

ECP-2006-DILI-510003

TELplus

**Recommendations for full text indexing
in The European Library and
Software solution for a full text search engine**

Deliverable number	<i>D3.9-3.10 (combined)</i>
Dissemination level	<i>Public</i>
Delivery date	<i>26 January 2010</i>
Status	<i>Final</i>
Author(s)	<i>Jorge Machado (IST), Nuno Freire (BNP), José Borbinha (IST), Gilberto Pedrosa (IST), Diogo Reis (IST)</i>



eContentplus

This project is funded under the *eContentplus* programme¹,
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

¹ OJ L 79, 24.3.2005, p. 1.

Contents

1	INTRODUCTION	1
2	ADDITIONS TO THE EUROPEAN LIBRARY APPLICATION PROFILE FOR OBJECTS FOR TELPLUS	2
2.1	METADATA FOR LINKING TO FULL-TEXT FOR INDEXING	2
2.1.1	<i>Full Text [tel:fullText]:</i>	2
2.1.2	<i>Structural Metadata [tel:structuralMetadata]:</i>	3
2.2	METADATA TERMS FOR USE WITH HIERARCHICAL RECORDS	3
2.2.1	<i>Bibliographic Citation [dcterms:bibliographicCitation]:</i>	4
2.2.2	<i>Is Part Of [dcterms:isPartOf]</i>	4
2.2.3	<i>Identifier [dc:identifier]:</i>	5
3	THE FULL TEXT HARVESTER	6
3.1	GENERAL ARCHITECTURE	6
3.2	TECHNICAL REQUIREMENTS	7
3.3	INSTALLATION	7
3.4	OPERATION OF THE HARVESTER	7
3.4.1	<i>Commands line operations:</i>	7
3.4.2	<i>Other operations</i>	8
4	THE SEARCH ENGINE	10
4.1	SOLR INDEX AND SEARCH API	10
4.2	TECHNICAL REQUIREMENTS	11
4.3	GENERAL ARCHITECTURE	11
4.3.1	<i>The INDEXES:</i>	11
4.3.2	<i>The SCRIPTS:</i>	12
4.3.3	<i>The SOLR Engine:</i>	12
4.3.4	<i>Indexing History and Benchmark Tests:</i>	16
5	IMPLEMENTATION RECOMMENDATIONS	20
5.1	DISTRIBUTED INDEX VERSIONING FOR HIGH AVAILABILITY SEARCH ENGINES	20
5.2	UPDATES	20
5.2.1	<i>Recommendations</i>	20
5.2.2	<i>Committing between versions:</i>	20
5.3	SEARCH TIME	21
5.4	ARCHITECTURE	21
6	CONTENT ACQUIRED AND INDEXED	22

1 Introduction

As part of the TELplus project at least 20 million pages of text resulting from OCR were expected to be made available by the partners to The European Library, with the purpose of indexing them and thus improving overall search services.

In order to automate the processes, the partner libraries were expected to provide links in the metadata records to their full-text material. However, not all the partners could fulfill this requirement in time, so alternative methods had to be implemented for these cases. The alternative techniques used for this purpose were harvesting the texts by HTTP from links in HTML pages, or, as a last resort, manual file transfer.

This document reports the results of the developments to acquire these pages and index them, which comprise these main results:

- A new set of metadata elements for the TEL-AP (tel:fullText and tel:structuralMetadata)
- An implementation of a Full Text Harvester (implements the methods for linking to full text via the TEL-AP metadata records).
- An implementation of a Search Engine (a server application implemented using the SOLR open source indexer and retrieval system based on the Apache's Lucene text retrieval library).
- The harvesting and indexing of nearly 24 million pages consisting of collections from these partners: Austria, Czech Republic, Estonia, France, Hungary, Iceland, Latvia, Lithuania, Norway, Poland, Slovenia, Sweden.

Chapter 2 describes the metadata elements of the TEL-AP that were used in this context, including the newly defined ones.

Chapter 3 describes the Full Text Harvester, and chapter 4 describes the Search Engine. These descriptions assume that the reader has access to the code of the Full Text Harvester, provided by the IST to the TEL Office.¹

Chapter 5 provides extra relevant recommendations to take into consideration for the implementation of a search engine, based on the experience from this action.

Finally, chapter 6 reports the collections harvested and indexed in the moment of writing of this document.

It is expected that readers of this document will have access to the software provided by the IST and be knowledgeable about the technologies used (Maven 2, SOLR, SRU, etc.) and the indexing techniques mentioned, especially text indexing with snippets.

¹ The full-text harvester code will be added to The European Library's source code repository. Access to this code is available on request by contacting Sally Chambers (sally.chambers@kb.nl).

2 Additions to The European Library Application Profile for Objects for TELplus

The detailed processes to be implemented and the requirements for that were defined in a TELplus WP1 meeting held in The Hague in March 2009. This chapter reports the decisions concerning The European Library Application Profile for Objects.

2.1 Metadata for linking to full-text for indexing

After the initial analysis of the purpose and the methods to reach it, it was concluded that two new metadata elements should be added to The European Library Application Profile for Objects for this purpose:

- tel:fullText
- tel:structuralMetadata

2.1.1 Full Text [tel:fullText]:

Name	fullText				
Type	Property				
Qualified Name	tel:fullText				
Source label	Not Applicable				
Label	Full Text				
Defined by	The European Library Application Profile for Objects v2.0				
Identifier	[TEL_Identifier]fullText				
Status	Proposed				
Source definition	Not Applicable				
TEL definition	A URL linking to a full-text document for indexing.				
TEL guidelines	<p>The metadata element tel:fullText is used for partner libraries to provide a link to the full-text that they would like to be included in The European Library. This element should include a link (URL) to a full-text document.</p> <p>The links contained in tel:fullText will not be shown to users. If the same URL is also used for providing access to the full-text for users, using the “See Online” functionality, this URL should be included in both elements.</p> <p>The full-text materials will be located in The European Library by the existence of tel:fullText. Records that do not contain tel:fullText or tel:structuralMetadata are considered to be metadata-only records. Even if such records link to full text via the dc:identifier element, they will not be indexed.</p>				
Sub-property of:	not applicable	Is refined by:	not applicable	Encoding scheme:	URI
Obligation:	Mandatory for access to full-text	Repetition:	Yes	Data type:	URI
Functions:	Administrative Accessing full-text				

2.1.2 Structural Metadata [tel:structuralMetadata]:

Name	structuralMetadata				
Type	Property				
Qualified Name	Tel:structuralMetadata				
Source label	Not Applicable				
Label	Structural Metadata				
Defined by	The European Library Application Profile for Objects v2.0				
Identifier	[TEL_Identifier]structuralMetadata				
Status	proposed				
Source definition	Not Applicable				
TEL definition	A URL linking to a file with structural metadata. This file contains the URLs linking to full-text documents for indexing.				
TEL guidelines	<p>The metadata element tel:structuralMetadata is used for partner libraries to provide a link to a file with structural metadata. Any structural metadata format may be used, since The European Library will only use the URLs linking to full-text documents for indexing.</p> <p>The links contained in tel:structuralMetadata will not be shown to users.</p> <p>The full-text materials will be located in The European Library by the existence of tel:structuralMetadata. Records that do not contain tel:structuralMetadata or tel:fullText are considered to be metadata-only records. Even if such records link to full text via the dc:identifier element, they will not be indexed.</p>				
sub-property of:	not applicable	Is refined by:	not applicable	Encoding scheme:	URI
Obligation:	Mandatory for access to full-text	Repetition:	yes	Data type:	URI
Functions:	Administrative Accessing full-text				

2.2 Metadata terms for use with hierarchical records

In addition, for those partners who want to make the full-text of newspapers available, The European Library proposed a solution for creating links between the newspaper (parent) record and the newspaper issues (child) records. For this solution, the existing metadata elements were required:

- **dc:terms:bibliographicCitation**
- **dc:terms:isPartOf**
- **dc:identifier**

The European Library provided additional guidelines to assist partners creating the links between the parent and child records.

2.2.1 Bibliographic Citation [dcterms:bibliographicCitation]

Name	bibliographicCitation				
Type	property				
Qualified Name	dcterms:bibliographicCitation				
Source label	Bibliographic Citation				
Label	Bibliographic Citation				
Defined by	Dublin Core Library Application Profile http://dublincore.org/documents/library-application-profile/index.shtml#BibliographicCitation				
Identifier	http://purl.org/dc/terms/bibliographicCitation				
Status	in use				
Source definition	Bibliographic citation information for a journal article, or similar bibliographic resource				
TEL definition	The label containing bibliographic citation information that you want to display to the user. It is used for linking back to the parent metadata record.				
TEL guidelines	<p>For creating links between hierarchical records; records that have both a parent and a child element, e.g. a (parent) record for the newspaper which is linked to individual (child) records for the different issues of the newspaper. dcterms:bibliographicCitation should be used in the “child” record. It should contain the bibliographic citation information (text) that you want to display to the user. It is used for linking back to the parent record together with the dcterms:isPartOf element.</p> <p>DCMI provides Guidelines for Encoding Bibliographic Citation Information in Dublin Core Metadata that may be useful.</p>				
Sub-property of:	dc:identifier	Is refined by:	not applicable	Encoding scheme:	not applicable
Obligation:	Mandatory for linking records	Repetition:	no	Data type:	Text and numbers
Functions:	Navigation Object level usage Linking Records				

2.2.2 Is Part Of [dcterms:isPartOf]

Name	isPartOf				
Type	property				
Qualified Name	dcterms:isPartOf				
Source label	Is Part Of				
Label	Is Part Of				
Defined by	Dublin Core Metadata Initiative http://dublincore.org/documents/dcmi-terms/#terms-isPartOf				
Identifier	http://purl.org/dc/terms/isPartOf				

Status	in use				
Source definition	A related resource in which the described resource is physically or logically included.				
TEL definition	Contains the unique identifier of the parent record.				
TEL guidelines	For creating links between hierarchical records; records that have both a parent and a child element, e.g. a (parent) record for the newspaper which is linked to individual (child) records for the different issues of the newspaper. dcterms:isPartOf should be present in the child record and contain the unique identifier of the parent record (provided in dc:identifier). It is used for linking purposes together with the dcterms:bibliographicCitation element.				
Sub-property of:	dc:relation	Is refined by:	not applicable	Encoding scheme:	not applicable
Obligation:	Mandatory for linking records	Repetition:	no	Data type:	text and numbers
Functions:	Navigation Object level usage Linking Records				

2.2.3 Identifier [dc:identifier]:

Name	identifier				
Type	property				
Qualified Name	dc:identifier				
Source label	Identifier				
Label	Identifier				
Defined by	Dublin Core Metadata Initiative http://dublincore.org/documents/dcmi-terms/#terms-identifier				
Identifier	http://purl.org/dc/terms/identifier				
Status	in use				
Source definition	An unambiguous reference to the resource within a given context.				
TEL definition	The unique identifier of the record				
TEL guidelines	For creating links between hierarchical records; records that have both a parent and a child element, e.g. a (parent) record for the newspaper which is linked to individual (child) records for the different issues of the newspaper. dc:identifier should be used in both the parent and the child record to provide the unique identifier of the record for linking purposes.				
Sub-property of:	not applicable	Is refined by:	not applicable	Encoding scheme:	not applicable
Obligation:	Mandatory for linking records	Repetition:	no	Data type:	text and numbers
Functions:	Navigation Object level usage Linking Records				

3 The Full Text Harvester

The Full Text Harvester implements the methods for linking to full text via the TEL-AP metadata records as agreed during the TELplus WP1 meeting in The Hague in March 2009.

The harvesting is done based on TEL-AP metadata records previously harvested by OAI-PMH, by web crawling a page that contains the links to the full text documents for those cases where no metadata exists, or by file transfer if no other automated means could be used.

When the harvesting is performed based on TEL-AP metadata, the harvester uses the TEL-AP elements <tel:fullText> and <tel:structuralMetadata> for discovering where to download the full text files associated to a metadata record. These elements contain links to full text materials or structural metadata (which will contain links to the actual full text files).

It supports the harvesting of PDF, HTML and plain text files. In the case of PDF files, the harvester downloads the files, extracts the text and the PDF original file is discarded.

3.1 General architecture

The Full Text Harvester consists of an application that receives commands from an operator via a command line. Each command is executed by an independent process. The operator should not execute possible conflicting commands simultaneously, as it may create inconsistencies in the harvester's repository.

The harvester is configured by an XML file. This file allows the operator to specify the parameters of the harvester, to register all the full text collections to be harvested, and also to specify the parameters for each collection independently.

An example of a configuration is provided with the software, in the path:

- `"/fullTextHarvester-config.xml"`

The collections to be harvested have to be provided as a file system folder containing metadata records in the TEL-AP format, or alternatively by a URL to a web page that contains the links to the full text documents. These folders and URLs must be specified in the configuration file.

The operator can execute the commands for harvesting each collection. The duration of the harvest of one collection depends of the size and the file formats and can range from a couple of hours to several weeks. Therefore, the harvester should typically be executed as a background process.

During the harvesting process the harvester stores all text files and associated TEL-AP records in its local repository. This repository consists of a folder on the file system. This folder contains one subfolder for each collection, and inside the collection folder, the harvested objects are spread in subfolders to avoid having many files on one folder.

Log files of the harvests can also be found in the parent folder of the harvester's repository.

Once a collection's harvest is finished, it can be exported to files in the proper input format required for SOLR. This export is initiated by the operator with the appropriate command.

The SOLR files are generated in the harvester's repository in the subfolders "solr" and "solrPages".

3.2 Technical requirements

The software was structured and built as a Maven 2 project, and requires Java 1.6 to run.

The disk requirements depend on the amount of text being harvested. During its usage in TELplus, approximately 600 GB were necessary during the harvesting of nearly 30 million pages.

The memory requirements can reach the 1GB of RAM for the JRE, therefore it should be run on a computer with at least 2 GB of RAM.

The software was used in the Linux operating system.

3.3 Installation

The installation of the Full Text Harvest consists in the following:

1. Install of a Java JRE 1.6
2. Make sure that java command is in the PATH system variable.
3. Unzip the distribution zip archive file in the installation folder.
4. Choose and create a folder in the file system for the harvester's repository.
5. Edit the file "fullTextHarvester-config.xml" and configure the collections for harvesting. The file contains details on how to configure the harvester and the collections.
6. Give executable permissions to the *.sh files on the installation folder (ex: "chmod u+x *.sh")
7. Make available on the file system the source metadata records in TEL-AP, in the locations configured in the "fullTextHarvester-config.xml" file.

3.4 Operation of the harvester

Operation of the harvester is performed by executing commands over the command line, and by direct manipulation of the repository of the harvester on the file system.

3.4.1 Commands line operations

After the installation, three shell scripts are located on the installation folder. These are the commands that the harvester makes available for managing the full text harvests.

All commands are executed via shell scripts. The scripts can be customized by the operators. The default scripts assume the following:

- Java is in the PATH;
- The configuration file is on the same folder and is named "fullTextHarvester-config.xml";

- The commands are to be executed in the background;
- The output is redirected to a file.

3.4.1.1 RunHarvester.sh

This command starts the full text harvesting of one collection. It generates a log in the harvester's repository named "collection.harvest.log". Its output is redirected to a file named "out-harvester-collection.txt"

The command line parameters are:

- RunHarvester.sh *collection*

3.4.1.2 ExportToSolr.sh

This command starts the exporting of a previously harvested collection to xml files formatted according to the input format of SOLR. The SOLR files are generated in the harvester's repository, in the subfolders "solr" and "solrPages". Its output is redirected to a file named "out-export-solr-collection.txt"

The command line parameters are:

- ExportToSolr.sh *collection*

3.4.1.3 DisplayStatistics.sh

This command displays statistics about the harvested contents of a Collection. It calculates:

- The number of objects with full text
- The total number of text files
- The total number of files per file type
- The number of pages

Calculation of the number of pages is not possible for all file formats, and only PDF is supported in a reliable way. Counting of pages of HTML files only works for BnF.

The command line parameters are:

- DisplayStatistics.sh *collection*

3.4.2 Other operations

Other operations on the full text harvester are not available via commands, and have to be executed by direct manipulation of the harvester's repository on the file system, or via operating system commands.

3.4.2.1 Stopping running operations

If the operator wants to interrupt a command while it is executing, that can be done safely by terminating the process using the operating system command. Stopping the commands in this way, is unlikely to create any inconsistencies in the harvester.

3.4.2.2 Cleaning the repository

When a collection that was previously harvested needs to be harvested again, the previous harvest must be deleted from the harvester's repository. If it is not deleted, the harvester will only harvest the new full text objects that exist in the repository.

To completely delete a collection these three folders should be deleted:

- *harvester_repository/collection*
- *harvester_repository/solr/collection*
- *harvester_repository/solrPages/collection*

4 The Search Engine

The Search Engine is a server application implemented using the SOLR open source indexer and retrieval system based on the Apache's Lucene text retrieval library. This system is able to index XML documents previously generated by the Full Text Harvester and provide search and other services over them.

Examples of services provided by this Search Engine are the faceted search, snippets generation and word suggestion. SOLR provide and HTTP/XML API to index and search over the indexes.

Typically, SOLR is not the final User Interface application. Another application is needed to request a search query to SOLR and transform the results into a human readable format. In our case we have a middle application, named TELHandler, which requests the search queries to SOLR and transforms the answers into the SRU/SRW XML protocol. Finally, the TEL Client applications will use the TELHandler SRU/SRW API.

4.1 SOLR Index and Search API

The XML documents indexed by SOLR are sets of fields structured in the SOLR schema:

```
<docs>
  <doc>
    <field name="id">urn:example:id</field>
    <field name="text">Example of text content</field>
    ...
  </doc>
  ...
</docs>
```

The Full Text Harvester is able to transform the initial text files in the SOLR ingest format, according to its required schema.

The SOLR allows a fully distributed search engine using separated instances, known as *shards*, which could be installed in the same or in separated machines. In case of using the same machine to install two shards the servers need to be configured using different PORTS (see SOLR documentation). Each instance can handle with one or more indexes defined as *cores*. A core works as a separated instance inside the same web server (the same PORT). Each instance could be used as a point of access to itself and/or to other instances through an HTTP/GET → XML service.

To build a distributed architecture the client application just needs to set a GET parameter to identify the requested shards.

The following example illustrate a scenario of a client requesting a query to the server instance *localhost:7480/solr/Iceland* defining as shards the same instance and another *localhost:7480/solr/France*:

```
http://localhost:7480/solr/Iceland/select?shards=localhost:7480/solr/Iceland,localhost:7480/solr/France&q=world war&.....
```

In this scenario both shards are in the same server (localhost:7480) but in two separated cores (Iceland and France).

If the system becomes too slow it is possible to put each core in a different server machine just by changing the client's application URL. For example, to move the server application for the collection "France" to the machine with IP 192.168.1.23 the previous request URL, in the client, could be like this:

```
http://localhost:7480/solr/Iceland/select?shards=localhost:7480/solr/Iceland,192.168.1.23:7480/solr/France
&q=world war&.....
```

4.2 Technical requirements

The software is already compiled using the version 1.4 of SOLR which brings together a Java server Jetty, and requires Java 1.6 to run.

The disk requirements depend on the amount of text being indexed. During its usage in TELplus, approximately 500GB were necessary during the index of nearly 30 million pages.

The memory requirements can reach the 1GB of RAM for the JRE, therefore it should be run on a computer with at least 8GB of RAM.

The software was used in the Linux operating system.

4.3 General Architecture

The retrieval system is placed at the SOLR_BASE folder and is structured in 3 main parts:

- The INDEXES folder
- The index SCRIPTS folder
- The SOLR engine

4.3.1 The INDEXES

The INDEXES were structured into PAGES and FULLTEXT. This was done because the snippets generation algorithm provided by SOLR is very slow for big documents. The process of searching was changed to search first in FULLTEXT indexes and then request the best pages for each TOP document and generate the snippets using the page. There were a few performance tests which led to this decision. Those tests are reported below in this section.

Inside INDEXES/FULLTEXT and INDEXES/PAGES were created the following specific indexes:

- Austria
- Estonia
- France
- Hungary
- Iceland

- Latvia
- Lithuania
- Norway
- Poland
- Slovenia
- Sweden
- CzechRepublic_monograph

The Iceland PAGES index was separated into four sub indexes, as it was very large.

4.3.2 The SCRIPTS

The SCRIPTS folder contains a folder *include* which has the script files to generate one file for each sub index creation.

With the scripts already created, in order to generate the indexes for Iceland pages, for example, the file *pagesIceland.sh* should be executed. To create the indexes of Iceland's full text, the file *fulltextIceland.sh* should be executed. To do that see the README_IST file placed inside the SOLR_BASE APP.

4.3.3 The SOLR Engine

The SOLR engine application is already configured and just needs to be started in order to run the server applications. The application includes:

- The TELHandler SRU server
- The SOLR FULLTEXT search engine
- The SOLR PAGES search engine

The TELHandler application was developed to serve the SRU requests. The SOLR instances were configured to use one *core* for each sub index created, one for each country. In this way if the system becomes to slow it is possible to separate the *cores* in different machines.

The TELHandler is placed in the FULLTEXT server instance at *webapps/telHandler* folder. SOLR instances are in the *webapps/solr.war*. TELHandler provides a configuration file at:

"/webapps/telHandler/WEB-INF/classes/eu/europeana/tel/conf.properties"

4.3.3.1 TEL Handler

The internal architecture of *telHandler* is very simple. First of all it parses an SRU query with a lexical and syntactic CQL (Common Query Language) parser. In second place it creates a final query for SOLR/Lucene query API using the configuration provided in file *conf.properties*. Thirdly it sends a request query and facets generation to the FULLTEXT server which can be a distributed set of shards. For each TOP result it sends a query to PAGES engine requesting the best pages for that query. This request carries a snippets generation parameter. Finally it merges the snippets with the FULLTEXT response and creates the final SRU response using a XSLT transformation which uses a XSL placed at:

/webapps/telHandler/WEB-INF/classes/eu/europeana/tel/sru.xsl

By default the Ngrams query is requested to SOLR but TEL Handler provides the following services:

- Get SRU response with Ngrams query
- Get SOLR response with Ngrams query
- Get Query with NGrams to Lucene
- Get Extended CQL with NGrams
- Parse CQL level 2
- Get SRU response with LuceneTranslator query
- Get SOLR response with LuceneTranslator query
- Get LuceneTranslator query (using Level 3 CQL)
- Parse CQL level 3

The configuration of TELHandler very simple and modular. The configuration provides the follow possibilities:

- The SRU Fields mapping to Lucene Fields.
- Stemming N-GRAMS techniques configuration or none.
- Boosting factors of each SRU field
- The URL of the FULLTEXT point of access
- The URL of the PAGES point of access (Which can be turned off in case of the snippets generation is made at FULLTEXT engine)

The configuration is already set to use all cores in both FULLTEXT and PAGES server and to use Ngrams linear combination ranking scheme. This stemming technique was evaluated in TEL@CLEF task of Cross Language Evaluation Forum getting the best results between all participants. The working notes of our experiment are available at clef campaign web site¹ but a reviewed version will be published in the Springer LNCS issue with the proceedings of the CLEF 2009 workshop.

4.3.3.2 Examples

Figure 1 illustrates the interface for the TEL Handler.

Figure 2 shows the query request to SOLR for the service *Get Query with Ngram to Lucene*.

The same query, but structured for better visual inspection, can be requested using the service Get Extended CQL with NGrams. That is illustrates in Figure 3. In this example the SOLR response will be given by the screenshot in Figure 4. Figure 5 illustrates the response in SRU

¹ www.clef-campaign.org/2009/working_notes/machadoCLEF2009.pdf

schema.

But if we request the default service, illustrated in

Figure 5, the response given by TELHandler comes in SRU schema and each record brings the best snippets.

TEST FORM for SRU service

CQL Fields: (author, title, subject, description, date, language, fulltext)
CQL Operators (for CQL Level2): AND, OR, NOT (can also use brackets for associativity purposes)

Query	<input type="text" value="fulltext=drame OR dc.creator=durand"/>
Mode	<input type="text" value="Get SOLR response with Ngrams query"/>
Start Record	<input type="text" value="1"/>
Maximum Record	<input type="text" value="10"/>

Figure 1 – SRU Test Form Services

```
<telHandler>
+ <comments></comments>
- <lceneQueryExtended cql_level="2" cql_generator="eu.europeana.tel.cql.parser.CQLParser" extensionType="ngrams">
  (text:(drame)^0.53 text_5:(drame)^0.26 text_4:(dram rame)^0.12 text_3:(dra ram ame)^0.06)^1 OR (dc_creator:(durand)^0.53 dc_creator_5:
  (duran)^0.26 dc_creator_4:(dura)^0.12 dc_creator_3:(dur)^0.06)^3
</lceneQueryExtended>
</telHandler>
```

Figure 2 - Lucene Query with Ngrams.

```
<telHandler>
+ <comments></comments>
- <xcqlExtended cql_level="2" cql_generator="eu.europeana.tel.cql.parser.CQLParser" extensionType="ngrams">
  - <triple>
    - <boolean>
      <value>OR</value>
    </boolean>
    - <leftOperand>
      - <searchClause generator="eu.europeana.tel.cql.ngrams.NGramsConverter">
        <index boost="1">fulltext</index>
        - <relation>
          <value>=</value>
        </relation>
        <term>drame</term>
      - <fieldQuery field="fulltext" targetField="text">
        <configuration minGramSize="3" maxGramSize="5" stemmingType="pure"/>
        <subQuery boost="0.53" targetField="text">drame</subQuery>
        <subQuery boost="0.26" targetField="text_5">drame</subQuery>
        <subQuery boost="0.12" targetField="text_4">dram rame</subQuery>
        <subQuery boost="0.06" targetField="text_3">dra ram ame</subQuery>
      </fieldQuery>
    </searchClause>
    </leftOperand>
    - <rightOperand>
      - <searchClause generator="eu.europeana.tel.cql.ngrams.NGramsConverter">
        <index boost="3">dc.creator</index>
        - <relation>
          <value>=</value>
        </relation>
        <term>durand</term>
      - <fieldQuery field="dc.creator" targetField="dc_creator">
        <configuration minGramSize="3" maxGramSize="5" stemmingType="front"/>
        <subQuery boost="0.53" targetField="dc_creator">durand</subQuery>
        <subQuery boost="0.26" targetField="dc_creator_5">duran</subQuery>
        <subQuery boost="0.12" targetField="dc_creator_4">dura</subQuery>
        <subQuery boost="0.06" targetField="dc_creator_3">dur</subQuery>
      </fieldQuery>
    </searchClause>
    </rightOperand>
  </triple>
</xcqlExtended>
</telHandler>
```

Figure 3 - TEL Handler Response for Ngrams Query Generation.

4.3.3.3 SOLR PAGES and FULLTEXT instances

For SOLR, FULLTEXT and PAGES, a set of configurations in the *multicore* folder was created. This folder contains one *solr.xml* file declaring the *cores* and one folder that is a replica for each core. It is also where the index schema and the SOLR *core* configuration are defined.

The indexing schema is available at *multicore/[core]/conf/schema.xml*. All text fields were indexed using three separated indexes. One for fragmented words with size 4 characters, another for fragmented words with size 5 characters and another for original words. This will be used by TELHandler application to build clever queries independent from document languages. The SOLR configuration is adapted to TEL requirements and is available at *multicore/[core]/conf/solrconfig.xml*. It is not recommended to change this files because that could cause integration problems between TELHandler and SOLR instances.

4.3.4 Indexing History and Benchmark Tests

This section provides a set of explanations and results which were the basis for selecting the chosen architecture for the TEL Search Engine. We go on to justify the multicore architecture and the reason for the separation between PAGES and FULLTEXT. Finally we show some benchmark tests performed to three SOLR configurations.

4.3.4.1 SOLR Multicore

The indexing history began with the indexation of all TEL harvested collections using one single instance of SOLR search engine with just one core. This means that we started using just one single index folder. Initially this approach didn't reveal any problems taking into account that the SOLR administration API provides incremental indexing at document granularity. However several machine denials of service caused by full disk space and electrical problems showed that the index could become corrupted at indexing time causing the indexing process, which takes several days, to start again from the beginning. For that reason we decided to keep separated indexes at collections level. With this policy the indexes become lighter and easier to maintain. Initially we thought that this policy could be detrimental to the speed of the search engine. However, we subsequently found that the system operated more slowly when very large indexes were being used in comparison to when using separated ones. This was very clear for us because with a very large index (e.g.300 Gigabytes) we could perform searches taking several minutes.

4.3.4.2 PAGES index and Snippets

Beside the problems reported above, there was another issue with the snippets generation. Snippets are small fragments of text retrieved from the documents that are relevant to the user query. They help the user to understand the context in which the query term is found in the retrieved document. The generation of snippets requires full storage of the full text of the documents because snippets are generated at runtime.

The algorithm provided by SOLR is very slow when used in very big documents because it is based in regular expressions. It is possible that future versions of SOLR might provide new algorithms to do this task.

We considered the possibility of removing the snippets because creating snippets increases the response time from several seconds to minutes even creating the snippets just for the 10 shown documents. However, snippets were considered to be very important to the project's success so we tried to find a technique to solve the problem with the available technology. We found that splitting the documents into small pages could be a solution if we indexed each page as a separate document without the full text. The process was based on two steps, first do a full text search to retrieve the most relevant documents and then do a new search, this one with snippets, in pages field to retrieve the best 3 pages for each top document. The snippets were very quick to produce because a page is a very small fragment of the document.

Another problem was that the number of documents increased exponentially considering that each page was a document and considering an average of 100 pages per document. Beside that the full text fields didn't need to store the text any more, the index size increases 50 times because of the number of documents. To solve the snippets problem the resulting index size takes us again to our first problem, the speed of the responses increase again. For that reason we decided to have a separate instance for pages and another for full text. In this way the full text index was small and the pages index was bigger but not very big. We also found that pages search queries were quicker because the query was restricted to one document and because the logic AND makes the index lookups operate much more efficiently.

4.3.4.3 Benchmark Evaluation

All of our benchmark evaluation was done using just the Austrian collection because it was one of the biggest collections. The configurations of the three tests are shown in Table 1. Our best results were obtained with a separated instance with an index of pages to generate the snippets. This solution is the one which needs most disk space.

Table 1: Configurations for snippets generation using page field.

	Conf. 1	Conf. 2	Conf. 3
Multi fields			NotStored/Indexed
One field <i>fulltext</i>	NotStored/Indexed	NotStored/Indexed	
One field <i>page /</i>	Stored/Indexed	separated instance	separated instance
Snippets Generation	10 queries in <i>page</i> filtering docID	10 queries in separated instance <i>page</i> filtering docID	10 queries in separated instance <i>page</i> filtering docID
Index Size / GB	25	4	4
Separated		15	15

Table 2: Configurations for snippet generation using the fulltext field.

	Conf. 4	Conf. 5
Multi fields <i>fulltext</i>		Stored/Indexe
One field <i>fulltext</i> / doc	Stored/Indexe	
Snippets Generation	in <i>fulltext</i>	in <i>fulltext</i>
Index Size / GB	9	9

Configuration 1 was the first approach using just one instance, one index, for all documents and page documents. The second configuration use the same approach but the pages index was created in a separated instance running in the same machine.

The third approach was a variation of the second approach where we split the full text in several full text fields, which in practice will be the same for us in terms of search process but could possible change the performance at Lucene level (Table 2). We also tried this approach at Configuration 5 to make a final attempt to generate the snippets in full text index but the response times where again very poor. Finally, the Configuration 4 is reported here because it was our first approach using just one index and indexing the original full text to generate snippets. This was the worst configuration.

It is important to note that we used this performance test using just one machine with separate server instances using different ports. Independent machines for instances possible will increase the results because we note that CPU was always processing at 100%.

We evaluated 8 queries, where the last one was very difficult to answer by the system because it included several logical OR operators and several different words, as we can see:

Q1: fulltext=Kundmachung AND (dc.creator="durand" OR dc.title=drame OR fulltext=Aufgebotesbei)
 Q2: fulltext=treibenden AND (dc.creator="durand" OR dc.title=drame OR fulltext=Partei)
 Q3: fulltext=Weiteren AND (dc.creator="durand" OR dc.title=drame OR fulltext=Volkymnmisfen)
 Q4: fulltext=Josef OR fulltext=Wedlschen OR fulltext=Erben OR fulltext=Belrage OR fulltext=im AND Q3:
 (dc.creator="durand" OR dc.title=drame OR fulltext=Volkymnmisfen)
 Q5: fulltext=Gericht OR fulltext=meldetoder OR fulltext=einen
 Q6: fulltext=Mitgliede OR fulltext=Nervlltungsratcs OR fulltext=odereinem
 Q7: fulltext=gleichnamigen OR fulltext=odereinem
 Q8: fulltext=baranyllvar AND fulltext=odereinem
 Q9: fulltext=gtsttzmtwuif AND fulltext=odereinem
 Q10: fulltext=Nervlltungsratcs AND fulltext=odereinem

Each query was submitted to the 5 configurations resulting in the times reported in Table 3. All results were obtained in the first interaction when the SOLR cache results was empty.

Query Q2 uses less frequent terms than the other queries and that is visible in the results. All configurations perform well with that query. The term “treibenden” occurs only in 2215 documents of Austria collection and probably all documents that include it are in near segments of Lucene index.

The queries Q8, Q9 and Q10 use words already used in previous queries to test the caching performance in all configurations versus the architecture. As we show, configuration 2 using separated instances for full text and pages is the better solution. Configuration 1, using just

one index for pages and full text, is slower and we think that is because the index is very big. Even in Q10 where words, which uses words already used in previous queries, so we expect system to have them in cache, Configuration 1 and 3 can't recover as well as Configuration 2. Informally we ran some tests with 3 Estonian and Lithuanian collections and we found that the Configurations 1 and 3 take minutes to answer the queries while configuration two grows linearly.

Table 3: Query response times in milliseconds (first interaction)

Query/Conf	Conf 1	Conf 2	Conf 3	Conf 4	Conf 5
Q1	6858	5624	10865	9279	16284
Q2	3602	3729	3787	2096	4228
Q3	4332	3519	10921	8812	14796
Q4	4932	3787	7196	6138	10346
Q5	6453	4664	9753	10304	9723
Q6	5435	3646	8654	12852	11234
Q7	4536	2773	6896	7535	9934
Q8	3987	1346	2356	9067	10345
Q9	4678	2172	3543	11234	7856
Q10	3467	1830	4345	7852	10234

5 Implementation Recommendations

In this section we report relevant recommendations to take into consideration for the implementation of a search engine. These recommendations are founded on the literature and on our practical experience.

5.1 Distributed Index Versioning for High Availability Search Engines.

Search engine platforms have two main jobs: update and search an index. The first job is directly related with availability because the index, that is the heart of the system, will change. If we don't take careful precautions the index could become corrupted and search service will be unavailable. For that reason the update should be versioned. In distributed environments this problem increases because there are more points of failure. To solve this problem the system should be capable of performing transactional updates over distributed nodes. This can be accomplished using versioned indexes but not only with it.

5.2 Updates

5.2.1 Recommendations

Information Retrieval Systems have the capability to offer the same levels of availability as regular database systems. Usually the task of updating is controlled by the server and not by client. The indexes remain for long periods of time without any kind of update. The problem we want to solve is to replace the index while a query is being executed without stopping the service. Replacing an index implies to replace Giga or even Terabytes of data. Below we specify a set of precautions and guidelines to have in mind when an update system is being developed:

- The old copy of the index should always be kept until the availability confirmation of the new index. Indexing technologies are not as stable as the old DBMS's. In some situations index could be irreparably corrupted. For this to happen just needs a simply write operation not to run well in a specific file and our index could be damaged.
- The system should be capable of working with different versions at same time to guarantee that queries being performed when the update is being done are always answered.
- When we are using disjoint clusters the system should always use a cluster confirmation from all nodes before do something with the new index.

We will now explain a simple technique to commit between index versions.

5.2.2 Committing between versions

We don't want to stop the service even for a second because we never know if the system will start again after a stop and because in the majority of cases the restart takes a few seconds. To

accomplish this we should test index correctness and respond with a confirmation. In clusters the system should wait for confirmation from all nodes. After that we need to bring up the new index and ready it to perform search results. A good policy is to test index correctness using the regular search interface against the new index. At the end the system will need to switch between versions. The objective is to accomplish that without major system status modifications. This can be done using a middle layer between user and server. The system should keep an index version number to pass from the user to the server layer. When the new version is operational and tested the version number is updated in the middle layer and the next query will use transparently the new index version. This means that the server should be capable to keep an array of at least two opened indexes at same time. When no more queries are being executed in the old index the system can close it and eventually delete it.

5.3 Search Time

Search time can be improved both using distributed indexes and a merged central server but this will require always a set of benchmark tests to show the effective improvement. Some thoughts¹ and experiments lead us to assume that better results can be obtained with extensive caching of pre-computed partial hit lists for frequent terms and phrases².

5.4 Architecture

Scalable search engines can be created using several proposed architectures. Some possibilities are, for example, distributed indexes, distributed processing in server replicas or both. The distributed index is achieved distributing sets of documents into a cluster of machines. Each machine builds its index based on its own set of documents. Results are merged in merge servers. This architecture requires that every cluster node compute a partial search result. The other possibility is to replicate the index on every machine in the cluster and create a query plan to compute, for example, each word inverted list in different machines and use intersection or union to merge them later in the final results.

The distributed processing architecture does not affect the relevance score function, but that does not happen with distributed indexes. Some of the calculations require information about the entire index, such as the term frequency. This information is not available when the partial indexes are built independently.

¹ <http://www.opensubscriber.com/message/java-dev@lucene.apache.org/8554277.html>

² Xiaohui Long. "Three-level caching for efficient query processing in large web search engines", Proc. of the 14th Int. World Wide Web Conference. (2005)
[<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.2342>]

6 Content acquired and indexed

Table 4 reports the collections acquired and indexed in the moment of writing of this document.

Table 4: Report of the acquired collections and the indexing process

Provider	Acquisition Process				SOLR files size (in Kbytes)
	Method	last harvest	Coverage	Pages	
Austria	OAI-PMH	29-10-2009	100%	511.000	7.113.648
Czech Republic	OAI-PMH	23-10-2009	PARTIAL	2.579.511	4.727.796
Estonia	OAI-PMH	10-10-2009	100%	713.933	9.533.640
France	Manual	28-10-2009	100%	8.242.908	57.706.636
Hungary	OAI-PMH	13-11-2009	100%	237.914	2.094.880
Iceland	OAI-PMH	10-10-2009	100%	5.727.149	111.459.064
Latvia	OAI-PMH	05-11-2009	100%	195.075	6.165.564
Lithuania	HTML/HTTP	23-10-2009	100%	125.477	2.225.904
Norway	OAI-PMH	15-10-2009	100%	1.600.000	1.293.160
Poland	OAI-PMH	10-10-2009	100%	436.198	1.419.316
Slovakia	HTML/HTTP	08-11-2009	PARTIAL	185.000	0
Slovenia	OAI-PMH	02-11-2009	100%	328.502	5.890.612
Spain	OAI-PMH	09-11-2009	PARTIAL	3.033.525	10.275.680
Sweden	OAI-PMH	25-10-2009	100%	1.409.095	8.876.824
Total				25.325.287	228.782.724

Offline transfers of extra contents from the collections of the Czech Republic and the overall collection of the Slovakia were still in progress in the date of this report (total numbers yet to be counted, but for the case of the Czech Republic it already was estimated to be of nearly 3 million pages more). Also the harvesting of plus 4,8 million pages from Spain is in progress. **The work to complete these actions will continue in early 2010, after the end of the TELplus project.**

For France it was not possible to implement OAI-PMH in time for TELplus, so manual metadata export was used.

Finally, for Sweden we faced problems with the scalability of the OAI-PMH service, so the metadata also was sent by file manual transfer.