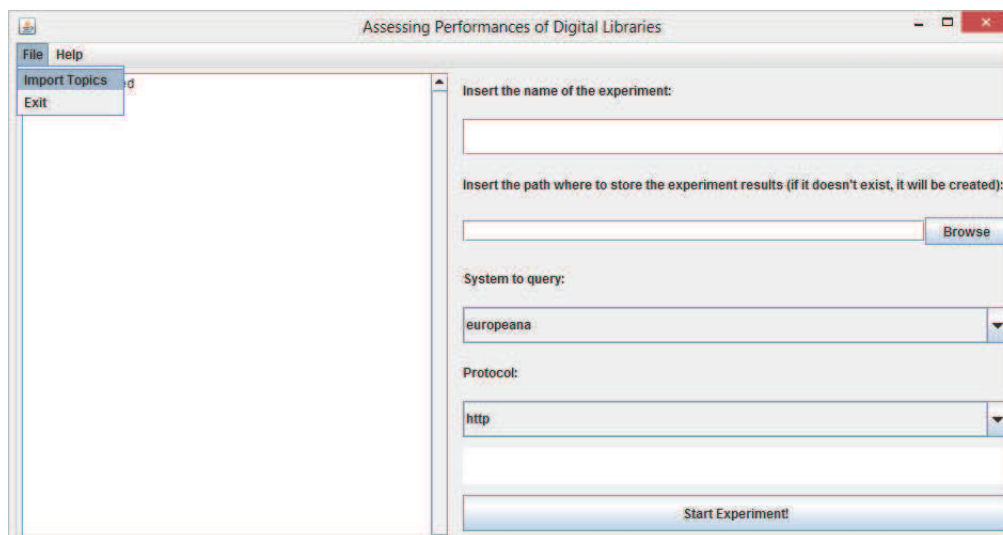


Figure 5.2: The import of topics.

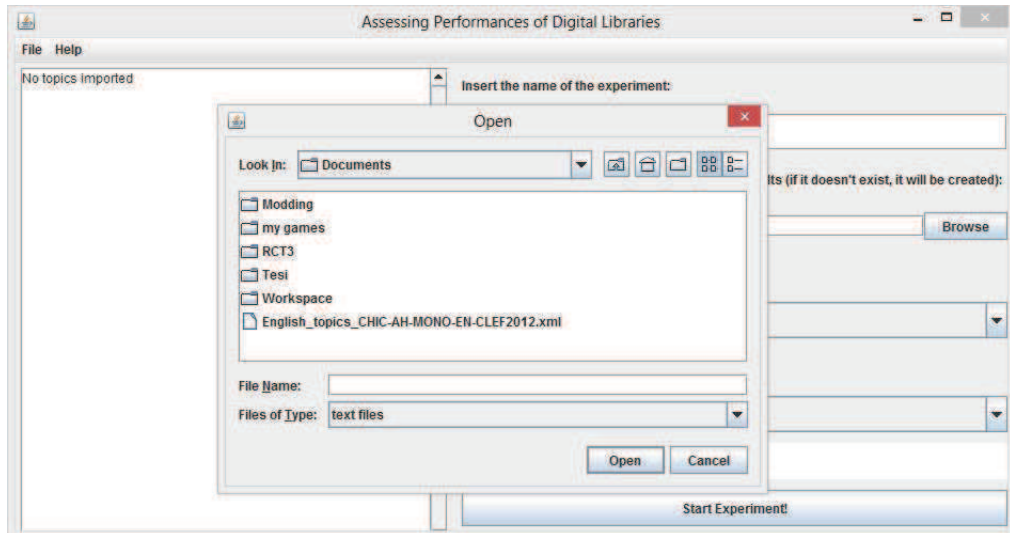


Figure 5.3: Select the file with topics

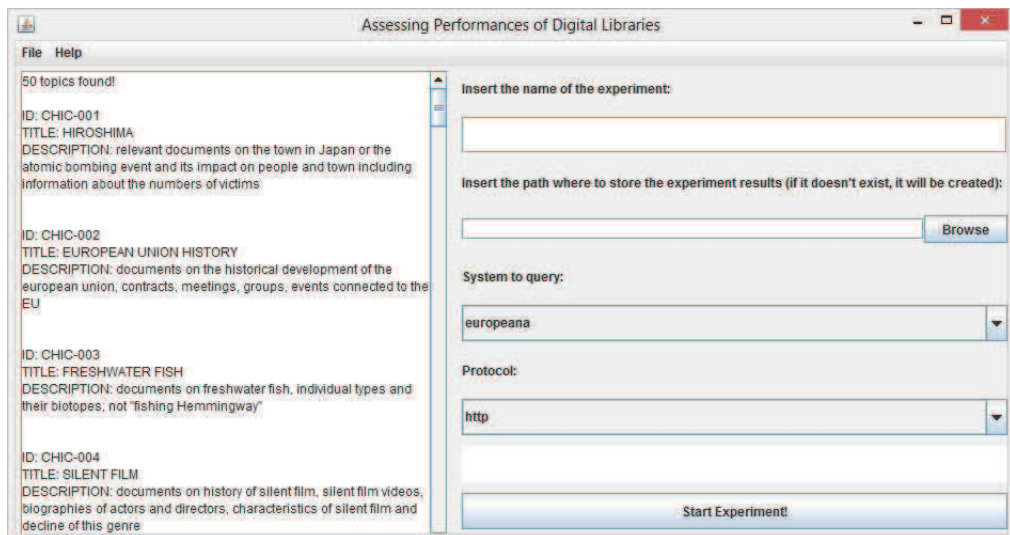
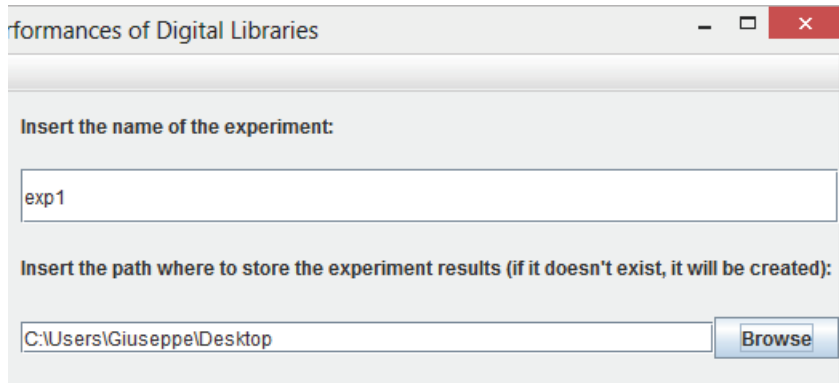


Figure 5.4: The GUI updated after the import of topics.

5. DEMONSTRATION OF THE FUNCTIONING



Assessing Performances of Digital Libraries

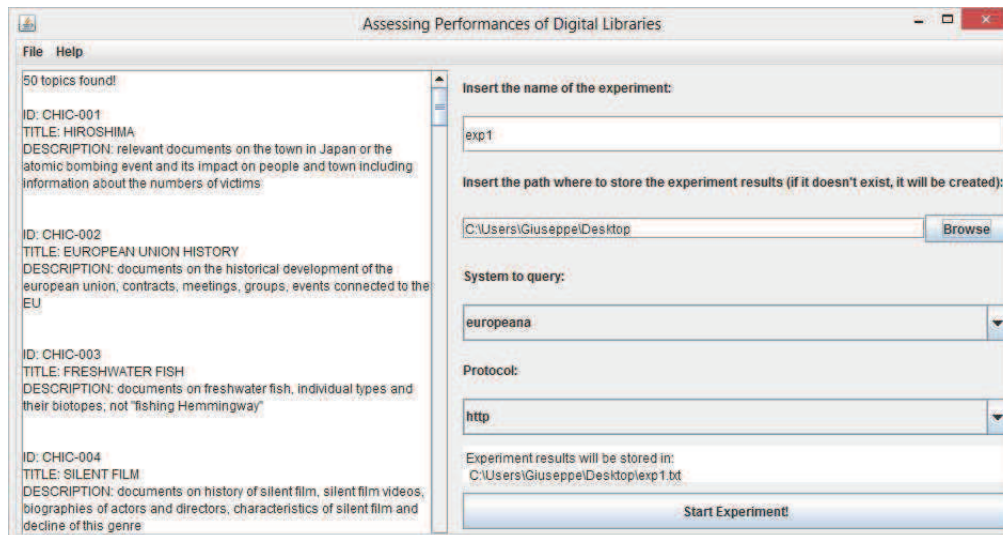
Insert the name of the experiment:

exp1

Insert the path where to store the experiment results (if it doesn't exist, it will be created):

C:\Users\Giuseppe\Desktop

Figure 5.5: Compiling fields.



Assessing Performances of Digital Libraries

File Help

50 topics found!

ID: CHIC-001
TITLE: HIROSHIMA
DESCRIPTION: relevant documents on the town in Japan or the atomic bombing event and its impact on people and town including information about the numbers of victims

ID: CHIC-002
TITLE: EUROPEAN UNION HISTORY
DESCRIPTION: documents on the historical development of the european union, contracts, meetings, groups, events connected to the EU

ID: CHIC-003
TITLE: FRESHWATER FISH
DESCRIPTION: documents on freshwater fish, individual types and their biotopes. not "fishing Hemmingway"

ID: CHIC-004
TITLE: SILENT FILM
DESCRIPTION: documents on history of silent film, silent film videos, biographies of actors and directors, characteristics of silent film and decline of this genre

Insert the name of the experiment:

exp1

Insert the path where to store the experiment results (if it doesn't exist, it will be created):

C:\Users\Giuseppe\Desktop

System to query:

europeana

Protocol:

http

Experiment results will be stored in:
C:\Users\Giuseppe\Desktop\exp1.txt

Figure 5.6: The experiment is ready to be launched.

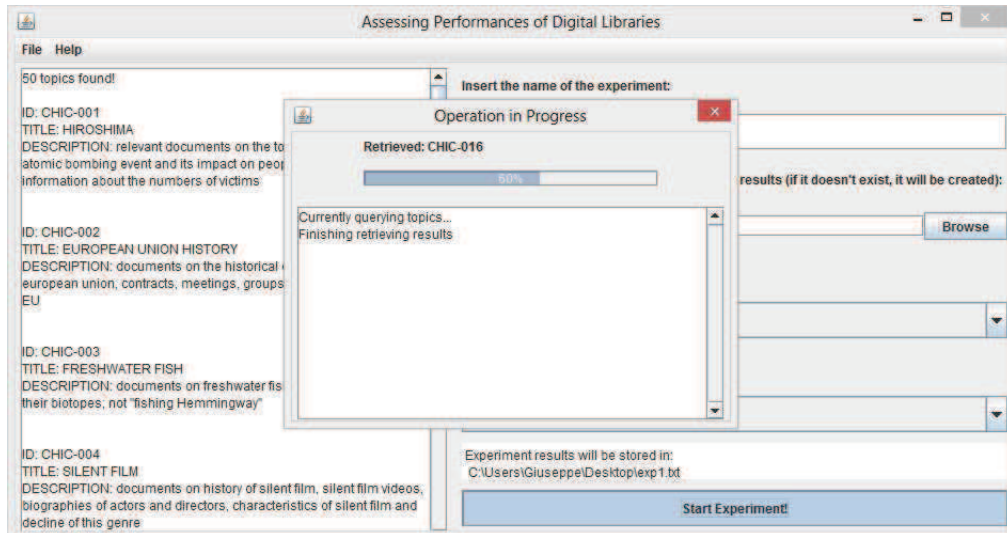


Figure 5.7: The experiment is being taken.

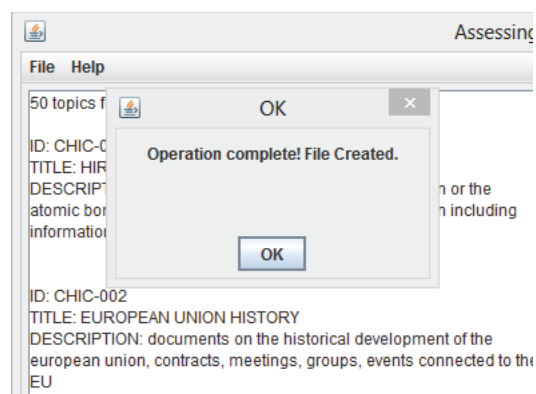
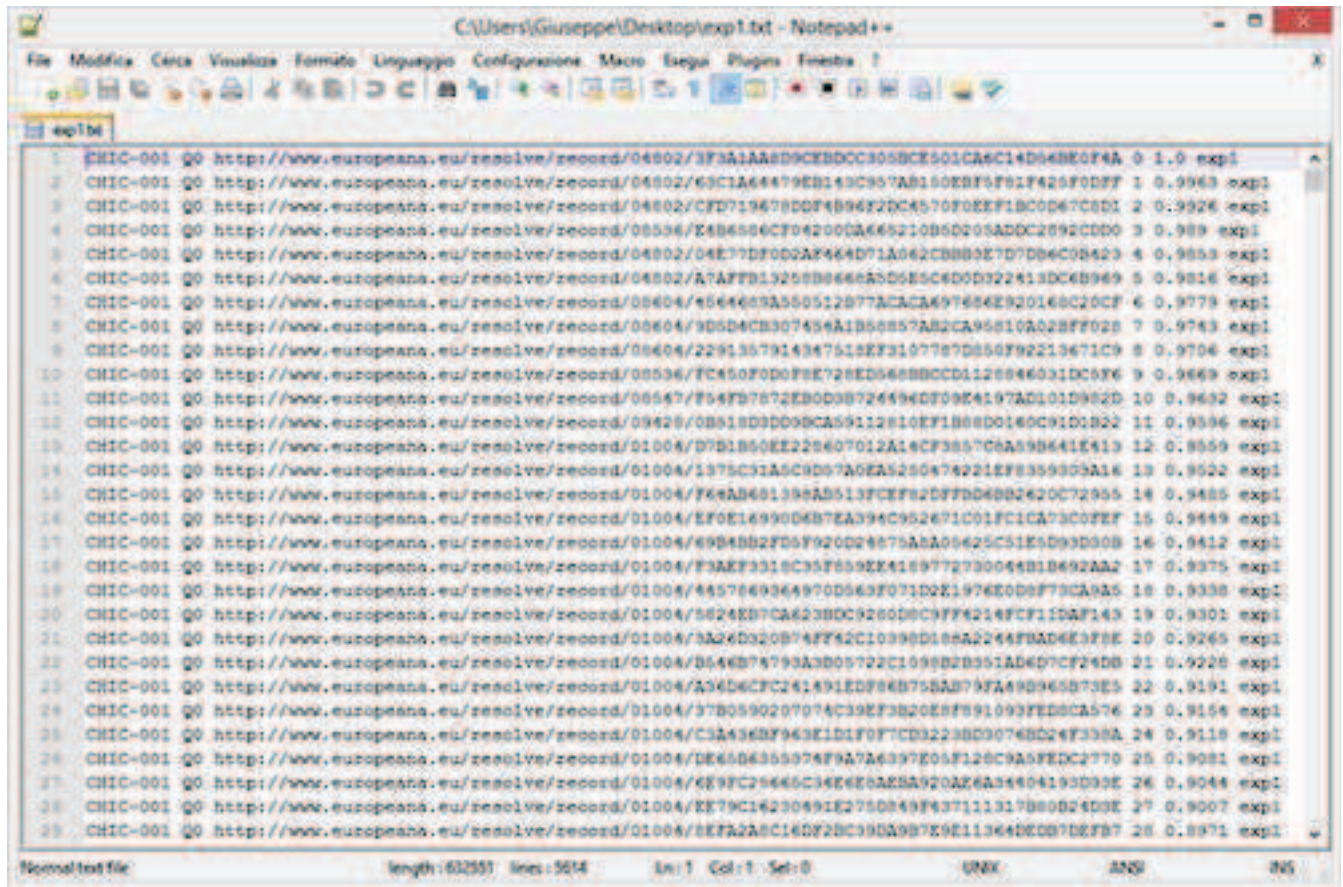


Figure 5.8: Experiment completed.

5. DEMONSTRATION OF THE FUNCTIONING



```
C:\Users\Giuseppe\Desktop\exp1.txt - Notepad++
File Modifica Cerca Visualizza Formato Linguaggio Configurazione Macro Esigui Plugin Finestra ?
exp1.txt
1 CHIC-001 Q0 http://www.europeana.eu/resolve/record/04802/3F3A1AA8D9CEBDC305BCE501CA6C14D948E0F4A 0 1.0 exp1
2 CHIC-001 Q0 http://www.europeana.eu/resolve/record/04802/63C1A64479EB149C957AB160EBF5F91F425F0D7F 1 0.9963 exp1
3 CHIC-001 Q0 http://www.europeana.eu/resolve/record/04802/CFD719678D0F4B9632DC4570F0EEF1BC0D67C8D1 2 0.9926 exp1
4 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08536/K4B6586CF04200DAK65210B6D005ADCC2692CDD0 3 0.989 exp1
5 CHIC-001 Q0 http://www.europeana.eu/resolve/record/04802/04E77DF0D0AF464D71A062CBB83E7D7D96C08423 4 0.9853 exp1
6 CHIC-001 Q0 http://www.europeana.eu/resolve/record/04802/A7AFFB13258B866EAB5D5E6C6D0D322413DC6B969 5 0.9816 exp1
7 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08404/4644603A550512B77ACACA697694E920168C20CF 6 0.9779 exp1
8 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08404/9D5D4C8307454A1B58857AB2CA96810A028FF028 7 0.9743 exp1
9 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08404/2291357914347515EF3107787D850F92213671C9 8 0.9704 exp1
10 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08536/FC450F0D0F8E728ED668BCCD1126846031DC0F6 9 0.9669 exp1
11 CHIC-001 Q0 http://www.europeana.eu/resolve/record/08547/F54FB7672EB0D3B724494DF09E4197AD101D982D 10 0.9632 exp1
12 CHIC-001 Q0 http://www.europeana.eu/resolve/record/09426/0B918D3D098CA59112810EF1888D0140C91D0822 11 0.9596 exp1
13 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/D7B1B50EE228407012A14CF3857C6A59B64E4E13 12 0.9559 exp1
14 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/1375C31A5C9D07A0EA5250474221EF8355903A16 13 0.9522 exp1
15 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/F64AB481394B513FCEFB2DFFD6882620C72955 14 0.9485 exp1
16 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/EF0E1499068B7EA394C952671C01FC1CA73C0FEF 15 0.9449 exp1
17 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/69B4BB2FD5F920D24879A5A05625C51E5D93D50B 16 0.9412 exp1
18 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/F3AEF3318C35F659XE416977273004481B692AA2 17 0.9375 exp1
19 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/4457849344970D863F071D2E1976E0D9F73CA9A5 18 0.9338 exp1
20 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/5624EB7CA623BDC9260D9C9FF4214FCF11DAF143 19 0.9301 exp1
21 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/3A26D020B74FF42C10398D185A2244FAD4643F8E 20 0.9265 exp1
22 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/B546B74793A3B05722C1098B2B351AD4D7CF24DB 21 0.9228 exp1
23 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/A34D6CF241491EDF86B7583B797A490965D73E5 22 0.9191 exp1
24 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/37B0590207074C39EF3B20E8F891093FED8CA576 23 0.9154 exp1
25 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/C32436DF943E1D1F0F7CD32238D00748D24F358A 24 0.9118 exp1
26 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/DE65B6355074F9A7A6397E04F126C9A5FEDC2770 25 0.9081 exp1
27 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/8E9FC29646C34E6E0A2BA920AK6A34404193D93E 26 0.9044 exp1
28 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/KE79C14230491E275D849F437111317B80824D0E 27 0.9007 exp1
29 CHIC-001 Q0 http://www.europeana.eu/resolve/record/01004/8EFA2A8C160F2BC99DA9B7K9E11364DE007DEFB7 28 0.8971 exp1
Normal text file length: 632551 lines: 5614 ln: 1 Col: 1 Sel: 0 UTF-8 ANSI 825
```

Figure 5.9: The output file.

6

The Source Code

6.1 An Overview

The code of the software is structured to give to future developers the possibility of easily expanding the functionalities of the program. The design of packages reflects this choice.

For now, the program can just use the Europeana search system and obtain responses from the DLS through HTTP.

The package *it.unipd.dei.ims.direct.ce.application* contains only a class, the *Launcher*, that manages the behavior of the software. The package *it.unipd.dei.ims.direct.ce.util* contains classes that are independent from digital libraries: this is where network protocol management classes and support ones are stored. The package *it.unipd.dei.ims.direct.ce* contains those abstract classes that will be extended by the most relevant classes of the digital libraries packages: indeed the main differences between the queries to different digital libraries involve the parsing of the results and the search form of the string that is about to be queried; these abstract classes have been developed to give the launcher an unrelieved structure, independent from the digital library chosen. Finally, the package *it.unipd.dei.ims.direct.ce.europeana* contains classes developed according to Europeana rules: here we can find an XML parser interpreting the page containing the results of a query, the *EuroQueryConstructor* class (that provides a method that returns a string in the search form of Europeana search engine), and some secondary classes which will be explained afterwards. There is also the package *it.unipd.dei.ims.direct.guielements*: indeed the launcher provides a basic GUI for simplifying the work for the user, and this

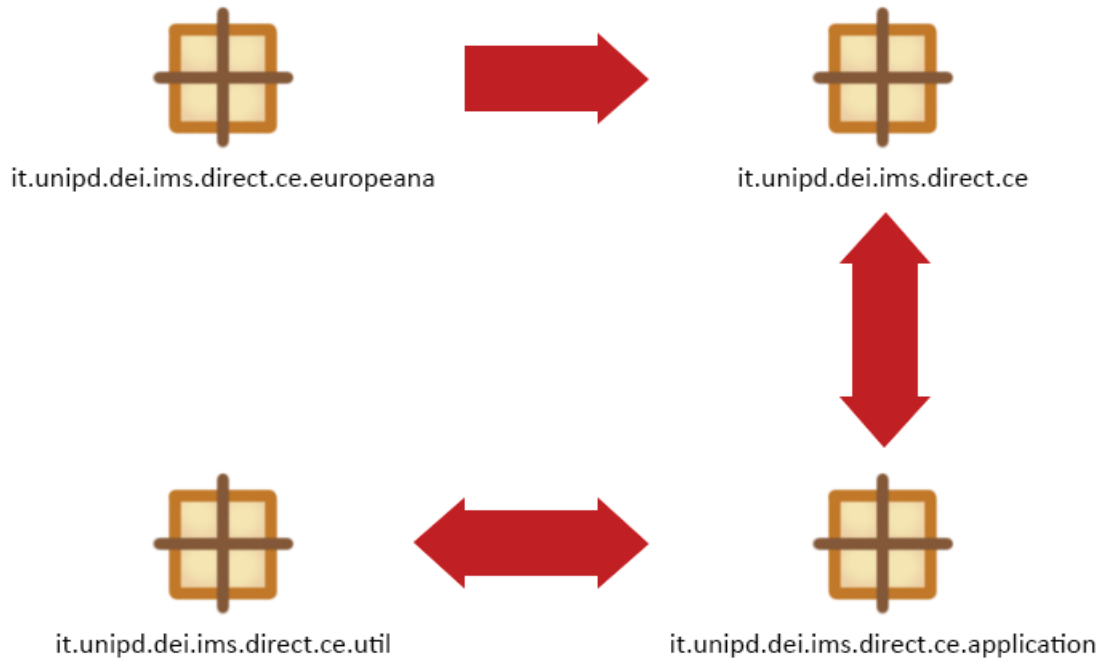


Figure 6.1: The organization of the packages.

package store some dialogs that can appear during the execution of the software.

6.2 Main Package

This package contains the following classes:

- *Constructable.java*
- *Parsable.java*

6.2.1 Constructable

This abstract class must be extended by all those classes intended to have a query constructor functionality. Indeed, the method *construct(Topic topic)* should be used to get the string that is about to be queried in a form consistent with the search form of the search engine of digital library. Use the API of the digital library system for overriding this method in the more corrected way for it.

```
/**
 * Abstract class that will be extended by all those class intended to have a
 *   query constructor functionality
 * @author Giuseppe Bandiera
 */
public abstract class Constructable {

/**
 * @param topic The topic from which you wish to obtain the query
 * @return The string that is about to be queried, according to the
 *   APIs of the search engine
 */
public abstract String construct(Topic topic);

}
```

6.2.2 Parsable

This class must be extended by all those classes that have a parsing functionality. The parameter of the method *parse(String content)* is, indeed, a string: then, the collecting of results pages must return (or must be converted to) a string, or a collection of it. For example, the *HTTP.getEntities(String query)* returns a list of strings, that will then be parsed singularly. Parsable provides two further methods: one returns the number of results, the other returns the list of results themselves. It is then implied that every class that extends Parsable should store these variables.

```
/**
 * Abstract class that will be extended by all those class intended to have a
 *   parsing functionality (i.e. XML parser, JSON parser...)
 * @author Giuseppe Bandiera
 */
public abstract class Parsable {

/**
 * Parses the document
 * @param content
 */
}
```

6. THE SOURCE CODE

```
public abstract void parse(String content);

/**
 * Returns the count of results found
 * @return The count of results found
 */
public abstract int returnCount();

/**
 * Returns the list of results
 * @return The list of results
 */
public abstract LinkedList<Item> returnList();
}
```

6.3 Europeana package

This package contains the following classes:

- *EuroQueryConstructor.java*
- *EuroXMLParser.java*
- *IDChanger.java*

6.3.1 EuroQueryConstructor

Extends *it.unipd.dei.ims.direct.ce.Constructable*. It allows to construct the string to be queried in a form consistent with the search form indicated on the Europeana OpenSearch API¹. The only method :

```
@Override
public String construct(Topic topic)
{
String result = "";
String title = topic.getTitle();
```

¹Available at <http://europeanalabs.eu/wiki/EuropeanaOpenSearchAPI>

```
StringTokenizer sTitle = new StringTokenizer(title, " ");
while (sTitle.hasMoreTokens())
{
    result += sTitle.nextToken() + "+AND+";
}

// remove the last '+AND+'
result = result.substring(0, result.length() - 5);

// remove ','
result = result.replace(",", "");

return result;
}
```

It was taken the decision to construct the query using only the title of the topic, in order to obtain more pertinent results. The tokens on the title are separated by '+AND+', as suggested on the API.

6.3.2 EuroXMLParser

Extends *it.unipd.dei.ims.direct.ce.Parsable*. It allows to parse an europeana results page, that follows this schema:

```
<item>
<guid></guid>
<other tags here, not relevant for software aims>
</item>
```

where item is a single result of the query, and guid is the tag indicating the link to that result. The partial source code:

```
/**
 * The parser of the xml pages, where query results are stored
 * @author Giuseppe Bandiera
 */
public class EuroXMLParser extends Parsable {
```

6. THE SOURCE CODE

```
int countItem = 0;
LinkedList<Item> itemList = new LinkedList<Item>();

@Override
public void parse(String content)
{
    InputStream inputStream = new
        ByteArrayInputStream(content.getBytes());
    XMLInputFactory inputFactory=XMLInputFactory.newInstance();
    XMLStreamReader streamReader = null;
    try {
        streamReader =
            inputFactory.createXMLStreamReader(inputStream);
    } catch (XMLStreamException e) {
        e.printStackTrace();
    }
    try {
        while(streamReader.hasNext() && countItem < 1000){

            streamReader.next();
            if(streamReader.getEventType() ==
                XMLStreamReader.START_ELEMENT){
                String elementName = streamReader.getLocalName();
                if("item".equals(elementName)){
                    // 'item' is the tag for each results
                    // collect link and rank of the item
                    Item item = parseItem(streamReader, countItem);
                    // then add the item to the list of results
                    itemList.add(item); countItem++;
                }
            }
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    }
```



```
//close the streams
try {
streamReader.close();
} catch (XMLStreamException e) {
e.printStackTrace();
}
try {
inputStream.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
/**
 * Method that stores the appropriate properties (link and rank) of the item
 * @param xml
 * @param rank
 * @return The item itself
 * @throws XMLStreamException
 */
public Item parseItem(XMLStreamReader xml, int rank) throwsXMLStreamException
{
Item item = new Item();
item.setRank(rank);
while(xml.hasNext()){
xml.next();
if(xml.getEventType() == XMLStreamReader.END_ELEMENT)
{
String elementName =
xml.getLocalName();
if("item".equals(elementName)){
break;
}
}
else if(xml.getEventType() == XMLStreamReader.START_ELEMENT){
String elementName = xml.getLocalName();
switch (elementName){
```

6. THE SOURCE CODE

```
        case "guid":
            {
                String link = xml.getElementText();
// change the link
                String id = IDChanger.toID(link);
item.setId(id);
                break;
            }
    }
}
}
```

These two methods basically work as the *parse()* and *parseTopic()* in *it.unipd.dei.ims.direct.ce.uti.TopicParser* . The main differences are the element tags together with the storing collection, that here is a *LinkedList* of *Items* (instead of a *Topics*' one). Note that the link stored for each item isn't the one found under "guid" tag, but this is changed through a static method of the class *IDChanger* (that will be described in the next paragraph). *EuroXMLParser* provides two further accessory methods:

```
@Override
public int returnCount()
{
return countItem;
}
```

```
@Override
public LinkedList<Item> returnList()
{
return itemList;
}
```

that allow to access to results and to their count.

6.3.3 IDChanger

Class that provides a static method for changing the structure of the link to the one used during CLEF evaluation campaign. For example, the link

```
http://www.europeana.eu/portal/record/  
08604/3998043176120F8071C26EDF2565D10E27B7BA33.html
```

will be changed to:

```
http://www.europeana.eu/resolve/record/  
08604/3998043176120F8071C26EDF2565D10E27B7BA33
```

The source code of the method is shown below.

```
/**  
 * Changes the structure of the link  
 * @param s  
 * @return The link structured in the form used during CLEF campaigns  
 */  
public static String toID(String s)  
{  
    {  
    String result = "";  
    StringTokenizer st = new StringTokenizer(s, "/.");  
    while (st.hasMoreTokens())  
    {  
    String tmp = st.nextToken();  
    switch (tmp)  
    {  
    case "portal":  
    result += "/resolve/";  
    break;  
  
    case "http":  
    result += tmp+ "//";  
    break;  
  
    case "html":
```

6. THE SOURCE CODE

```
break;

case "www":
result += tmp + ".";
break;

case "europeana":
result += tmp + ".";
break;

case "record":
result += tmp;
break;

case "eu":
result += tmp;
break;

default:
result += "/" + tmp;
break;

}
}
return result;
}
}
```

6.4 Application Package

This package contains only the class *Launcher.java*.

6.4.1 Launcher

This class is the core of the software: it mainly defers to:

- Import the topics to query
- Select the digital library to be queried, choosing an appropriate protocol
- Start a new experiment and save its results

It also provides a basic GUI for simplifying the work to the user.

Once the user clicks on Import Topics (as shown on chapter 5) the program launches a new *File Chooser* (through the method *selectTextFile()*); this will return a file that will then be parsed through the *TopicParser* :

```
class importAction implements ActionListener {
    public void actionPerformed(ActionEvent e) {
imported = selectTextFile();
TopicParser tp = null;
try
{
    tp = new TopicParser(imported);
    topicCount = tp.returnCount();
    if (topicCount == 0) {
        noTopicsAlert dialog = new noTopicsAlert();
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        dialog.setVisible(true);
    }
    topicList = tp.returnList();
    final Iterator<Topic> iterTopic = topicList.iterator();

    //filling the text area with the content of the topics
    String content = topicCount + " topics found!" + "\n\n";
    while (iterTopic.hasNext()) {
        Topic topic = iterTopic.next();
        content += topic.toString();
    }
    txtArea.setText(content);
}
catch (NullPointerException ex)
{
    //do nothing; the user will be warned of
```

6. THE SOURCE CODE

```
undesirable behaviour through the dialog
}

    }
}
```

The content of topics is added to the left side of the GUI through the line

```
txtArea.setText(content)
```

imported, *topicList* and *topicCount* are class variables:

```
public class Launcher {

    private static File imported;
    private static LinkedList<Topic> topicList;
    private static int topicCount;
[...]
```

The *selectTextFile()* method:

```
/**
 * Method invoked when "Import Topics" is clicked
 * @return The file selected
 */
public static File selectTextFile() {
    final JFileChooser chooser = new JFileChooser();
    //creating an opportune filter
    FileNameExtensionFilter filter =
new FileNameExtensionFilter("text files", "xml");
    chooser.setFileFilter(filter);

    int returnVal = chooser.showOpenDialog(null);

    if (returnVal == JFileChooser.APPROVE_OPTION) {
return chooser.getSelectedFile();
}
}
```

```
    else return null;
}
```

Note that, according to how this method works, it is required for topics to be stored in .xml files. Each topic retrieved by the parsing of the document is saved as an object of the class *Topic.java* and added to the *LinkedList* of *Topics topicList*. The parsing itself is made through the methods of the class *TopicParser.java*. More on these classes will be explained afterwards. In order to start the experiment, the user must select the name of it and the path of its save; the name of the experiment will be reported under each line of the output file. The relevant source code concerning the act of saving the results on the output file is shown below.

```
/**
 * Method that writes the results of each topic on the output file
 * @param toWrite The string to be written
 */
public static void writeToFile(String toWrite) {
    PrintWriter prf = null;
    try {
        prf = new PrintWriter(new FileWriter(experimentFile,true));
        prf.println(toWrite);
        prf.flush();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    finally {
        if (prf != null){
prf.close();
        }
    }
}
```

6. THE SOURCE CODE

The class Launcher stores also the informations about the Digital Library and the protocol chosen¹. Anyway, the core of the class is the method *launchExp()* : this method allows the user to create fifty *Futures* (fifty is the estimated number of topics stored in the file imported), each:

- establishing a connection through the required protocol
- collecting the responses obtained through that connection
- parsing these responses in order to store the results of the query

Finally the results are written on the output file. The code of this method is shown below.

```
/**
 * Method that queries the topics to the digital library system
 */
public static void launchExp(){
    final Iterator<Topic> iterTopic = topicList.iterator();
    ExecutorService executor = new ScheduledThreadPoolExecutor(THREADS);
    List<Future<String>> list = new ArrayList<Future<String>>();

    operationInProgressDialog dialog = new operationInProgressDialog();
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    dialog.setVisible(true);

    while (iterTopic.hasNext()) {
        final Topic topic = iterTopic.next();
        Future<String> future = executor.submit(new Callable<String>(){

@Override
public String call() throws Exception {
    Parsable parser = null;
    Constructable constructor = null;
    if (type.equals("europeana")){
parser = new EuroXMLParser();
constructor = new EuroQueryConstructor();
```

¹For more details about how to add more functionalities see paragraph 6.6


```

}
// otherwise match other types (once developed) here

    String result = ""; //string that will be printed on the output file
    String constructed = constructor.construct(topic);
    LinkedList<String> contentList = null;
    if (protocol.equals("http"))
contentList = HTTP.getEntities(constructed);
    // otherwise match other protocols (once developed) here

    Iterator<String> iterContent = contentList.iterator();
    while (iterContent.hasNext()) {
    // removing non utf-8 characters
    String content = iterContent.next().
replaceAll("[^\\x20-\\x7e]", "");
    parser.parse(content);
}

//difference that will be subtracted to each score, so that range
score will be 0.001 - 1
double difference = DoubleHelper.getDifference(parser.returnCount());
LinkedList<Item> itemList = parser.returnList();
Iterator<Item> it = itemList.iterator();
double score = 1;
while (it.hasNext()) {
    Item item = it.next();

// setting score manually, because europeana doesn't return the score of a result.
if (type.equals("europeana")){
    item.setScore(DoubleHelper.roundUp(score, 4));
    score -= difference;
    result += topic.getID() + " Q0 " + item + " " + experiment + "\n";
}

return result;
}

```

6. THE SOURCE CODE

```
    });

    list.add(future);
}
executor.shutdown();
for (Future<String> fut : list)
try {
writeToFile(fut.get());
}
catch (InterruptedException | ExecutionException e) {
e.printStackTrace();
}

dialog.setVisible(false);
operationCompleteDialog dialog2 = new operationCompleteDialog();
dialog2.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
dialog2.setVisible(true);
dialog2.setAlwaysOnTop(true);
}
```

A new *Future* is associated to each topic of the *LinkedList*. The abstract classes *Parsable* and *Constructable* allow to create abstract objects that can then be initialized to the appropriate objects, according to the value of string type. The *HTTP* class provides the static method *getEntities(String s)*: this returns a *LinkedList* of *Strings*, each storing a page of results. For each string that belongs to this *LinkedList*, the method *launchExp()* first removes the non-UTF8 characters, and then parse the content of the *String* through *EuroXMLParser.parse(String s)* method. This method returns a linked list of *Item*, one per result: each *Item* is then added to the *String* result together with other fixed elements, so that every row of result will follow the schema shown on Table 5.1. Once iterated the *LinkedList* of the *Topics*, the *Futures* are executed; each of them returns the results (as a *String*) of the query of a single topic. These results are then written in the output file, through the method *writeToFile(String s)* described before. The *launchExp()* is actually called once the button “Start experiment” is clicked; before

calling it the program creates the output file, here's the concerning code:

```
class startExpAction implements ActionListener {
    public void actionPerformed(ActionEvent e) {
if (expField.getText().equals("") || pathField.getText().equals(""))
    //can't start the experiment if there's no output file...
{
    compileFieldsAlert dialog = new compileFieldsAlert();
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    dialog.setVisible(true);
}
else if (topicCount == 0) //...or no topics imported
{
    noTopicsAlert dialog = new noTopicsAlert();
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    dialog.setVisible(true);
}
else {
    experiment = expField.getText();
    path = pathField.getText();

    // create new dirs if necessary
    File tmp = new File(path);
    if (tmp.exists() == false)
tmp.mkdirs();

    // create the file of the experiment
    experimentFile = new File(path + "\\\" + experiment + ".txt");
    if (experimentFile.exists() == false)
    try {
experimentFile.createNewFile();
    }
    catch (IOException e1) {
e1.printStackTrace();
    }
}
```

6. THE SOURCE CODE

```
        //finally launch the experiment
        launchExp();
    }
}
}
```

6.5 Util Package

This package contains the following classes:

- *HTTP.java*
- *Topic.java*
- *TopicParser.java*
- *Item.java*
- *DoubleHelper.java*

6.5.1 HTTP

This class manages a single HTTP connection; indeed, it collects (under a linked list of *Strings*) the several responses to a query, each per page. To handle the HTTP connection, the class uses the Apache HTTP Client library¹. The class HTTP contains only a static method:

```
/**
 * A method for getting the content of http entities
 * @param query The string representing the query
 * @return A linked list of http entities' content, each stored in a string
 * @throws IOException
 */
public static LinkedList<String> getEntities(String query) throws IOException {
    String type = Launcher.type;
    LinkedList<String> contentList = new LinkedList<String>();
```

¹Available at <http://hc.apache.org/httpcomponents-client-ga/index.html>

```
HttpClient httpClient = new DefaultHttpClient();
boolean morePages = true;
int page = 1;
while (morePages)
{
String get = null;
if (type == "europeana")
get = "http://api.europeana.eu/api
/opensearch.rss?searchTerms="+query+
"&wskey=APIkey&startPage="+page;

// other types here; if the results are hosted
on a only page, scan it and then set morePages to false

HttpGet httpget = new HttpGet(get);
HttpResponse response = httpClient.execute(httpget);
HttpEntity entity = response.getEntity();

// since I must get 1000 results at the most,
it's useless to require more than 84
pages (there are 12 results per page)
if(entity.getContentLength() == -1 &&
page < 85 && type == "europeana")
{
// the europeana page contains results
contentList.add(EntityUtils.toString(entity));

// close the reading of the entity
EntityUtils.consume(entity);
page++;
}
else morePages = false;
}
return contentList;
}
```

6. THE SOURCE CODE

This class reads the value of *String Launcher.type*, and consequently queries the opportune digital library database. The HTTP request is put inside a *while* cycle, so that it is possible to handle the response of those digital libraries that store the results on more than a page. In regards to the second *IF* statement: it has empirically been noticed that if an Europeana page contains results, the *http.HttpEntity.getContentLength()* method applied to its entity returns -1, otherwise it returns a random positive integer value; in add to this, the method will scan only the first 84 pages of results, because Europeana stores 12 results per page and the purpose of the software is to collect 1000 results per query at maximum; then, scanning more than 84 pages would be useless. If a page contains results, its content is converted to a *String* through the method *http.util.EntityUtils.toString(HttpEntity entity)* and then this *String* is added to the *LinkedList* of *Strings* that will finally be returned to the calling method.

6.5.2 Item

This class represents a single result of a query.

```
/**
 * Class representing a single result of the query
 * @author Giuseppe Bandiera
 */
public class Item {

    private int rank;
    private String ID;
    private double score;

    /**
     * Sets the rank of the result
     * @param i The desired rank
     */
    public void setRank(int i)
    {
        rank = i;
    }
}
```

```
/**
 * Sets the ID of the result
 * @param s The desired ID
 */
public void setId(String s)
{
ID = s;
}
/**
 * Sets the score of the result
 * @param d The desired score
 */
public void setScore(double d)
{
score = d;
}

/**
 * Prints the result
 */
public String toString()
{
return ID + " " + rank + " " + score;
}
}
```

The method *toString()* is defined so that every item will be printed in the desired way on the output file.

6.5.3 Topic

This class represents a single topic. It has the fields *ID*, *title* and *description*, and methods for setting and getting those fields. The method *toString()* was optional and done just for having a correct visualization in the GUI of the topics stored in the file imported.

6. THE SOURCE CODE

6.5.4 TopicParser

This class reads an XML file and parses it, collecting topics found on the file. The file must follow this pattern to be correctly parsed:

```
<topic>
  <identifier></identifier>
  <title></title>
  <description></description>
</topic>
```

That file is stored as a variable of the class:

```
/**
 * Reads an XML file, collecting topics stored onto it
 * @author Giuseppe Bandiera
 */
public class TopicParser {

    File file;
    LinkedList<Topic> topicList;
    int countTopic = 0;

    public TopicParser(File f)
    {
        file = f;

        try
        {
            parse();
        }
        catch (ParserConfigurationException | SAXException | IOException | XMLStreamException e)
        {
            //do nothing
        }
    }
    [...]
}
```


If an exception is caught, code will impose to do nothing, since the GUI will avoid undesirable behavior of the program; indeed, only xml files can be parsed (thanks to the filter applied to the file chooser, in the *Launcher.selectTextFile()* method). The source code of *parse()* :

```
/**
 * Parses the XML document; collects the topics found into the
 *   LinkedList<Topic> topicList
 * @throws ParserConfigurationException
 * @throws SAXException
 * @throws IOException
 * @throws XMLStreamException
 */
private void parse() throws ParserConfigurationException, SAXException,
{
topicList = new LinkedList<Topic>();
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader streamReader = factory.
    createXMLStreamReader(new FileReader(file));
while(streamReader.hasNext()){
streamReader.next();
if(streamReader.getEventType() == XMLStreamReader.START_ELEMENT){
String elementName = streamReader.getLocalName();
if("topic".equals(elementName)){
Topic topic = parseTopic(streamReader);
topicList.add(topic);
countTopic++;
}
}
}
}
```

The library *javax.xml.** is used here to create the XML Stream Reader. The source code of *parseTopic(XMLStreamReader streamReader)* :

```
/**
```

6. THE SOURCE CODE

```
* Saves information for each topic found
* @param xml
* @return The topic itself
* @throws XMLStreamException
*/
public Topic parseTopic(XMLStreamReader xml) throws XMLStreamException
{
    Topic topic = new Topic();
    while(xml.hasNext()){
        xml.next();
        if(xml.getEventType() == XMLStreamReader.END_ELEMENT)
        {
            String elementName = xml.getLocalName();
            if("topic".equals(elementName)){
                break;
            }
        }
        else if(xml.getEventType() == XMLStreamReader.START_ELEMENT){
            String elementName = xml.getLocalName();
            switch (elementName)
            {
                {
                    case "identifier":
                        topic.setId(xml.getElementText());
                        break;
                    case "title":
                        {
                            topic.setTitle(xml.getElementText());
                            break;
                        }
                    case "description":
                        topic.setDescription(xml.getElementText());
                        break;
                }
            }
        }
    }
    return topic;
}
```

```
}
```

The mechanism of these two methods is basically the following: while getting the event type of every element of the file, *parse()* method compares this type with the final value *XMLStreamReader.START_ELEMENT* : if the result of the comparison is affirmative and the element name is “topic”, then the stream reader is meeting a new topic element and has to save its attributes through the *parseTopic(XMLStreamReader xml)* method. Once finished saving these attributes, this method returns an object of the class *Topic*, that will be added by the *parse()* method to the *LinkedList<Topic> topicList* . This list can be returned through the method *returnList()* :

```
/**
 * Returns the list in which all topics found are stored
 * @return the list of topics found
 */
public LinkedList<Topic> returnList()
{
return topicList;
}
```

The class provides a further accessory method:

```
/**
 * Returns the count of topics found
 * @return The count of topics found
 */
public int returnCount()
{
return countTopic;
}
```

that can be used for controls, in order to have a safer execution of the program.

6.5.5 DoubleHelper

Not all digital libraries report the score of a result of a query: in this case the score must be added manually. The class *DoubleHelper* is intended to provide help in giving

6. THE SOURCE CODE

an appropriate score to a result. The class provides two static methods:

```
/**
 * Gets the difference that must elapse between two consecutive results
 * @param count The number of results
 * @return The difference between the score of two consecutive items
 */
public static double getDifference(int count)
{
return ((double)1/count);
}

/**
 * Rounds a decimal value
 * @param double d The value to be rounded
 * @param int p The desired quantity of decimal values
 * @return The double value d rounded up to p decimal values
 */
public static double roundUp(double d, int p)
{
return Math rint(d * Math.pow(10,p)) / Math.pow(10,p);
}
```

The *getDifference(int count)* method should be used in order to get the difference of score between two consecutive results. Using this method will allow to spread evenly the scores in the range from 0.001 to 1. The parameter count should corresponds to the number of results. The *roundUp(double d, int p)* method returns the double d value rounded up to p decimal values. Scores are usually rounded up to 4 decimal values.

6.6 Extension to Other DLS

It may be necessary to load the file containing topics from remote locations, rather than from a local path. In this case, problem would be solved modifying *Launcher.selectTextFile()* or calling a new more adapted method instead of *selectTextFile()* . It also could be necessary to save experiments' results in remote paths rather than in local ones: in this case lines 271-284 of *Launcher.java* should be advisably revised:

```
// create new dirs if necessary
File tmp = new File(path);
if (tmp.exists() == false)
tmp.mkdirs();

// create the file of the experiment
experimentFile = new File(path + "\\\" + experiment + ".txt");
if (experimentFile.exists() == false)
try {
experimentFile.createNewFile();
}
catch (IOException e1) {
e1.printStackTrace();
}
```

6.6.1 Adding Support to Other Application-Layer Network Protocols

New network protocols¹ management classes should be added in the *it.unipd.dei.ims.direct.util* package. It is important that the method for retrieving the response returns a *LinkedList* of *Strings* so that the implementation of new protocols would be possible just adding few lines in this part of *Launcher.java* :

```
LinkedList<String> contentList = null;
if (protocol.equals("http"))
contentList = HTTP.getEntities(constructed);
// otherwise match other protocols (once developed) here
```

New protocols must also be added in *Launcher.protocols* (that is an array of *Strings*): indeed the protocol value is picked by the user as a string of that array.

6.6.2 Expanding HTTP management class

HTTP.java can be expanded to manage HTTP responses from several Digital Libraries Systems. In particular the following ones are the fragments to modify.

¹See paragraph 3.1 for more details on network protocols

6. THE SOURCE CODE

```
boolean morePages = true;
int page = 1;
while (morePages)
{
String get = null;
if (type == "europeana")
get="http://api.europeana.eu/api/openserch.rss?
searchTerms="+query+"&wskey=APIkey&startPage="+page;

// other types here; i.e.: else if (type == \DLname") get = \..."
```

If results are stored in just a page, ignore the *page* variable, scan that page and then set *morePages* to *false* .

Here's how to scan a page :

```
HttpGet httpget = new HttpGet(get);
HttpResponse response = httpClient.execute(httpget);
HttpEntity entity = response.getEntity();

if( // type == \europeana" and the page contains results)
{
[...]
}
else if( // type == \yourDL" and the page contains results)
{
contentList.add(EntityUtils.toString(entity));

// close the reading of the entity
EntityUtils.consume(entity);
page++;
}
else morePages = false;
```

6.6.3 Adding support to other Digital Library Systems

The support to other digital libraries should be added by adding or modifying the class for the management of the communication through network-layer protocols (as indicated in the previous paragraphs) and also by creating at least two classes in new provided packages. The two created classes should extend *it.unipd.dei.ims.direct.ce.Parsable* and *it.unipd.dei.ims.direct.ce.Constructable* respectively. The class that extends *Parsable* must store the results found in a *LinkedList* of *Items*; the class that extends *Constructable* must use the Digital Library Search Engine API to get the string to be queried in the correct form. Once written these classes, *Launcher.launchExp()* must be modified in these lines (line 427 and on):

```
Parsable parser = null;
Constructable constructor = null;
if (type.equals("europeana"))
{
parser = new EuroXMLParser();
constructor = new EuroQueryConstructor();
}
// otherwise match other types (once developed) here
```

If the new Digital Library System doesn't report the score of a result, you can use the functionalities of the class *DoubleHelper*, simply modifying the line 463 of *Launcher.java*:

```
if (type.equals("europeana") || type.equals("yourNewDLS")){

item.setScore(DoubleHelper.roundUp(score, 4));
score -= difference;
}
```

New digital libraries must also be added in *Launcher.types* (that is an array of *String*): indeed the *type* value is picked by the user as a string of that array.

6. THE SOURCE CODE

7

Conclusions

The software introduced allows to automate the execution of experiments during an evaluation campaign. Indeed, it automatically queries the topics to Europeana search system. Using these software, the strain required to participants of an evaluation campaign in order to conduct the experiments will be significantly smaller.

As far as future work is concerned, the software will be implemented as a component of PROMISE, corresponding with its plans to increase automation in the evaluation process. Participants will use this component to automatically generate the experiment results and will then upload them into DIRECT portal.

The software can be extended for supporting other Digital Library Systems (using the guide at Chapter 6.6); the possibility of automatic evaluation will then be incremented and many Digital Libraries will benefit from it.

Bibliography

- [1] Harman, D. K., & Voorhees, E. M. (2005). *TREC. Experiment and Evaluation in Information Retrieval*. Cambridge: MIT Press.
- [2] Sparck Jones, J., & Willet, P. (1997). *Reading in information retrieval*. San Francisco: Morgan Kaufmann Publishers.
- [3] Di Buccio, E., Dussin, M., Ferro, N., Masiero, I., Santucci, G., & Tino, G. (2011). *Interactive Analysis and Exploration of Experimental Evaluation Results*. (J. Kalbach, B. Larsen, T. Russell-Rose, & M. Wilson, Eds.) Retrieved August 23, 2012, from EuroHCIR 2011: <http://ceur-ws.org/Vol-763/>
- [4] Peters, C. (2001). Introduction. In C. Peters, *Cross-Language Information Retrieval and Evaluation: Workshop of Cross-Language Evaluation Forum (CLEF 2000)* (pp. 1-6). Heidelberg: Springer.
- [5] DIRECT. (2012). *About DIRECT*. Retrieved August 30, 2012, from <http://direct.dei.unipd.it/>
- [6] Dussin, M., & Ferro, N. (2007). *Design of the user interface of a scientific digital library system for large-scale evaluation campaigns*. Retrieved August 29, 2012, from Second DELOS Conference on Digital Libraries: http://www.delos.info/index.php?option=com_content&tast=view&id=602&itemid=334

- [7]Ferro, N., & Agosti, M. (2010). Towards an infrastructure for digital library performance evaluation. In G. Tsakonias, & C. Paptheodorou, *Evaluation of Digital Libraries: An Insight to Useful Applications and Methods* (pp. 93-120). Oxford: Chandos Publishing.
- [8]Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks, 5th Ed.* Boston: Prentice Hall.
- [9]Cerf, V., & Kahn, R. (1974). A Protocol for Packet Network Interconnection. In *IEEE Trans. on Commun.* (pp. 637-648).
- [10]Braden, R. (1989). *Requirements for Internet Hosts - Communication Layers*. Retrieved August 29, 2012, from RFC 1122: <http://tools.ietf.org/html/rfc1122>
- [11]Fielding, R., Irvine, U., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., et al. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*. Retrieved September 4, 2012, from W3.org: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [12]Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (2004). *Extensible Markup Language (XML) 1.1*. Retrieved August 29, 2012, from W3.org: <http://www.w3.org/TR/2004/REC-xml11-20040204/>
- [13]Oracle. (2004, September 30). *Concurrency Utilities*. Retrieved August 29, 2012, from J2SE(TM) 5.0 - New Features and Enhancements: <http://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html>
- [14]Oracle. (2004). *Callable Interface*. Retrieved August 29, 2012, from Java™ 2 Platform Standard Ed. 5.0: <http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Callable.html>
- [15] Agosti, M., Braschler, M., Di Buccio, E., Dussin, M., Ferro, N., Granato, G.L., Masiero, I., Pianta, E., Santucci, G., Silvello, G., & Tino, G. (2011). *Promise. Specification of the evaluation infrastructure based on user requirements. Deliverable number: 3.2*. Retrieved September 15, 2012, from promise-noe.eu: <http://www.promise-noe.eu/documents/10156/fdf43394-0997-4638-9f99-38b2e9c63802>
- [16]EuropeanaConnect. (2010). *Europeana Connect Related projects*. Retrieved September 15, 2012, from EuropeanaConnect - Related Projects: <http://www.europeanaconnect.eu/related-projects.php>

- [17] Europeana. (2012). *Facts and Figures*. Retrieved September 15, 2012, from Europeana Professional: <http://pro.europeana.eu/web/guest/about/facts-figures>
- [18] Møller, A. and Schwartzbach, M. I. (2007). *Introduzione a XML*. Torino: Pearson Italia.
- [19] Bray, T., Paoli, J., Sperberg-McQueen, C. (1998). *XML 1.0 Specifications*. Retrieved August 29, 2012, from W3.org: <http://www.w3.org/TR/1998/REC-xml-19980210>
- [20] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., & Yergeau, F. (2008). *XML 1.0 Specifications (Fifth Edition)*. Retrieved August 29, 2012, from W3.org: <http://www.w3.org/TR/REC-xml/>