Abu-MaTran

Automatic building of Machine Translation

PIAP- GA-2012-324414

D3.2. Corpora Acquisition Software

Dissemination level	Public
Delivery date	2014/09/20
Status and version	Final, v1
Authors and affiliation	Vassilis Papavassiliou, Prokopis Prokopidis (ILSP), Miquel Esplà-Gomis (UA), Sergio Ortiz (Prompsit Language Engineering)

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community under the Seventh Framework Programme for Research and Technological Development

Table of Contents

1 Executive Summary

This deliverable documents two open-source crawlers, Bitextor and ILSP Focused Crawler, which were extended and enhanced in the context of Abu-MaTran's WP 3: Rapid Acquisition for MT. The bilingual Croatian--English (HR—EN) datasets acquired using these crawlers (which will be described in D.3.1b: Acquisition for the second development cycle, due in M24) will be used in the MT systems developed in WP 4 (Development and deployment of MT) during the project's second development cycle.

Apart from describing the main components of the software toolkits, this deliverable includes hands-on information on how to get, install and use them.

2 Introduction

This deliverable documents two open-source crawlers, Bitextor and ILSP Focused Crawler, which were extended and enhanced in the context of Abu-MaTran's WP 3: Rapid Acquisition for MT. The bilingual Croatian--English (HR--EN) datasets acquired using these crawlers (which will be described in D.3.1b Acquisition for the second development cycle, due in M24) will be used in the MT systems developed in WP 4 (Development and deployment of MT) during the project's second development cycle. In Sections 2 and 3, we describe Bitextor and ILSP-FC, respectively. We conclude and mention further work and possible enhancements in the last section.

3 Bitextor

Bitextor is a free/open-source tool for automatically harvesting bitexts (parallel documents) from multilingual websites. This tool downloads a website (applying a filter to keep only those files written in HTML or XML) and process it in order to detect parallel documents. Bitextor has been developed as a highly modular tool consisting of a collection of scripts, mainly written in Python and Bash, that run simultaneously on Unix pipeline. This architecture allows to easily parallelize the different sub-tasks for an optimal use of the available resources, as well as to replace or add new modules for specific purposes. The process for crawling parallel data from a website is divided in four main stages: website downloading, target document selection and cleaning, parallel document identification, and sentence alignment. The first step can be omitted, so Bitextor can be used to crawl already downloaded data.

Bitextor has been developed in collaboration between the University of Alacant[1] and the company Prompsit Language Engineering,[2] and it is at the 4.1 version currently. Its code is publicly available under the GNU GPL v3 license on Sourceforge.[3]

3.1 Bitextor Architecture

Bitextor is a highly modular tool, implemented following the Unix philosophy: each module is implemented as an independent tool, and they communicate to each other by using text interfaces on a Unix pipeline. This architecture eases the maintainability and replacement of each module and, at the same time, makes parallelization straightforward. The workflow for crawling parallel data from a multilingual website is organized in three main stages that cannot be parallelised; they are:

- **Website downloading**: When a user provides a URL, Bitextor completely downloads it by using the free/open-source tool httrack,[4] which is able to filter the content of a website to download exclusively text content (plain text, XML, and HTML, mainly). Only when the website is completely downloaded it is possible to start the processing.
- **Documents processing**: All the documents downloaded are processed in order to detect target content: useful text content in the languages of interest. The normalisation consists of the following sub-tasks:
    - *Normalisation and cleaning*: Any errors in the HTML structure of the files are fixed

[1] http://www.ua.es
[2] http://www.prompsit.com
[3] http://www.sourceforge.net/projects/bitextor
[4] http://www.httrack.com/

and UTF-8 encoded valid XHTML is obtained, by using the tool Apache Tika 1.5.[5] On the normalised documents, the tool boilerpipe is applied to clean up boilerplates: useless content, such as menus, advertisements, etc.

- *Text extraction and language identification*: The plain text is extracted from the XHTML documents and the tool langid.py[6] is used to identify the language. At this stage, only documents written in the language specified by the user are kept.
- *XHTML structure fingerprint extracted*: one of the heuristics used by Bitextor to identify parallel documents is to compare the XHTML *fingerprint* (the XML markup structure) of the documents. A representation of this structure is extracted at this stage for further comparison.

- **Parallel document identification**: At this stage all the useful information has already been extracted from the documents and the pair-identification process can be carried out. Bitextor mainly uses two heuristics for detecting parallel documents. On one hand, a bilingual lexicon is used to preliminary identify the most suitable candidates to be the translation of each document by applying a variant of the *tf/idf* algorithm. Then, on this set of candidates, a deeper comparison of the XHTML structure is performed to confirm which are the most likely pairs of documents. The sub-tasks of this stage are:
  - *Building a document index*: An index is built so, for each document, the list of the words from the lexicon that appear in it are identified.
  - *Obtaining list of suitable candidates*: For each document, a list of the most similar documents in the other language is obtained based on the number of words connected through the bilingual lexicon.
  - *Re-ranking of candidates*: The list of candidates for each document is re-ranked by comparing the XHTML structure by using an edit distance algorithm. This process is independent of the content and, therefore, is a more reliable method, although more expensive.
  - *Document pairs extraction*: The list of candidates for the documents in both languages are compared. Only those pairs of documents which are in their mutual candidates lists are aligned. It is possible to change the size of these candidates lists: long lists produce more pairs of documents (higher recall) while short lists of candidates produce more reliable alignments (higher precision).
- **Output of the system**: Once the parallel documents are identified, it is possible to produce different kinds of output. Appart of the list of pairs of documents they can also be sentence-aligned by using the Hunalign[7] tool; this second option can produce tab-separated raw text or a translation memory in TMX:
  - *Alignment of documents*: It is possible to score the pairs of documents by using Hunalign, which provides a general score based on the tentative alignment of the segments in each document.
  - *Sentence aligned in raw text*: the tool Ulysses[8] is used for sentence splitting, and Hunalign for sentence-alignment between the detected pairs of documents. The output of this process is a tab-sepparated list of pairs of documents, together with their score (provided by Hunalign) and the URLs from which the segments come from. An additional cleaning can be done on this output which removes pairs of

[5] http://tika.apache.org/1.5/index.html

[6] https://github.com/saffsd/langid.py

[7] http://mokk.bme.hu/resources/hunalign/

[8] https://github.com/sortiz/ulysses-sentence-splitter/compare

segments under a given score, and pairs of documents with a given number of unaligned segments.

- *TMX output*: The output of the previous process can easily be converted into TMX format, a widely used format for translation memories based on XML.

3.2 Installing Bitextor

Bitextor can be downloaded from `http://sourceforge.net/projects/bitextor/files/bitextor/`. Several versions of the tool are available in this website, as well as a collection of pre-trained bilingual lexicons. Once one downloads the tool, the first step is to uncompress it:

```
[user@machine:~]$ user@pc:~$ tar -xvzf bitextor-4.1.0.tgz
```

To install Bitextor it is first necessary to run the script 'configure', which identifies the location of the external tools used. Then the code is compiled and installed by means of the command 'make':

```
[user@machine:~]$ user@pc:~$ ./configure
[user@machine:~]$ user@pc:~$ make
[user@machine:~]$ user@pc:~$ sudo make install
```

In case one does not have *sudo* privileges, it is possible to install the tool locally by specifying a different installation directory when running the script 'configure':

```
[user@machine:~]$ ./configure --prefix=LOCALDIR
[user@machine:~]$ make
[user@machine:~]$ make install
```

where LOCALDIR can be any directory where the user has writing permission, such as `~/local`. In both examples, HTTrack is a requirement and an error will be prompted to the user if this tool is not installed when running configure. If one does not want to use this tool it is possible to run configure with the option --without-httrack.

Some tools and libraries must be installed before proceeding with the installation of Bitextor. Autotools are needed to configure and install the tool, as well as javac and jar for packaging the Java libraries distributed, and a C++ compiler to compile the tools Hunalign and GIZA++ included in this package. Java and Python 2.7 are necessary to run the scripts and libraries used by Bitextor. In addition to these general tools some python modules are necessary: langid, python-Levenshtein, NLTK, and regex. These libraries can be easily installed by using the tool pip:

```
[user@machine:~]$ sudo pip install langid python-Levenshtein NLTK regex
```

3.3 Crawling parallel data with Bitextor

There are three ways to call Bitextor. Two of them include the first step (downloading the websites) and are:

```
[user@machine:~]$ bitextor [OPTIONS] -u  URL -v LEXICON LANG1 LANG2
```

```
[user@machine:~]$ bitextor [OPTIONS] -f FILE -v LEXICON LANG1 LANG2
```

In the first case, Bitextor downloads the URL specified. In the second case, the file specified with the option -f should contain a list of URLs, which are processed separately. One more way to run Bitextor is available, using option -d to specify a directory containing a crawled website (this step starts in the second step described in the previous section):

```
[user@machine:~]$ bitextor [OPTIONS] -d DIRECTORY -v LEXICON LANG1 LANG2
```

Options -u and -d can be combined to specify the directory where the website will be downloaded. Several options can be set using the command line options:

- *-f FILE* sets the path to a file containing a list of URLs to be crawled and processed (one per line).
- *-d FILE* sets the path to a directory containing a crawled website.
- *-I DIR* path where all th intermediate files (the output of each script) are stored; if not specified, they are stored in a temporal directory.
- *-L DIR* directory where the logs of each script are stored; this option may be useful for debugging.
- *-b NUM* when this option is enabled, only the first NUM candidates from the RINDEX candidate list are taken into account when computing the bidirectional document alignment.
- *-v FILE* path to the dictionary used by the script bitextor-lettr2idx.
- *-m NUM* if the number of wrong alignments in a pair of documents processed by bitextor-align-segments is higher than NUM, the pair of documents is discarded (5 by default).
- *-q NUM* if the confidence score for a pair of segments aligned by Hunalign in bitextor-align-segments is lower than NUM it is discarded (0 by default).
- *-x* if this option is enabled, the output of Bitextor will be formatted as a standard TMX translation memory (this option adds at the end of the pipeline the script bitextor-buildTMX).
- *-t DIR* choosing a temporal directiory different to /tmp to store temporal files.
- *-O FILE* Setting the output file of the whole process.

3.4 Obtaining bilingual lexicons for Bitextor

Bilingual lexicons are necessary to crawl parallel data with Bitextor. These lexicons must follow the format:

```
LANGUAGE1_CODE              LANGUAGE2_CODE
word1_in_language1          word1_in_language2
word2_in_language1          word2_in_language2
word3_in_language1          word3_in_language2
...                         ...
```

where words can appear more than once. For example, a valid dictionary could be:

```
en          es
car         coche
and         y
letter      carta
...         ...
```

Some lexicons are already available at
https://sourceforge.net/projects/bitextor/files/bitextor/bitextor-4.0/dictionaries/.

However, customised dictionaries can be automatically built from a parallel corpora by using the script `bitextor-builddics`, included in Bitextor. The script uses the tool GIZA++[9] to build probabilistic dictionaries, which are filtered to keep only those pairs of words fitting the following two criteria:

- both words must occur at least 10 times in the corpus; and
- the harmonic mean of translating the word from lang1 to lang2 and from lang2 to lang1 must be equal or higher than 0.2.

To obtain a dictionary, it is only needed to have a parallel corpus in the usual format:

- two files: one containing the segments in lang1, and the other containing the segments in lang2; and
- the segments appearing in the same line in both files are parallel.

For a pair of files FILE1 and FILE2 containing a parallel corpus, the script would be used as follows:

```
user@pc:~$ bitextor-builddics LANG1 LANG2 FILE1 FILE2 OUTPUT.dic
```

with LANG1 and LANG2 being the codes of the two languages to be added in the first line of the lexicon, and OUTPUT.dic being the path to the file which will contain the resulting dictionary.

3.5 Examples

To ilustrate the usage of the tool, an example will be shown on the web site of the Wolrd Health Organisation related to tratments of ebola: http://www.who.int/mediacentre/news/statements/2014/ebola-ethical-review-summary, which is available in Chinese, English, French, Rusian, and Spanish. For any of these pairs of languages, it is necessary to obtain a valid bilingual lexicon. At http://sourceforge.net/projects/bitextor/files/bitextor/bitextor-4.0/dictionaries/, English-Spanish, English-French, and Spanish-French lexicons are available. Once the lexicon is obtained, the tool is run:

```
[user@machine:~]$ bitextor -u
http://www.who.int/mediacentre/news/statements/2014/ebola-ethical-review-
summary/ -b 1 -v ~/en-es.dic en es
```

In the example, the option for setting the size of candidates lists (-b) is set to 1, which means that only mutual best parallel-document candidates are aligned. This is the highest level of quality that can be obtained in document alignments with Bitextor. The output of this command whould look like this:

```
en/index.html  es/index.html  Ethical considerations for use of unregistered interventions for Ebola
virus disease (EVD)    Consideraciones éticas sobre el uso de intervenciones no registradas en la
enfermedad por el virus del Ebola (EVE)
en/index.html  es/index.html  Summary of the panel discussion      Resumen de la mesa redonda
```

[9] http://code.google.com/p/giza-pp, http://code.google.com/p/giza-pp

...

If option -x is set, a TMX translation memory would be obtained:

```
<?xml version="1.0"?>
<tmx version="1.4">
 <header adminlang="ca" srclang="en" o-tmf="PlainText" creationtool="bitextor"
creationtoolversion="4.0" datatype="PlainText" segtype="sentence" creationdate="20140907T124619"  o-
encoding="utf-8"></header>
 <body>
   <tu tuid="1" datatype="Text">
    <tuv xml:lang="en">
     <prop type="source-document">en/index.html</prop>
     <seg>Ethical considerations for use of unregistered interventions for Ebola virus disease
(EVD)</seg>
    </tuv>
    <tuv xml:lang="es">
     <prop type="source-document">es/index.html</prop>
     <seg>Consideraciones éticas sobre el uso de intervenciones no registradas en la enfermedad por
el virus del Ebola (EVE)</seg>
    </tuv>
   </tu>
   <tu tuid="2" datatype="Text">
    <tuv xml:lang="en">
     <prop type="source-document">en/index.html</prop>
     <seg>Summary of the panel discussion</seg>
    </tuv>
    <tuv xml:lang="es">
     <prop type="source-document">es/index.html</prop>
     <seg>Resumen de la mesa redonda</seg>
    </tuv>
   </tu>
...
```

If it was necessary to crawl English-Rusian parallel text from this website, a custom lexicon would be needed. For this task, one could download a parallel corpus in these two languages, for example the United Nations corpus available at `http://opus.lingfil.uu.se/download.php?f=UN/en-ru.txt.zip` which is already in the desired format. Once the zip package is uncompressed, one would call the bitextor-builddics script as follows:

```
[user@machine:~]$ bitextor-builddics en ru UN.en-ru.en UN.en-ru.ru ~/en-ru.dic
```

and the dictionary would be created in the home directory of the user.

4 ILSP Focused Crawler

ILSP Focused Crawler (ILSP-FC) is a research prototype for acquiring domain-specific monolingual and bilingual corpora. In general, the crawler initializes its frontier (i.e. the list of pages to be visited) from a seed URL list provided by the user, classifies fetched pages as appropriate for the user's aims (i.e. in the targeted language and/or relevant to the targeted domain), extracts links from fetched web pages, adds them to the list of pages to be visited and repeats this process until some threshold set by the user is reached. ILSP-FC integrates modules for text normalization, language identification, document clean-up, text classification and identification of bitexts (documents that are translations of each other). If the user does not provide a list of terms, the software can be used as a general crawler.

ILSP-FC is being developed by researchers of ILSP/Athena RIC in the framework of European projects like Abu-MaTran and QTLaunchPad project, a European Commission-funded collaborative research initiative dedicated to overcoming quality barriers in machine and human translation and in language technologies. An initial version of the crawler was produced during PANACEA, an EU FP7 project for the acquisition and production of Language Resources.

ILSP-FC is a Java project released under the GNU GPL, v. 3.0 license. It depends on open-source libraries for web mining and building data-processing workflows. The latest version, ILSP-FC 2.2.1 was released on 2014-09-09.

A recent scientific paper describing ILSP-FC in detail is Papavassiliou et al (2013).

4.1 ILSP-FC Architecture

ILSP-FC is a modular and open-source focused crawler for the automatic acquisition of monolingual or bilingual, domain-specific or general corpora from the Web, depending on its configuration. In order to ensure modularity and scalability, the crawler is built using Bixo[10] , an open-source web mining toolkit that allows easy configuration of workflows and runs on top of the Hadoop[11] framework for distributed data processing. The workflow of the ILSP-FC is illustrated in Figure 1, followed by a short description of its main modules:

[10] http://openbixo.org/

[11] http://hadoop.apache.org

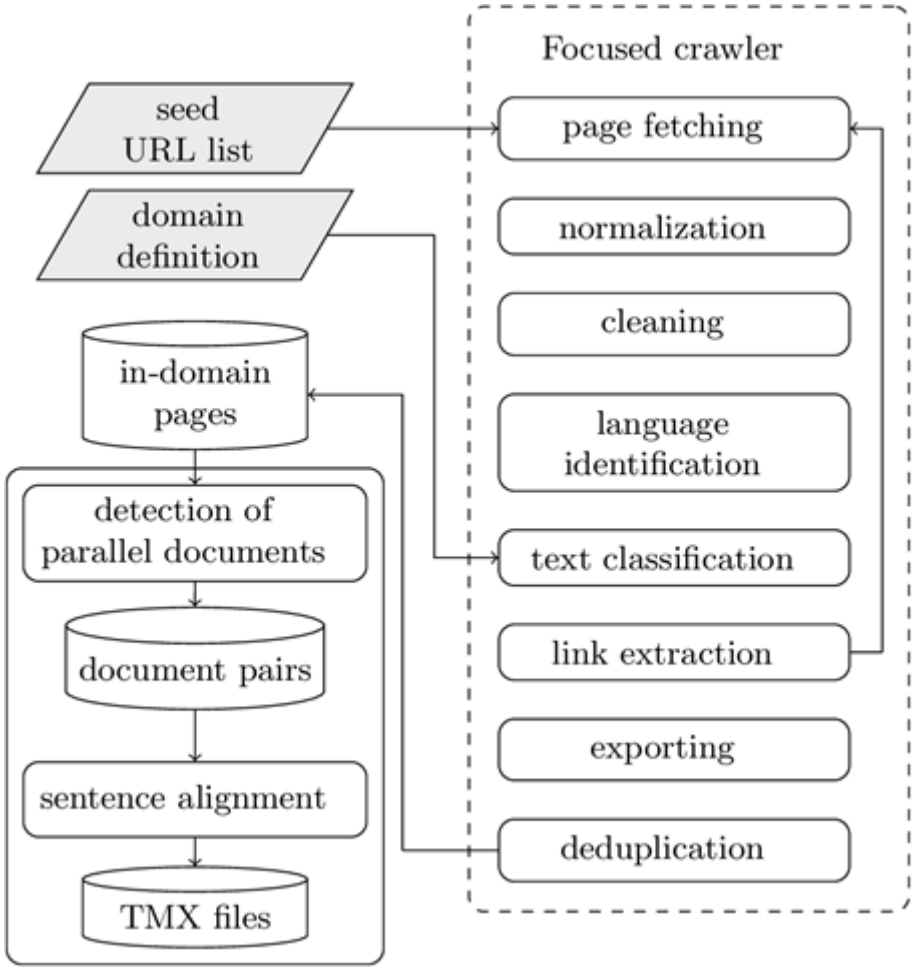

Figure 1. ILSP-FC architecture

- **Page Fetcher** concerns visiting and downloading web pages. A multithreaded crawling implementation has been adopted in order to ensure concurrent visiting of multiple hosts. Users can configure several settings that determine the fetching process, including the number of concurrent harvesters and specific document types that are to be filtered out during crawling.
- **Normalizer** detects the text encoding of the downloaded web pages and if needed, the content is converted into UTF-8. It also takes special care for the normalization of specific characters like no break space, narrow no-break space, three-per-em space, etc. In addition, the module uses the Apache Tika toolkit[12] to parse the structure of each fetched web page and extract its metadata (e.g. title, description, keywords, publisher, author, license etc.).
- **Cleaner** deals with the detection and appropriate annotation of the boilerplate (i.e. the "noisy" content such as navigation headers, advertisements, disclaimers, etc) of a web page. It uses a modified version of Boilerpipe[13] , which also segments the main text in paragraphs and extracts structural information like *title*, *heading* and *list item*.
- **Language identifier** detects the language of the main textual content of a web page. In the context of the Abu-MaTran project, a "factory" of four alternative language identifiers

[12] http://tika.apache.org
[13] http://code.google.com/p/boilerpipe/

(LangID[14] , Cybozu[15], Tika[16] and a specific one (Jorg Tiedemann and Nikola Ljubesic, 2012) for discriminating among Serbian, Croatian, and Bosnian) has been integrated into the system. If a web page is not in the targeted language, its only further use is in the extraction of new links. Even in cases when the main content of a web page is in the targeted language, it is likely that the web page includes a few paragraphs that are not in this language. Thus, language identification is also applied at paragraph level as well and, if needed, appropriate annotations are added.

- **Text Classifier** is included in the process workflow in cases of using the ILSP-FC for the construction of domain-specific collections. The text classification method[17] compares the content of the page to a user-provided domain definition[18] and based on the number of terms' occurrences, their location in the web page and the weights of found terms, a page relevance score is calculated. This score and the number of unique terms found, are compared with predefined thresholds and if the values are higher than the thresholds, the web page is categorized as relevant to the domain and stored. It is worth mentioning that the user can affect the strictness of the classifier by setting the values of thresholds in the crawler's configuration file.
- **Link Extractor** determines the crawling strategy. This module examines the anchor and the surrounding texts of the extracted links and calculates a score for each link. As in the previous module, the link score is influenced by the found terms and/or special patterns which denote that a link probably points to a candidate translation of the page it was originated from. Then, the list of extracted links that are scheduled to be visited is sorted and the most promising links (i.e. links that point to "in-domain" web pages and/or candidate translations of the already visited web pages) are selected.
- **Exporter** generates an XML file for each stored web page. Each file contains metadata (e.g. language, domain, URL, title, description, publisher, author, availability/license, etc) of the corresponding web document inside a *header* element. In addition, a *body* element includes the textual content segmented into paragraphs and the attributes of each paragraph (i.e. title, heading, boilerplate, an indication if the paragraph is not in the targeted language, etc).
- **Deduplicator** deals with the identification of (near) duplicate documents. To this end, a naive language dependent stemmer is applied on the paragraphs of the main content of each document. Then each document is represented by a list containing the MD5 hashes of the paragraphs, and each document list is checked against all other document lists. For each candidate pair, the intersection of the lists is found and if the intersection (in terms of words) is over a predefined threshold the documents are considered duplicates and the shortest is discarded.
- **Pair Detector** aims to identify documents that could be considered parallel. To this end, this module uses three alternative criteria based on special patterns included in the URLs of the acquired web pages, cooccurrences of images with the same filename, and structure similarity. Therefore, it does not use any knowledge of the targeted languages but applies language independent methods for pair detection.
- **Sentence Aligner** extracts sentence alignments (i.e. pairs of candidate parallel sentences). In

[14] https://github.com/saffsd/langid.py

[15] http://code.google.com/p/language-detection/

[16] http://tika.apache.org/1.2/api/org/apache/tika/language/LanguageIdentifier.html

[17] The method is integrated in the Combine crawler (http://combine.it.lth.se/)

[18] A list of term triplets (<relevance weight, (multi-word) term, subdomain>) that describe a domain and, optionally, subcategories of this domain

the context of Abu-MaTran this module was added to the process workflow in order to provide resources that are appropriate for training/testing statistical machine translation systems. To this end, a rule-based sentence splitter and the Hunalign tool have been integrated in ILSP-FC.

4.2 Installation and Development Setup

Depending on your requirements and system setup, there are several ways to get ILSP-FC from its site, http://nlp.ilsp.gr/redmine/projects/ilsp-fc/.

If you just want to run ILSP-FC directly from the command line, you can download the latest runnable jar-with-dependencies jar file provided at the Files section of the ILSP-FC site[19]. Then you can proceed with the usage instructions provided below.

If you want to use ILSP-FC in your Java project, you can integrate it in several ways.

The suggested way is by using Maven and Eclipse. We assume a recent Eclipse installation with Maven Integration (m2e) installed. The following were tested in Eclipse Juno with m2e 1.1

- Download the latest ilsp-fc*project.zip archive from the Files section of the ILSP-FC site.
- In Eclipse, go to menu File > Import->General->Existing Projects Into Workspace.
- Choose Next.
- Click Select archive file.
- Browse to where you saved the ilsp-fc*project.zip archive.
- Press Finish.
- The project will now be imported and all dependencies will be downloaded in your system.

An alternative way of building ILSP-FC using Maven only is the following:

- Make sure a recent version of Maven (3.*) is installed on your machine
- Download the latest ilsp-fc*project.zip archive from the Files section of the ILSP-FC site and create a runnable jar-with-dependencies jar file, e.g.

```
[user@machine:~/src ]$ wget http://nlp.ilsp.gr/.../ilsp-fc-2.2-project.zip
[user@machine:~/src ]$ unzip ilsp-fc-2.2-project.zip
[user@machine:~/src ]$ cd ilsp-fc-2.2
[user@machine:~/src/ilsp-fc-2.2 ]$ mvn clean install
```

- A runnable jar has now been created. You can test it by running, e.g.,

```
[user@machine:~/src/ilsp-fc-2.2 ]$ java -jar ./target/ilsp-fc-2.2-jar-with-dependencies.jar --help
```

If you do not want to use Maven, you can use the latest source-with-dependencies zip file provided from the Files section of the ILSP-FC site. This archive contains the source for ILSP-FC and jars for all third party libraries. You can add these jars to the Java build path in your favorite IDE.

[19] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/files

4.3 Sentence Alignment Setup

For the generation of sentence alignments from bilingual crawls, ILSP-FC integrates the java sentence aligner provided at http://align.sourceforge.net/. Alternatively you can use an external aligner like Hunalign. For example, for the current version of ILSP-FC, you can

- download the hunalign-1.2 source code from http://mokk.bme.hu/en/resources/hunalign/
- follow the instructions on the Hunalign page for building Hunalign
- put the Hunalign directory containing the Hunalign executable next to the runnable ilsp-fc jar.

For example, if you run ilsp-fc from:

```
~/ilsp-fc/ilsp-fc-2.2-jar-with-dependencies.jar
```

you should do the following

```
cd ~/ilsp-fc/
wget ftp://ftp.mokk.bme.hu/Hunglish/src/hunalign/latest/hunalign-1.2.tgz
tar xvfz hunalign-1.2.tgz
cd hunalign-1.2/src/hunalign/
make
ln -sf hunalign-1.2 hunalign
```

This should create `hunalign/src/hunalign/hunalign` with the suggested Hunalign directory structure, including

```
~/ilsp-fc/hunalign/data/
~/ilsp-fc/hunalign/src/hunalign/hunalign
```

Now, you are ready to produce TMX files from bilingual crawled data using the -align, -dict, -oft and -ofth options described in the Usage part of the documentation.

4.4 Usage

Once you have an ilsp-fc runnable jar using one of the ways described above, you can run it like this:

```
java -jar ilsp-fc-X.Y.Z-jar-with-dependencies.jar
```

4.4.1 Input

In case of general monolingual crawls the required input from the user is:

- a list of seed URLs (i.e. a text file with one URL per text line).

In case of focused monolingual crawls (i.e. when the crawler visits/processes/stores web pages that are related to a targeted domain), the input should include:

- a list of seed URLs pointing to relevant web pages. An example seed URL list for *Environment* in English can be found at ENV_EN_seeds.txt[20].
- a list of term triplets (<*relevance,term,subtopic*>) that describe a domain (i.e. this list is required in case the user aims to acquire domain-specific documents) and, optionally, subcategories of this domain. An example domain definition can be found at

[20] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/ENV_EN_seedstxt

ENV_EN_topic.txt[21] for the *Environment* domain in English. Details on how to construct/bootstrap such lists and how they are used in text to topic classification could be found at Papavassiliou et al. (2013).

In case of general bilingual crawling, the input from the user includes:

- a seed URL list which should contain URL from only one web domain (e.g. ENV_EN_ES_seed.txt[22]). The crawler will follow only links pointing to pages inside this web domain. However, the user could use the `filter` parameter (see below) to allow visiting only links pointing to pages either inside versions of the top domain of the URL (e.g. http://www.fifa.com/, http://es.fifa.com/ , etc.) or in different web domains (i.e. in cases the translations are in two web domains e.g. http://www.nrcan.gc.ca and http://www.rncan.gc.ca). Examples of seed URLs can be found at this seed_examples.txt[23].

In case of focused bilingual crawls, the input should also include:

- a list of term triplets (<*relevance,term,subtopic*>) that describe a domain (i.e. this list is required in case the user aims to acquire domain-specific documents) and, optionally, subcategories of this domain in both the targeted languages (i.e. the union of the domain definition in each language). An example domain definition of *Environment* for the English-Spanish pair can be found at ENV_EN_ES_topic.txt[24]

4.4.2 Language support

For both monolingual and bilingual crawling, the set of currently supported languages comprises Croatian, English, French, German, Greek, Italian, Japanese, Portuguese, and Spanish.

In order to add another language, a developer/user should:

- verify that the targeted language is supported by the default language identifier (https://code.google.com/p/language-detection/) integrated in the ILSP-FC,
- add a textline with common abbreviations for this language in the langKeys.txt file included in the ilsp-fc runnable jar, and
- add a proper analyser in the `gr.ilsp.fmc.utils.AnalyserFactory` class of the ilsp-fc source.

4.4.3 Other settings

There are several settings that influence the crawling process and can be defined in a configuration file before the crawling process. The default configuration files for monolingual and bilingual crawls are FMC_config.xml[25] and FBC_config.xml[26] respectively. They are included in the ilsp-fc runnable jar.

Some of the settings can also be overriden using options of the ilsp-fc runnable jar, as follows:

```
crawlandexport : Forces the crawler to crawl and export the results.
-a : user agent name (required)
```

[21] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/ENV_EN_topictxt
[22] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/ENV_EN_ES_seedtxt
[23] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/Seed_examplestxt
[24] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/ENV_EN_ES_topictxt
[25] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/FMC_configxml
[26] http://nlp.ilsp.gr/redmine/projects/ilsp-fc/wiki/FBC_configxml

```
-type : the type of crawling. Crawling for monolingual (m) or parallel
(p).
-cfg : the configuration file that will be used instead of the default
(see crawler_config.xml above).
-c : the crawl duration in minutes. Since the crawler runs in cycles
(during which links stored at the top of the crawler's frontier are
extracted and new links are examined) it is very likely that the defined
time will expire during a cycle run. Then, the crawler will stop only
after the end of the running cycle. The default value is 10 minutes.
-n : the crawl duration in cycles. The default is 1. It is proposed to use
this parameter for testing purposes.
-t : the number of threads that will be used to fetch web pages in
parallel. The default value is 10.
-f : Forces the crawler to start a new job (required).
-lang : the targeted language in case of monolingual crawling (required).
-l1 : the first targeted language in case of bilingual crawling
(required).
-l2 : the second targeted language in case of bilingual crawling
(required).
-u : the text file that contains the seed URLs that will initialize the
crawler. In case of bilingual crawling the list should contain only 1 or 2
URLs from the same web domain.
-tc : domain definition (a text file that contains a list of term triplets
that describe the targeted domain). If omitted, the crawl will be a
"general" one (i.e. the module for text-to-domain classification will not
be used).
-k : Forces the crawler to annotate boilerplate content in parsed text.
-filter : A regular expression to filter out URLs which do NOT match this
regex. The use of this filter forces the crawler to either focus on a
specific web domain (i.e. ".*ec.europa.eu.*"), or on a part of a web
domain (e.g.".*/legislation_summaries/environment.*"). Note that if this
filter is used, only the seed URLs that match this regex will be fetched.
-u_r : This parameter should be used for bilingual crawling when there is
an already known pattern in URLs which implies that one page is the
candidate translation of the other. It includes the two strings to be
replaced separated by ';'.
-d : Forces the crawler to stay in a web domain (i.e. starts from a web
domain and extracts only links to pages inside the same web domain). It
should be used only for monolingual crawling.
-len : Minimum number of tokens per paragraph. If the length (in terms of
tokens) of a paragraph is less than this value (default is 3) the
paragraph will be annotated as "out of interest" and will not be included
into the clean text of the web page.
-mtlen : Minimum number of tokens in a cleaned document. If the length (in
terms of tokens) of the cleaned text is less than this value (default is
200), the document will not be stored.
-align : Name of the aligner to be used for sentence alignment (default is
maligna).
-dict :  A dictionary for sentence alignment if Hunalign is used. The
default L1-L2 dictionary of Hunalign will be used if it exists.
-xslt : Insert a stylesheet for rendering xml results as html.
-oxslt : Export crawl results with the help of an xslt file for better
examination of results.
-dom : Title of the targeted domain (required when domain definition, i.e.
tc parameter, is used).
-dest : The directory where the results (i.e. the crawled data) will be
stored.
-of : A text file containing a list with the exported XML files (see
section Output below).
```

```
-ofh : An HTML file containing a list with the generated XML files (see
section Output below).
-oft : A text file containing a list with the exported TMX files (see
section Output below).
-ofth : An HTML file containing a list with the generated TMX files (see
section Output below).
```

4.4.4 Example monolingual crawls

Run a focused crawl for English, using a list of terms for the Environment domain:

```
java -jar ilsp-fc-X.Y.Z-jar-with-dependencies.jar crawlandexport \
      -a user@xyz.gr \
      -type m -c 10 -lang en -of output_test1_list.txt \
      -ofh output_test1_list.txt.html -tc ENV_EN_topic.txt  \
      -u ENV_EN_seeds.txt -f -k -dom Environment
```

Run a crawl for Spanish with no topic:

```
java -jar ilsp-fc-X.Y.Z-jar-with-dependencies.jar crawlandexport \
      -a test2 \
      -f -k -type m -c 5 -lang es -of output_test2_list.txt \
      -ofh output_test2_list.txt.html -u seed_examples.txt
```

4.4.5 Example bilingual crawls

Run a crawl for English-French using maligna for sentence alignment. Note that the filter parameter is used since the candidate translations are in different web domains.

```
java -jar ilsp-fc-X.Y.Z-jar-with-dependencies.jar crawlandexport -f -a
abumatran \
-type p -align maligna -l1 en -l2 fr -u seed_examples.txt \
-filter ".*(nrcan|rncan).*" -n 2 -xslt -oxslt -of output_demo_EN-FR.txt \
-ofh output_demo_EN-FR.txt.html \
       -oft output_demo_EN-FR.tmx.txt -ofth output_demo_EN-FR.tmx.html
```

Run a crawl for English-Spanish using Hunalign for sentence alignment.

```
java -jar ilsp-fc-X.Y.Z-jar-with-dependencies.jar crawlandexport \
      -a test4 -c 10 -f -k -l1 es -l2 en \
      -type p -u seed_examples.txt -filter ".*uefa.com.*" \
      -len 0 -mtlen 80 -xslt -oxslt -dest "/var/crawl_results/" \
      -of test_U_ES-EN_output.txt -ofh test_U_ES-EN_output.txt.html \
      -oft test_U_ES-EN_output.tmx.txt
      -ofth test_U_ES-EN_output.tmx.html \
       -align hunalign -dict
```

4.4.6 Output

The output of the ilsp-fc in the case of a monolingual crawl consists of:

- a list of links pointing to XML files following the cesDOC Corpus Encoding Standard (http://www.xces.org/). See this cesDoc[27] file for an example in English for the *Environment* domain.
- a list of links pointing to HTML files (by XSL transformation of each XML) for easier browsing of the collection. As an example, see this rendered cesDoc[28] file.

The output of the ilsp-fc in the case of a bilingual crawl consists of:

- a list of links to XML files following the cesAlign Corpus Encoding Standard for linking cesDoc documents. This example cesAlign file[29] serves as a link between a detected pair of cesDoc documents in English[30] and Spanish[31].
- a list of links pointing to HTML files (by XSL transformation of each cesAlign XML) for easier browsing of the collection. As an example, see this rendered cesAlign file[32].
- a list of links to TMX files containing sentence alignments that have been extracted from the detected document pairs. As an example, see this TMX file[33].
- a list of links pointing to HTML files (by XSL transformation of each TMX) for easier browsing of the collection. As an example, see this rendered TMX file[34].

5 Conclusions

In this deliverable we have described in detail two open-source complementary crawlers, Bitextor and ILSP Focused Crawler, which have been partly developed and adapted within Abu-MaTran, to support the project's data acquisition needs.

In the context of the project's WP3, we will collect bilingual document and sentence-aligned corpora using both crawlers for the languages and domains of the project. The quality of the corpora and the complementarity of the two crawlers will be examined in the context of several statistical machine translation experiments.

[27] http://nlp.ilsp.gr/xslt/ilsp-fc/1.xml
[28] http://nlp.ilsp.gr/xslt/ilsp-fc/1.xml.html
[29] http://nlp.ilsp.gr/xslt/ilsp-fc/44_98_i.xml
[30] http://nlp.ilsp.gr/xslt/ilsp-fc/98.xml
[31] http://nlp.ilsp.gr/xslt/ilsp-fc/44.xml
[32] http://nlp.ilsp.gr/xslt/ilsp-fc/44_98_i.xml.html
[33] http://nlp.ilsp.gr/xslt/ilsp-fc/44_98_i.tmx
[34] http://nlp.ilsp.gr/xslt/ilsp-fc/44_98_i.html

Bibliography

Tiedemann, J., and Ljubesic, N. (2012). Efficient Discrimination between Closely Related Languages. In *Proceedings of the 24th International Conference on Computational Linguistics* (COLING'12), Mumbai, India

Papavassiliou, V., Prokopidis, P. & Gregor Thurmair. (2013). A modular open-source focused crawler for mining monolingual and bilingual corpora from the web. In *Proceedings of the Sixth Workshop on Building and Using Comparable Corpora*, pages 43-51. Sofia, Bulgaria : Association for Computational Linguistics. http://www.aclweb.org/anthology/W13-2506.pdf