European Commission Seventh Framework Programme
Call: FP7-ICT-2007-1, Activity: ICT-1-4.1
Contract No: 216267

Integrated Prototypes Progress Report V2

Deliverable No: D6.7

Version 1.0

Editor:	EA
Work Package:	WP6
Status:	Final
Date:	M24
Dissemination Level:	PU

Project Overview

Project Name: LiWA – Living Web Archives

Call Identifier: FP7-ICT-2007-1

Activity Code: ICT-1-4.1

Contract No: 216267

Partners:
1. Coordinator: Universität Hannover, L3S Research Center, Germany
2. European Archive Foundation (EA), Netherlands
3. Max-Planck-Institut für Informatik (MPG), Germany
4. Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI), Hungary
5. Stichting Nederlands Instituut voor Beeld en Geluid (BeG), Netherlands
6. Hanzo Archives Limited (HANZO), United Kingdom
7. National Library of the Czech Republic (NLP), CZ
8. Moravian Library (MZK), CZ

Document Control

Title: Integrated Prototypes Progress Report – Version 2

Author/Editor: Radu Pop (EA)

Document History

Version	Date	Author/Editor	Description/Comments
0.1	Jan 12, 2010	Radu Pop	Outline
0.2	Jan 18, 2010	Radu Pop Dimitar Denev	Section 2 updated Section 3.3 updated
0.3	Feb 10, 2010	Andras Benczur	Section 3.2 added
0.4	Feb 12, 2010	Thomas Risse	Section 3.4 added
1.0	Feb 12, 2010	Radu Pop	Final version

Legal Notices

Table of Contents

1. Introduction

This document describes the first version of the integrated prototype for the European Archive platform (as required for D6.6). It presents the integration status for different components developed by the technical work packages in LiWA (Wp2 – Wp5) and focuses on the integration of LiWA components that have been selected for the first version of the prototype.

The test-bed description and building configuration were presented in the previous version of this report (D6.3). This document focuses on on the integration prototypes that become ready in month 23 of the project (see D6.6 and M6.1).

Section 2 represents a “user guide” for the LiWA components, with different details regarding their usability related to the crawler. Section 3 describes the external LiWA modules that shall be used in the post-processing phase, as well as several updates brought to the API set defined for their interaction.

2. Integrated Components

This section describes the list of the LiWA modules that have been selected for the first version of the prototype and that have been already integrated into the European Archive's platform.

They basically correspond to the LiWA components to be used at crawl time, as they were previously classified in D6.1, i.e. internal and external modules. As output from Wp2, we integrated the Rich Media Capturing module and the Link Extractor. From the Wp4 development, we integrated the Temporal Coherence analyser for crawl-time. All these components work in a relatively close relation with the crawler, through dedicated plugins, and use specific external tools, which we shall describe in this section.

We describe in the following a brief user guide for each component, as well as several implementation updates since their first released version.

2.1.Rich Media Capturing Module

The principle of the Rich Media Capturing module is to delegate the multimedia content retrieval to an external application (MPlayer or FLVStreamer) which is able to handle a larger spectrum of transfer protocols, such as real time protocols, widely used for video streaming.

The second version of the technology added a new transfer protocol (RTMP) to the list of real time protocols supported by the Rich Media Capturing module: RTSP, MMS and RTMP. An update of the Heritrix plugin was done accordingly, to be able to detect a broader list of URIs, now including those using RTMP.

As described in more detail in Section 3 of D6.5, the integration of this module entailed joining together 3 main sub-modules:

- a plugin for Heritrix, for detecting the URIs referencing streaming resources,
- a messaging server, for decoupling the Heritrix crawler from the downloading tools,
- an external cluster of downloading tools (using MPlayer and FLVStreamer)

2.1.1.How to activate the plugin in Heritrix

The plugin for Heritrix is written in Java and it represents an implementation of the "deciderules" external interface:

```
    org.archive.crawler.deciderules.ExternalImplInterface
```

The code is available on LiWA's SVN in: `/apps/streaming/src/java/HeritrixStreamingPlugin/` and is automatically compiled under the Cruise Control system in the "HeritrixStreamingPlugin" project.

In order to use the plugin with Heritrix, there are 2 steps to be done:

- modify the configuration file of the crawler: `conf/heritrix.properties`

The property `org.archive.net.UURIFactory.ignored-schemes` should exclude streaming schemes, therefore the RTSP, MMS and RTMP schemes should be removed from the list of ignored schemes. Nevertheless, they should be added to the list of accepted schemes, like for instance:

```
    org.archive.net.UURIFactory.schemes = http,https,dns,rtsp,mms,rtmp
```

- add the plugin to the list of pre-fetch processors:

A new `ExternalImplDecideRule` should be added to the crawl order and the name of the implementation class should be `org.europarchive.StreamingURI`. The decision setting for this deciderule should be REJECT.

2.1.2.How to configure and monitor the messaging server

Based on the standard messaging protocol AMPQ (Advanced Message Queueing Protocol), the messaging system that we use is an open source platform called RabbitMQ (http://rabbitmq.com). The messaging server was installed and properly configured on the test bed machine, using several methods provided by the system's tool for user management (rabbitmqctl). We created a generic “liwa” user for handling the queues of video URLs to be downloaded. An important configuration aspect is represented by the permissions that have to be associated between the user and the virtual host dedicated for external connections:

```
rabbitmqctl add_user liwa liwa
rabbitmqctl set_permissions -p vhost-liwa liwa “.*” “.*” “.*”
```

The messaging server created a virtual host dedicated to the “liwa” user and the messages are organised by specific queues, based on the type of resources to download: `jobs_rtsp, jobs_mms, jobs_rtmp`. The status of the queues can be checked on the test bed, using the system's API, e.g.:

```
rabbitmqctl list_queues -p vhost-liwa
```

The downloading tools connect to the specific queues, download the video files sequentially and pack them into arc files.

2.2.Link Extractor Module

The Link Extractor module works as an external tool for the crawler, providing complementary information related to the links contained by a Web page. It provides a simple method that retrieves the complete list of links discovered in a given Web page and is implemented as a Web service running on Hanzo's platform:

http://www.hanzoarchives.com/liwa/link-extractor/<URL>

Based on the principle of “*page execution*” (as described in Section 3 of D6.5), the benefit of the link extractor is twofold: it guarantees the discovery of *all* the links in a Web page (i.e. by loading and executing the embedded scripts) and, on the other hand, it removes the noise of false speculative links generated by the crawler (e.g. when activating the JS or the “aggressive” HTML extractor).

The mechanism used for integrating the Link Extractor with a running crawler is straight-forward as it is implemented as a plugin for Heritrix, to be activated in the list of extractor processors.

The working *algorithm* of the plugin is the following: for each URL in the seed list (or eventually for a subset of seeds selected by a regular expression) the plugin calls the Web service of the Link Extractor, sending the URL of the page to be analysed. The response of the Link Extractor consists of a list of discovered URIs for the given page. The plugin then injects these new URIs into the crawler's frontier for download and further analysis.

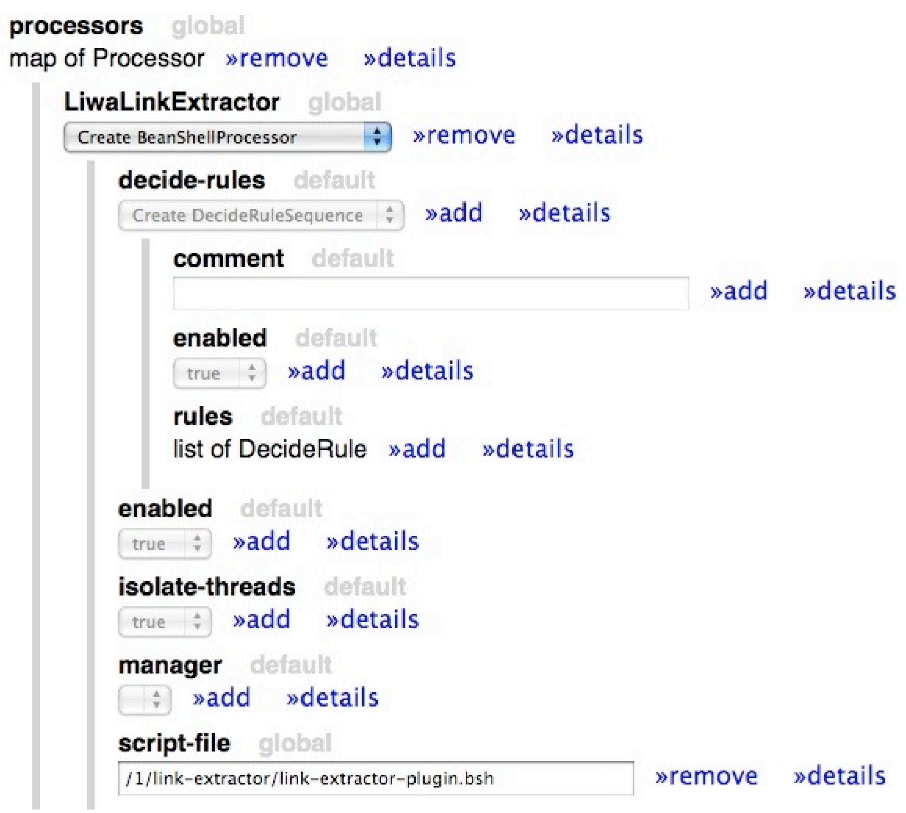

processors global
map of Processor »remove »details
LiwaLinkExtractor global
Create BeanShellProcessor »remove »details
decide-rules default
Create DecideRuleSequence »add »details
comment default
»add »details
enabled default
true »add »details
rules default
list of DecideRule »add »details
enabled default
true »add »details
isolate-threads default
true »add »details
manager default
»add »details
script-file global
/1/link-extractor/link-extractor-plugin.bsh »remove »details

Figure 2.1 The LinkExtractor processor in the map of Heritrix global processors

In order to activate the plugin in Heritrix, the only thing to do is to add a new module to the list of crawl processors, declared in the settings sheet of the crawl job. The global sheet can be edited in the Web interface of Heritrix, as illustrated in Figure 2.1 (map of Processor -> details -> add new element).

The object type to select is the existing processor:

```
org.archive.modules.BeanShellProcessor
```

Then, the only parameter to be set for the BeanShell processor is the name of the script-file on the test bed to be executed:

```
/1/link-extractor/link-extractor-plugin.bsh
```

The plugin implementation uses a BeanShell script that is executed by the Heritrix engine at crawl-time. After an HTTP connection to the link extractor's Web service, the core part of the plugin creates and adds new links to the frontier, using the methods of the `CrawlURI` class in Heritrix:

```
process(CrawlURI curi) {
...
Link getNewLink = new Link (curi.getUURI(),
                      UURIFactory.getInstance(discoveredLinks),
                      LinkContext.NAVLINK_MISC, Hop.NAVLINK);
curi.getOutCandidates().add(curi.createCrawlURI(curi.getBaseURI(),
                    getNewLink, SchedulingConstants.HIGH, false));
...}
```

During the crawl process, the newly discovered links can be observed either by regularly checking the crawl log or by applying some regular expressions on the crawl frontier when the crawl job is paused.

2.3.Crawl-time Temporal Coherence Module

From the perspective of the end user, the usage of the Temporal Coherence module involves 4 components:

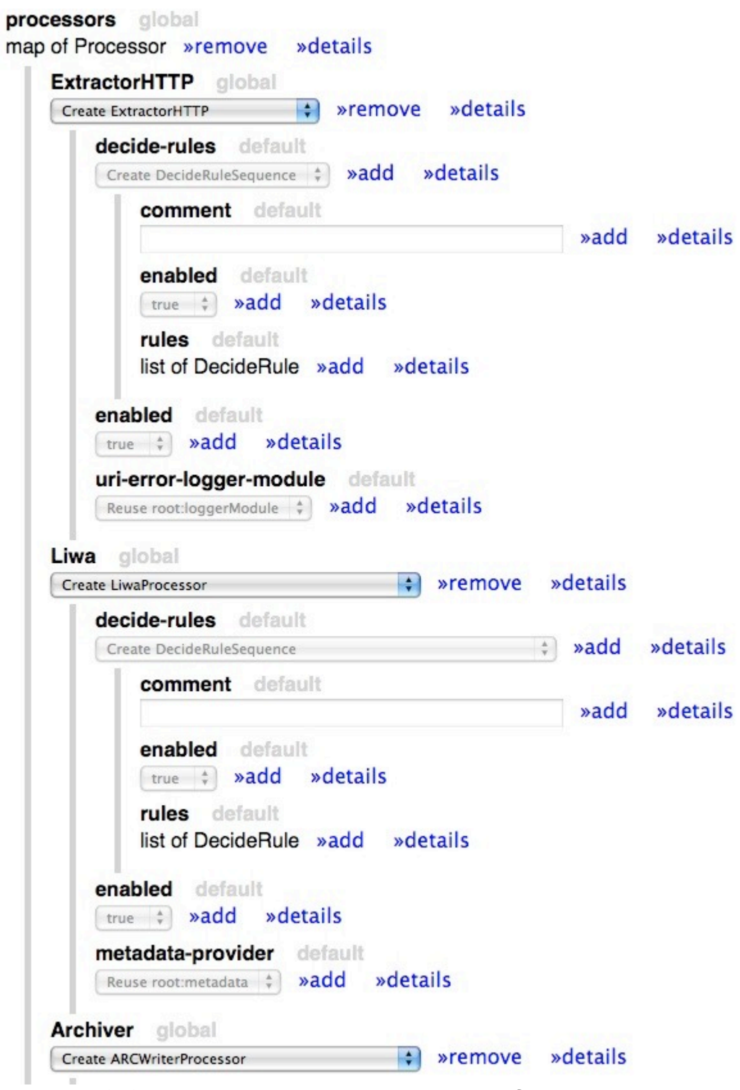
- the plugin for Heritrix, described in more detail in the first version of the integration report (D6.3)
- an Oracle database installed on the test bed for storing the crawl-time information
- the external analyser that connects to the database and builds the crawl graphs
- a graph visualization tool, very useful for generating a graphical representation of the crawl graph, and therefore of the Web site's structure (e.g. yEd Graph Editor – http://www.yworks.com/products/yed/).

A detailed description of the temporal coherence analysis is given in Section 5 of D6.5 and the structure of the database has already been presented in Section 3.8 of D6.1. The temporal coherence plugin is implemented in Java and the source code is available on LiWA's SVN in: `/crawler/coherence/liwa/`. It is automatically compiled under the Cruise Control system in the “TemporalCoherence” project.

2.3.1.How to activate the plugin in Heritrix

In order to activate the plugin for the temporal coherence analysis, there are two parameters to be added in the global sheet of the crawl job. The global sheet can be edited in the Web interface of Heritrix, as illustrated in the following screen shots.

The first parameter is represented by the “LiWAProcessor” itself and is added to the list of global processors for the crawl job, as shown in Figure 2.2. The second parameter represents a filter used by the temporal processor to track down all the URLs discovered for the Web site to be analysed. It implements a specific decide rule, called “MatchesRegExpDecideRule”, in the scope of the crawl. For example, in Figure 2.3, the regular expression of the decide rule accepts all the URLs from the www.irr.org.uk Web site.



processors global
map of Processor »remove »details

- **ExtractorHTTP** global
  Create ExtractorHTTP »remove »details
  - **decide-rules** default
    Create DecideRuleSequence »add »details
    - **comment** default
      »add »details
    - **enabled** default
      true »add »details
    - **rules** default
      list of DecideRule »add »details
  - **enabled** default
    true »add »details
  - **uri-error-logger-module** default
    Reuse root:loggerModule »add »details
- **Liwa** global
  Create LiwaProcessor »remove »details
  - **decide-rules** default
    Create DecideRuleSequence »add »details
    - **comment** default
      »add »details
    - **enabled** default
      true »add »details
    - **rules** default
      list of DecideRule »add »details
  - **enabled** default
    true »add »details
  - **metadata-provider** default
    Reuse root:metadata »add »details
- **Archiver** global
  Create ARCWriterProcessor »remove »details

Figure 2.2 LiwaProcessor in the map of Heritrix global processors

2.3.2. How to launch the temporal analysis

The external analyser is implemented as a stand-alone Java application that can be run locally on the user's desktop. It is bundled as an executable JAR file and can be launched in command line, e.g.:

```
        java  -classpath  /users/liwa/workspace/coherence_exporter.jar
org.liwa.coherence.graph.Main <crawl_id> <storage_path_graphml>
```

The main method takes two arguments: the crawl_id in the database to be analysed and the local folder where the graphML file will be created.

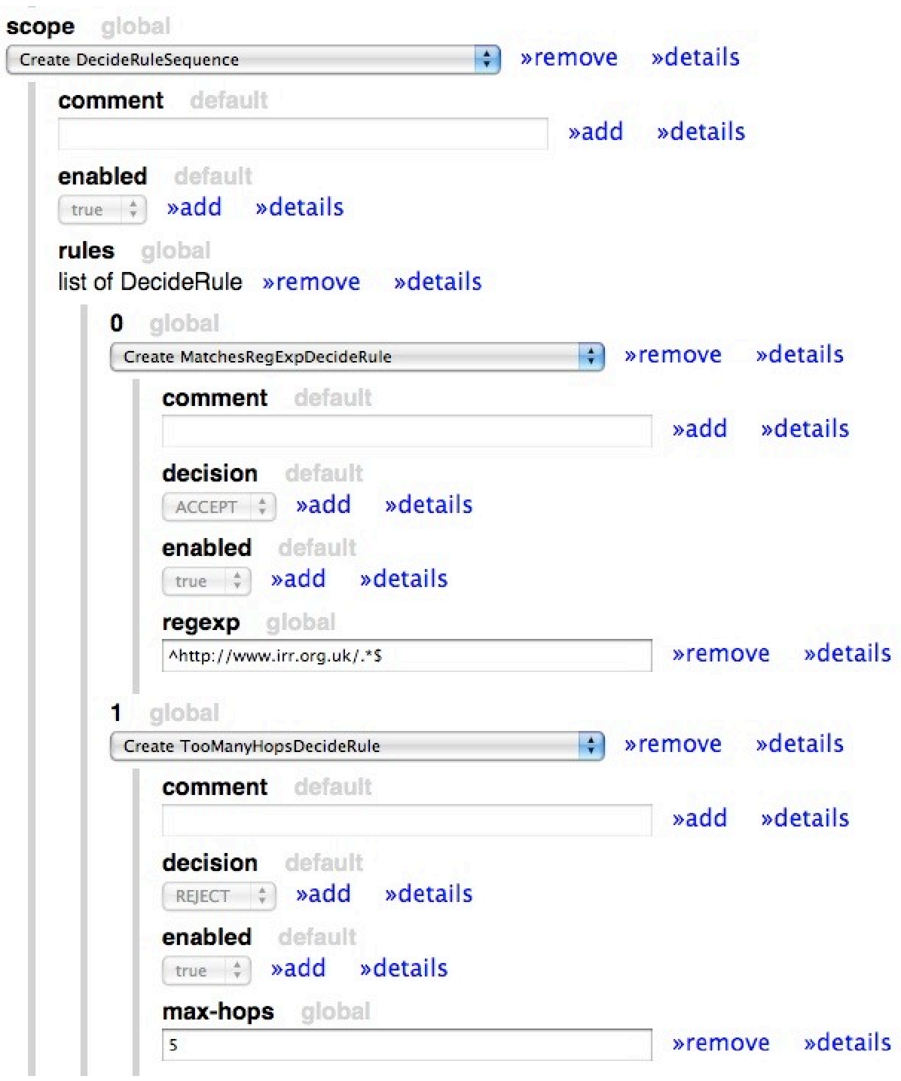

scope global
Create DecideRuleSequence »remove »details
comment default
»add »details
enabled default
true »add »details
rules global
list of DecideRule »remove »details
0 global
Create MatchesRegExpDecideRule »remove »details
comment default
»add »details
decision default
ACCEPT »add »details
enabled default
true »add »details
regexp global
^http://www.irr.org.uk/.*$ »remove »details
1 global
Create TooManyHopsDecideRule »remove »details
comment default
»add »details
decision default
REJECT »add »details
enabled default
true »add »details
max-hops global
5 »remove »details

Figure 2.3 Regular expression to match all the URLs of the Web site

The plugin is already activated on the Heritrix instance running on the test bed. For each new job launched on the test bed, a new crawl_id is incrementally generated in the database in the T_CRAWLS table. For the current stage of our tests, one has to manually check the crawl_id in the database (http://debug.europarchive.org:8080/apex) prior to performing the graph analysis.

The graph builder connects directly to the test bed to retrieve the information related to a given crawl. After the XML file is generated, it can be easily visualized by opening it with the yEd Graph Editor.

3. Post-processing Framework and Integration API

The integration of the post-processing modules is based on the API defined in D6.1.

As described in D6.1, the framework installed on the test-bed provides a post-processing manager (i.e. a set of crontab controlled scripts) that coordinates the 3 external modules in LiWA (Spam Filter, Off-line Coherence Analyser and Semantics Analyser), as well as the crawl storage module

An overview of the proposed API is presented in the following.

3.1.Storage Module

The storage module provides the basic methods to organize the collections and to access the archived files.

The abstract view on the collections and crawls is maintained by a PostgreSQL database installed on the test-bed machine, whereas the access methods are implemented through simple Web services deployed on a running Apache Web server.

http://debug.europarchive.org/apache2-default/

The implementation of the storage methods consists in several Python modules, used to access the WARC files in the local storage space (`/1/storage/liwa-collections`).

There are 2 types of operations provided by the storage module:

- operations dedicated to the collections management – providing information on the relations between collections, crawls and WARC files (see D6.1, Section 3.1)
- access operations to WARC files – based on the “WARC Tools Project”, with extensions from EA “arclib” tools for storage management

For the first category, the API proposed in D6.1 are implemented by the Storage module in the following Web services:

- List <crawl_id> = get_crawls (collection_id)

  Returns all crawls related to a given collection, e.g.:

  http://debug.europarchive.org/storage/main/get_crawls/<collection_id>
- collection_id = get_collection (crawl_id)

  Retrieves the collection to which a crawl belongs, e.g.:

  http://debug.europarchive.org/storage/main/get_collection/<crawl_id>
- List <warc_file> = get_archive_files (crawl_id)

  Returns the list of WARC files belonging to a given crawl, e.g.:

  http://debug.europarchive.org/storage/main/get_archive_files/<crawl_id>

The second category of operations gives direct access to the WARC files for file transfer, or is able to directly extract a specific record in an WARC file given its correct offset:

- get_arc (arc_file)

  Downloads an ARC file from the local storage, e.g.:

  http://debug.europarchive.org/storage/main/get_arc//1/storage/liwa-collections/
EA-TNA1108.www.cesg.gov.uk-20091203203837-00025.arc.gz

- get_record (arc_file, offset)

  Retrieves the resource file stored at a given offset in the ARC file, e.g.:

  http://debug.europarchive.org/arcfiles/get/
EA-TNA-1008-oct-20090107183223-00000.arc.gz/27411

In order to maintain an abstract view on the set of WARC files in the archive, the storage module uses a PostgreSQL database with a relatively simple structure, containing 3 tables:

T_COLLECTIONS:	`[collection_id, collection_name, start_date, end_date, frequency, organization, operator]`
T_CRAWLS:	`[crawl_id, collection_id, date, status]`
T_WARCS:	`[file_name, path, date, crawl_id]`

The WARC files are stored locally on the test-bed machine, using a flat structure, since all the files generated by the crawler receive a unique filename (containing also the creation timestamp).

A crontab controlled script regularly checks for new files generated by different crawl jobs and transfers them to the storage folder.

3.2.Spam Filter Analyser

The main objectives of the Spam Filter development were shaped by the following events.

- In September 2008 the joint LiWA-IIPC meeting has agreed on a plan of using the spam filter module for domain crawling. The efficiency of feature generation (Section 3.2.1) was adjusted for the needs of large domain crawls. Crawl-time filtering (Section 3.2.3) is also implemented accordingly.
- A research challenge "Quality in Web Archives" will be held at the ECML/PKDD 2010 Discovery Challenge. This event will be the first real user of the LiWA Spam feature generation codes as well as the LiWA Spam Assessment manual labeling interface (Section 3.2.2).
- The lessons we have learned from our successful KDD Cup 2009 participation also helped in identifying a small, efficiently computable and accurate spam feature set (Section 3.2.1).

3.2.1.Batch feature generation and classification

In the LiWA Spam Classifier Ensemble we apply state-of-the-art classification techniques by the lessons learned from KDD Cup 2009. Key in our performance is ensemble classification applied both over different feature subsets as well as over different classifiers over the same features. We also apply classifiers yet unexplored against Web spam, including Random Forest and LogitBoost.

Of key importance to the efficiency, in particular for the crawl-time filtering procedure (see Section 3.2.3), is that we compile a small yet very efficient feature set that, on the site level, can be computed by considering only sample pages from the site and completely ignore linkage. By this feature set a filter may very quickly react to a recently discovered site and intercept in time before the crawler would start to follow a large number of pages from a link farm. This feature set itself reaches AUC 0.893 over WEBSPAM-UK2007.

Last but not least we gain strong improvements over the Web Spam Challenge best performance. Our best result in terms of AUC reaches over 0.9.

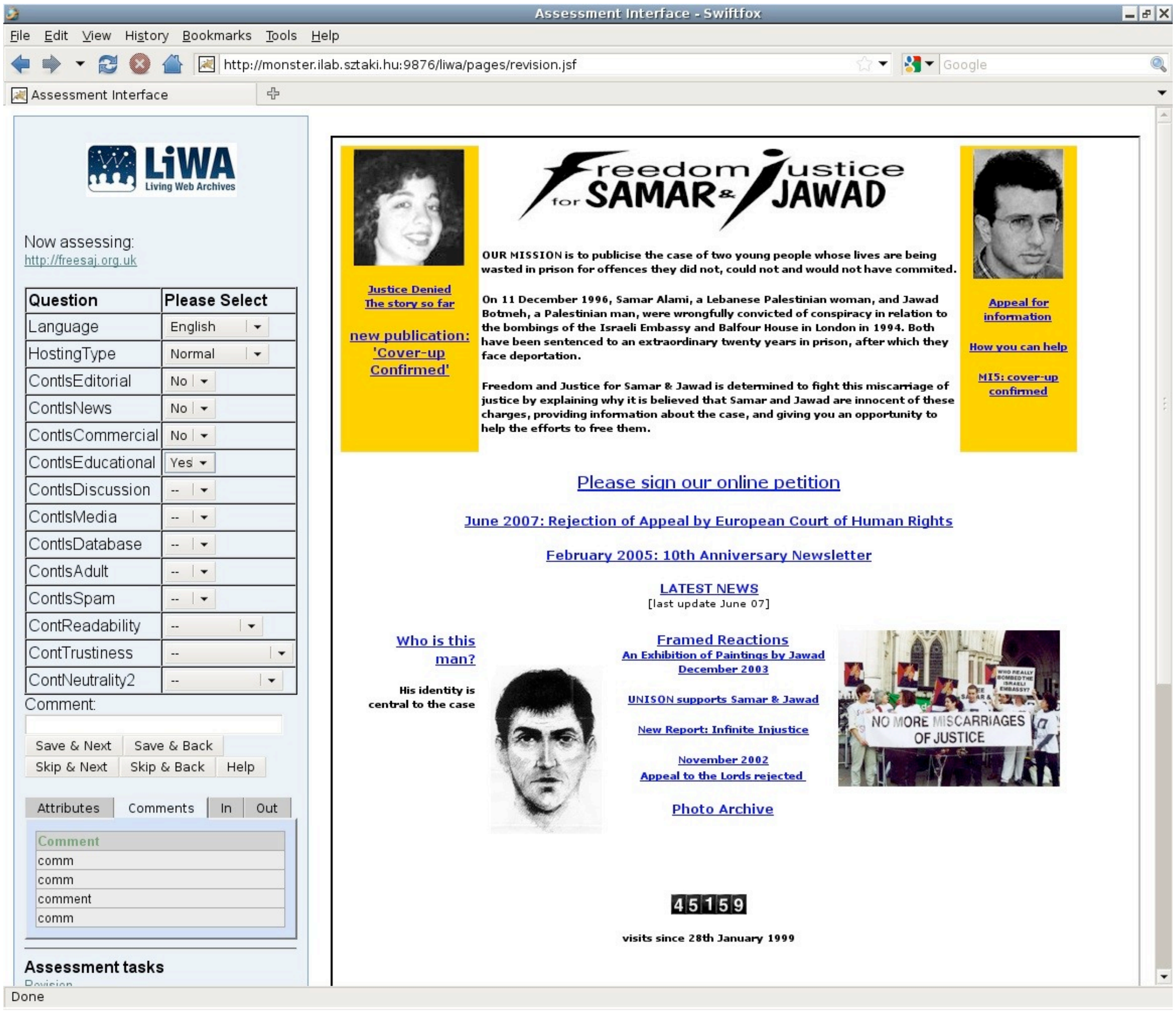
3.2.2.Assesment interface design

The assessment interface is modeled by the Web Spam Challenge 2007 volunteer interface. The right side of Figure 3.1 is for browsing in a tabbed fashion. In order to integrate the temporal dimension of an archive, the available crawl times are shown (called access bar). Upon clicking, the page which appears is the one with crawl date the closest to the crawl date of the linking page.

The selected version of the linked page can be either cached at some partner archive or the current version downloaded from the Web. We directly apply the Wayback interface of the Internet Archive, with a single modification. Since we present the archived content via IFrames, we use specific JavaScript injection in order to notify the left hand side informational window on the changes within the page view frame. We use the workaround of the IFrame security mechanisms as described in http://softwareas.com/cross-domain-communication-with-iframes.

The right side also contains in and out links as well as list or sample pages of the site. By clicking on an in or out link, we may obtain all possible information in all the subwindows from the central service.

In conjunction with the WP10 Community Platform, the interface illustrated in Figure 3.1, together with a centralized service, will act as a knowledge base also for crawler traps, crawling strategies for various content management technologies and other issues related to Web host archival behavior.



Now assessing:
http://freesaj.org.uk

Question	Please Select
Language	English
HostingType	Normal
ContIsEditorial	No
ContIsNews	No
ContIsCommercial	No
ContIsEducational	Yes
ContIsDiscussion	--
ContIsMedia	--
ContIsDatabase	--
ContIsAdult	--
ContIsSpam	--
ContReadability	--
ContTrustiness	--
ContNeutrality2	--

Comment:

Comment
comm
comm
comment
comm

Assessment tasks

Freedom Justice for SAMAR & JAWAD

Justice Denied
The story so far

new publication: 'Cover-up Confirmed'

OUR MISSION is to publicise the case of two young people whose lives are being wasted in prison for offences they did not, could not and would not have commited.

On 11 December 1996, Samar Alami, a Lebanese Palestinian woman, and Jawad Botmeh, a Palestinian man, were wrongfully convicted of conspiracy in relation to the bombings of the Israeli Embassy and Balfour House in London in 1994. Both have been sentenced to an extraordinary twenty years in prison, after which they face deportation.

Freedom and Justice for Samar & Jawad is determined to fight this miscarriage of justice by explaining why it is believed that Samar and Jawad are innocent of these charges, providing information about the case, and giving you an opportunity to help the efforts to free them.

Appeal for information

How you can help

MI5: cover-up confirmed

Please sign our online petition

June 2007: Rejection of Appeal by European Court of Human Rights

February 2005: 10th Anniversary Newsletter

LATEST NEWS
[last update June 07]

Who is this man?

His identity is central to the case

Framed Reactions
An Exhibition of Paintings by Jawad
December 2003

UNISON supports Samar & Jawad

New Report: Infinite Injustice

November 2002
Appeal to the Lords rejected

Photo Archive

45159

visits since 28th January 1999

Figure 3.1: LiWA spam assesment interface

3.2.3.Crawl-time filtering infrastructure design

In order to save bandwidth and storage, we have to filter out spam at crawl-time. For a host that is already blacklisted, we may simply discard all the URIs to save bandwidth. However for a yet unseen host we have to obtain a few pages to build a model and then apply our classifier again at crawl-time. From then on, no more URIs will be retrieved from spam hosts.

The main integration solution for the LiWA crawl-time filter is based on a separate *discovery crawler* instance that periodically gathers sample pages from hosts of the current crawl boundary. The spam filter module acts offline but over a small dedicated sample, thus achieving a fast reacting, practically online filtering mechanism.

The crawl-time filtering procedure is described in detail in the following steps.

- The *archiving crawler* instances receive sites that are already manually labeled or automatically predicted honest. Only these sites are gathered since the *archiving crawler* is not allowed to follow new hosts on the crawl boundary. This type of operation requires a frequent feed of new hosts predicted honest.
- The crawl boundary is processed by the *discovery crawler*. The boundary consists of unseen sites linked from predicted honest hosts. For these hosts the *discovery crawler* gathers sample pages in a single WARC. The number of sample HTML pages per host depends on configuration but ranges between 50-400.
- Unseen sites linked from unknown ones are not considered even by the discovery crawler in order to avoid getting trapped in a link farm.
- After the sample pages are gathered, the discovery crawler stops and the spam classifier proceeds in an offline manner. The host level features are built based on the sample pages and predictions are given based on the precompiled model. The most probably honest hosts are given as new seeds to the *archiving crawler* instances.

3.3.Off-line Coherence Analyser

The off-line coherence analysis investigates and visualizes the dynamics of the Web sites. It compares the contents and the structure of two captures of the same site. The analyser reads the captures either from a database or from WARC files. The analysis follows the crawl-time principles described in Section 5 of D6.5. The results are exported as a graph in a GraphML format. The graph visualization tool, which displays the results from the analysis, is an update of the application described in Section 2.3.2.

The analyser determines the change behaviour of a Web site. It takes as input two captures and examines the differences in their structure and contents. If the input consists of more than two captures, the off-line coherence analyser builds a full picture how the changes in the site happen. Similarly to the on-line case, the off-line analyser detects the added, the removed, and the changed pages. If a page changes, then the analyser checks if the links change as well. The result of the off-line analysis describes a property of the Web site, while the on-line analysis describes the quality of the crawl based on the visit-revisit technique.

The off-line analysis works on input both from a database and from WARC files. The database has the same structure as for the on-line coherence. The only difference is in what is compared: the off-line coherence analyser compares two different captures, while the online analyser compares a visit and an immediate revisit. The off-line coherence is extended to function over WARC files too.

As a part of the system, there is a WARC2DB tool which uses WARC files as input and imports them in the database. The coherence module accesses the WARC files through the API of the storage module. For instance, the coherence module obtains the list of WARC files for a given crawl_id, by calling the "`get_archive_files`" method. Using a loop of "`get_arc`" calls, the coherence module can directly access the content of each file in the list and extract the information needed for the analysis. For each crawl_id, the WARC2DB tool will store a new record for the given crawl capture in the database.

The off-line coherence analyser provides an API to the other modules. Currently, the proposed method returns the number of detected changes:

- coherenceMeasure = analyze (List <crawl_id>)

The implementation of the API is an ongoing work and a complete integration of the off-line coherence analyser will be available for the second version of the integrated prototype.

3.4.Semantic Analyser

The semantic analyser investigates the evolution of terms in the archive over time. As described in Section 6 of D6.5 and shown in Figure 3.2, it first extracts terms and annotates them part of speech information from the WARC files. Then term co-occurrence graphs are created for different time slices. The different graphs are clustered and are then compared to detect evolution of senses.

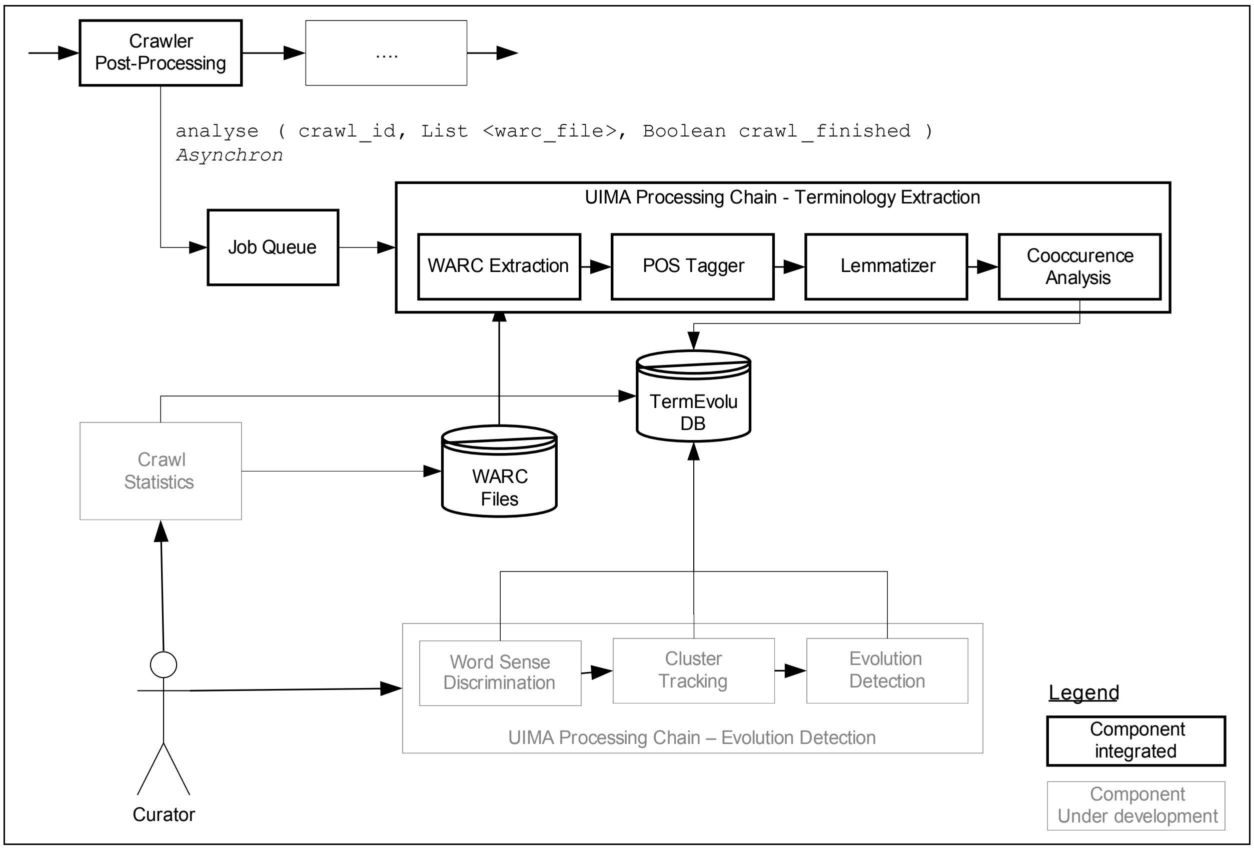


Figure 3.2: Overview of the integrated terminology evolution modules

The terminology extractor is integrated via UIMA pipelines implemented in Java. It takes as input a URL which contains a text file with a list of WARC file paths that have been crawled. The archives are then read by the UIMA WARC Reader which uses BoilerPipe (http://code.google.com/p/boilerpipe/) for extracting the textual content from the HTML pages in the archive. The next step in the pipeline is the part of speech and lemma annotation which is done by the DKPro UIMA tool (http://www.ukp.tu-darmstadt.de/projects/dkpro/).

The archive metadata is then stored in the TermEvolvDB database via the UIMA pipeline. As metadata, for each Web page in the archive, we store the title, time of crawl, the sentences annotated with their positions in the document, and the lemmas in the document. We only store lemmas that have as part of speech either noun, noun phrase, verb and conjunctions (such as “and”, “or” or commas). For each lemma we store its connection to the document, the sentence, the position and its part of speech.

For a given crawl or time span we then use the Co-occurrence analysis tool to extract lemma co-occurrence graphs from the database. We extract the graphs based on lemmas appearing together in the same sentence, in a window of words or around conjunctions. These graphs are then stored and analysed for detecting term evolution.

The Semantic Analyser API module contains the following 3 methods:

- analyze (crawl_id, List <warc_file>, Boolean crawl_finished)
- buildEvolutionGraph (crawl_id)
- List <keyword> = expand (List <keyword>, weight, start_time, end_time)

Moreover, the module uses the methods provided by the storage module, to access the WARC files in a collection, e.g.:

- List <warc_file> = get_archive_files (crawl_id)

The semantic analysis is triggered externally by the post-processing manager after a crawl is finished.