This deliverable describes the architecture of the Web crawling process inside the ARCOMEM project (work package 5 on intelligent content acquisition): its integration inside state-of-the-art Web archiving crawlers, its components, the interactions with other modules from other work packages to guide content acquisition, and the kind of information processed and produced during the crawl.
**ARCOMEM Consortium**

This document is a part of the ARCOMEM research project funded by the ICT Programme of the Commission of the European Communities by the grant number ICT-2009-270239. The following partners are involved in the project:

<table>
<thead>
<tr>
<th>The University of Sheffield (USFD) – Coordinator</th>
<th>Leibniz Universität Hannover (LUH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department of Computer Science</td>
<td>Forschungszentrum L3S</td>
</tr>
<tr>
<td>Regent Court</td>
<td>Appelstrasse 9a</td>
</tr>
<tr>
<td>211 Portobello</td>
<td>30169 Hannover</td>
</tr>
<tr>
<td>Sheffield, S1 4DP</td>
<td>Germany</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>Contact person: Thomas Risse</td>
</tr>
<tr>
<td>Contact person: Hamish Cunningham, Wim Peters</td>
<td>E-mail address: <a href="mailto:risse@L3S.de">risse@L3S.de</a></td>
</tr>
<tr>
<td>E-mail address: <a href="mailto:ARCOMEM-coord@lists.dcs.shef.ac.uk">ARCOMEM-coord@lists.dcs.shef.ac.uk</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Yahoo Iberia SLU (YIS)</th>
<th>Internet Memory Foundation (EA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avinguda Diagonal 177, 8th floor, Barcelona,</td>
<td>45 ter rue de la revolution</td>
</tr>
<tr>
<td>08018, CAT, Spain</td>
<td>93100 Montreuil</td>
</tr>
<tr>
<td>Contact person: Alejandro Jaimes</td>
<td>France</td>
</tr>
<tr>
<td>E-mail address: <a href="mailto:ajaimes@yahoo-inc.com">ajaimes@yahoo-inc.com</a></td>
<td>Contact person: Julien Masanes</td>
</tr>
<tr>
<td></td>
<td>E-mail address: <a href="mailto:julien@internetmemory.org">julien@internetmemory.org</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University of Southampton (SOTON)</th>
<th>Athens Technology Center (ATC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room 4011, Building 32</td>
<td>10, Rizariou Street</td>
</tr>
<tr>
<td>Highfield campus</td>
<td>15233, Halandri</td>
</tr>
<tr>
<td>University of Southampton</td>
<td>Athens, Greece</td>
</tr>
<tr>
<td>SO17 1BJ</td>
<td>Contact person: Dimitris Spiliotopoulos</td>
</tr>
<tr>
<td>Contact person: Paul Lewis</td>
<td>E-mail address: <a href="mailto:d.spiliotopoulos@atc.gr">d.spiliotopoulos@atc.gr</a></td>
</tr>
<tr>
<td>E-mail address: <a href="mailto:phl@ecs.soton.ac.uk">phl@ecs.soton.ac.uk</a></td>
<td></td>
</tr>
</tbody>
</table>

| ATHENA Research and Innovation Center in     | Télécim ParisTech (IT)           |
| Information Communication & Knowledge        | 46 rue Barrault                  |
| Technologies (ATHENA)                        | 75634 Paris Cedex 13             |
| National Technical University of Athens       | France                           |
| School of Electrical and Computer Engineering | Contact person: Pierre Senellart |
| Division of Computer Science                  | E-mail address: pierre.senellart@telecom-paristech.fr |
| Iroon Polytechniou 9                          |                                  |
| Athens, 15780                                 |                                  |
| Greece                                       |                                  |
| Contact person: Nectarios Koziris             |                                  |
| E-mail address: nkoziris@imis.athena-innovation.gr |                              |

<table>
<thead>
<tr>
<th>Deutsch Welle (DW)</th>
<th>SUDWESTRUNDFUNK (SWR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neue Medien / Distribution</td>
<td>Hans-Bredow-Strasse,</td>
</tr>
<tr>
<td>Voltastr. 6</td>
<td>D-76522 Baden-Baden</td>
</tr>
<tr>
<td>13355 Berlin, Germany</td>
<td>Germany</td>
</tr>
<tr>
<td>Contact person: Birgit Gray</td>
<td>Contact person: Robert Fischer</td>
</tr>
<tr>
<td></td>
<td>E-mail address: <a href="mailto:robert.fischer@swr.de">robert.fischer@swr.de</a></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>HELLENIC PARLIAMENT (HEP)</td>
<td></td>
</tr>
<tr>
<td>Amalias 14,</td>
<td></td>
</tr>
<tr>
<td>10557, Athens, Greece</td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td></td>
</tr>
<tr>
<td>Contact person: Dimitris Koryzis</td>
<td></td>
</tr>
<tr>
<td>E-mail address: <a href="mailto:dkoryzis@parliament.gr">dkoryzis@parliament.gr</a></td>
<td></td>
</tr>
</tbody>
</table>

| PARLAMENTSDIREKTION (AUP)                    |                                  |
| Dr. Karl Renner-Ring 3                       |                                  |
| A-1017 Vienna                                |                                  |
| Contact person: Guenther Schefbeck           |                                  |
| E-mail address: guenther.schefbeck@parlament.gv.at |                          |
Work Package Participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

1. Télécom ParisTech (IT)
2. ATHENA Research and Innovation Center in Information Communication & Knowledge Technologies (ATHENA)
3. Internet Memory Foundation (EA)
4. Leibniz Universität Hannover (LUH)

We also acknowledge contributions by SWR, even though this partner is not officially involved in WP5.

Change Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Amended by</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>16/05/11</td>
<td>Pierre Senellart (IT)</td>
<td>Initial detailed outline</td>
</tr>
<tr>
<td>0.2</td>
<td>17/05/11</td>
<td>Pierre Senellart (IT)</td>
<td>Distribution of tasks to partners</td>
</tr>
<tr>
<td>0.3</td>
<td>24/05/11</td>
<td>Thomas Risse (LUH)</td>
<td>Section 1.4</td>
</tr>
<tr>
<td>0.4</td>
<td>25/05/11</td>
<td>Pierre Senellart (IT)</td>
<td>Section 1.1, 1.5, parts of Sections 1 and 2</td>
</tr>
<tr>
<td>0.5</td>
<td>26/05/11</td>
<td>Pierre Senellart (IT)</td>
<td>Inversion of Sections 1 and 2. Section 1.1.</td>
</tr>
<tr>
<td>0.6</td>
<td>26/05/11</td>
<td>Yannis Stavrakas (ATHENA)</td>
<td>Heritrix feasibility</td>
</tr>
<tr>
<td>0.7</td>
<td>27/05/11</td>
<td>Dimitrios Tsoumakos (ATHENA)</td>
<td>ARCOMEM DB</td>
</tr>
<tr>
<td>0.8</td>
<td>30/05/11</td>
<td>Florent Carpentier (EA)</td>
<td>Contributions from EA in various places, minor changes</td>
</tr>
<tr>
<td>1.0</td>
<td>31/05/11</td>
<td>Pierre Senellart (IT)</td>
<td>Sections 3 and 4, all content incorporated</td>
</tr>
<tr>
<td>1.1</td>
<td>12/06/11</td>
<td>Pierre Senellart (IT)</td>
<td>Final editing before internal review</td>
</tr>
</tbody>
</table>

Executive Summary

We describe the architecture of the Web crawling process inside the ARCOMEM project (work package 5 on intelligent content acquisition). In contrast to classical Web archiving crawler, we provide facilities for extracting complex objects from the Web, for changing the behavior of a crawl depending on the kind of Web application currently crawled, and for interacting with content analysis modules through either online or offline interfaces and having them influence the prioritization of the crawl. The crawler used is the one developed at the Internet Memory Foundation, but we also plan to provide partial support of the Heritrix open-source crawler by integrating some of the functionalities of the ARCOMEM project with Heritrix and by exporting from the output of the crawl a definition file usable together with a standard Heritrix installation. Our main interaction mode is database-centric, where a HBase repository of information serves as a central point for storing all information produced by various modules.
Table of Contents

ARCOMEM Consortium..................................................................................................................2
Work Package Participants...........................................................................................................3
Change Log..................................................................................................................................3
Executive Summary......................................................................................................................3
Table of Contents .......................................................................................................................4
List of Tables ..............................................................................................................................5
List of Figures .............................................................................................................................5

1. Architecture of the ARCOMEM Crawling System.................................................................6
   1.1 General Architecture .........................................................................................................6
   1.2 Crawler Used in the ARCOMEM Project ......................................................................7
   1.3 Internal Architecture of the Web Crawler ..................................................................8
       Architectural principles .......................................................................................................8
       The crawling process ..........................................................................................................10
       The HBase process ............................................................................................................10
       The DHT output and input processes .............................................................................10
       Data structure ...................................................................................................................11

2. Intelligent Content Acquisition: Software Components .....................................................12
   2.1 Application-Aware Crawling ..........................................................................................12
       A knowledge base of Web applications .........................................................................12
       Detecting Web applications ...........................................................................................13
       Crawling actions ................................................................................................................13
   2.2 Crawling and Archival of Complex Content from the Web ...........................................14
       Forms ..................................................................................................................................14
       AJAX.................................................................................................................................14
       Application-specific APIs ...............................................................................................15
   2.3 Interfacing with the ARCOMEM Database ...................................................................15
       Storage of captured resources: HBase .............................................................................16
   2.4 Prioritization and Focusing of the Crawl .......................................................................17
       The crawl definition .........................................................................................................17
       Semantic impact ...............................................................................................................18
       Impact of the social Web .................................................................................................18
       Combining everything into a final ranking .....................................................................19
   2.5 Export of Crawl Definitions ..........................................................................................19

3. Online and Offline Interfaces to Content Analysis Modules ..............................................20
   Inputs (from the crawler to content analysis modules) ..........................................................20
   Outputs (from content analysis modules to the crawler) .......................................................20
   Time available .......................................................................................................................21
   Architecture of the interface ...............................................................................................21

4. Data Model ............................................................................................................................23

Citations and References.............................................................................................................25
List of Tables

Table 1. Ranking impact depending on various notions of relevance ........................................... 18

List of Figures

Figure 1. Traditional processing chain of a Web crawler .......................................................... 6
Figure 2. Processing chain of the ARCOMEM crawler .............................................................. 6
Figure 3. Preliminary Architecture of the Web crawler system ............................................... 9
Figure 4. Crawling processes ................................................................................................. 10
Figure 5. Crawler data structure implementation using Bloom filters .............................. 11
1. Architecture of the ARCOMEM Crawling System

1.1 General Architecture

We contrast the architecture of the ARCOMEM crawler (represented in Figure 2) with that of a traditional Web archiving crawler (Figure 1).

Consider first the simplified view of the architecture of a traditional Web crawler depicted on Figure 1. In a traditional Web crawler, such as Heritrix [S05], the archiving task is described using a crawl definition configuration file [IA11] that specifies the list of seed URLs to start the crawl from, patterns that specify which URLs to include or exclude from the crawl, etc. At runtime, URLs are managed in a priority queue, which ensures optimal allocation of the bandwidth available to the crawler given the URLs of the frontier (URLs discovered but not yet crawled) and politeness constraints. Web pages are then fetched one after the other, links are extracted from these Web pages and the crawl definition is used to determine whether the URLs found on a Web page should be added to the frontier.

We extend on this architecture in the ARCOMEM project, adding new functionalities to the crawling system as shown on Figure 2. This figure represents the different components implemented as part of the project (intelligent analysis modules, whether online or offline; are the outcome of work packages 2, 3, and 4, while the other components represented here will be developed as part of work package 5). The references shown in the figure point to the sections of this report where each component or interaction between components is discussed in more detail.

The traditional crawl definition file is replaced by an intelligent crawl definition, which allows specifying scoring and making references to the particular kinds of Web applications and of Web data that define the scope of the archiving task, as identified by the other ARCOMEM modules.
(Section 3.2). Queue management functions similarly as in a traditional architecture, but the page fetching module is replaced by some more elaborate resource fetching component that is able to retrieve resources that are not just accessible by a simple HTTP GET request (but by a succession of such requests, or by a POST request, or by the use of an API), or individual Web objects inside a Web page (Section 2.2). Each content item obtained by the crawler is stored in the ARCOMEM database for use by analysis modules and archivist tools (Section 2.3). After a resource (for instance a Web page) is fetched, an application-aware helper module is used in place of the usual link extraction function, in order to identify the Web application that is currently being crawled, decide and categorize crawling actions that can be performed on this particular Web application (Section 2.1). These crawling actions, depending on their nature (embeds or other Web resources) are either directly forwarded to the selection component or sent for further analysis and ranking to the online analysis modules (Section 3.1) provided by work packages 2, 3, 4. Since crawling actions are more complex than in a traditional crawler, and since we want to prioritize the crawl in an intelligent manner, the URL selection module is replaced by a resource selection & prioritization module that makes use of the intelligent crawling definition and of the feedback of the online analysis modules to prioritize the crawl (Section 2.4). Finally, in order to enhance the reusability of the intelligent analysis performed during the crawl in other environments, one of the outcome of a crawl campaign by the ARCOMEM crawler will be a standard Heritrix job configuration file [IA11] whose purpose is to launch new archiving campaigns with the same scope that the campaign just performed, using the well-spread version of Heritrix (Section 2.5).

1.2 Crawler Used in the ARCOMEM Project

Heritrix is the Internet Archive’s open-source Web crawler, well-spread and fairly well used among archiving institutions. It is a pure Java program. In different areas of the crawl (pre-fetch chain, queue distribution...) several behaviours are implemented in specific classes. The use of these classes, their parameters when applicable, and their order of use can be configured. This is done in a job configuration: a fairly complex XML file that reflects the Java configuration objects. Specifying a complex behaviour demands a deep knowledge of Heritrix’s internals. In addition, at runtime, the crawler can be controlled through a RESTful interface. Basic start/stop/pause commands can be easily achieved by simple HTML GETs and POSTs, however for more complex behaviour like pausing the fetching of a specific site or adjusting the scope of the crawl one needs to know the code of Heritrix in order to use the enhanced scripting functionality offered for this purpose.

The Internet Memory Foundation (IMF, partner of the ARCOMEM project, formerly known as European Archive or EA) is currently developing its own crawler (still in development stage and not officially named yet). This is proprietary software, entirely implemented in Erlang. Configuration and control of a crawl, however specific, is done through a user-friendly Web interface.

In ARCOMEM, two major constraints apply to the crawler: speed and fine-grained scope adaptation at runtime.

Focusing on the social Web, a rapidly changing medium, imposes certain constraints on the crawler. As information quickly evolves, time and efficiency is of the essence. Distribution is a key design element to achieve this. Heritrix is designed as a single instance type of crawler, with some additions to help with distribution. However, the standard distribution does not provide tools to distribute URLs across different servers and the instances remain completely independent from one another. One instance creates files with the URLs to be sent to another, the user having to manage the transfer of data between them. In addition, cluster changes are not catered for. In particular, nothing is done when one instance goes down, and the resources that were supposed to be handled by it will not be crawled. IMF’s crawler is designed from the ground up as a distributed crawler. The number of instances can vary during the crawl, as when a new instance is
added the others will equally share the load, and when one is going down, its work is automatically
distributed among the remaining ones, without any user intervention.

Adaptive crawling strategy translates to modification of the scope during the crawl. In Heritrix the
scope is global: one cannot define several scopes for different categories of resources, and once
the crawl has been started, only minor scope adaptations can be made, such as broadening the
scope. Also even the simple pausing or exclusion of a site would imply some development in the
Heritrix specific scripting environment. IMF’s crawler has an adaptive scope mechanism that allows
the user to change it at all times during the crawl, and we could envisage defining different crawl
strategies for different classes of resources. Also the pausing or complete removal of a site from
the crawling list, during the crawl, would be just one click away from the user-friendly Web
interface.

A serious constraint imposed to the crawler is the ARCOMEM storage environment. For this it has
been envisaged to use HBase as the permanent storage, the one with which the crawler will
interact. Heritrix outputs the crawled resources as ARC or WARC files. Some development will be
necessary in order to add further functionalities. Also it cannot be guaranteed that all the resources
of a Web page will be found in the same file, if that should be the constraint of any of the analysis
modules. IMF’s crawler is designed to store captured resources in HBase, granting a flexible and
efficient access to them, better suited to post-processing.

In conclusion, the limitations of Heritrix lead us to base the ARCOMEM system primarily on the
Web crawler developed by IMF (discussed further in Section 1.3). We also explain how we can
leverage the output of a crawling campaign in a usual crawling environment using Heritrix in
Section 2.5.

For dissemination purposes, it is also desirable to demonstrate the functionalities of ARCOMEM’s
modules with an open-source crawler, such as Heritrix. However, due to limitations of Heritrix
discussed earlier, it is first necessary to assess the feasibility of such a task. The feasibility test will
investigate the implementation of a Heritrix frontier, which is responsible for prioritizing the URLs to
crawl, and adapted modules in the crawler processing chain, which are responsible for fetching
pages and extracting links from the fetched pages. In line with the focus of the WP5 effort, the
implemented frontier and modules must demonstrate the functionalities required by an adaptive
crawler, which qualifies and prioritizes hyperlinks leading to Web pages relevant to a specific topic
of interest. The feasibility test will also assess the possibility to adapt Heritrix to perform limited
interpretation of JavaScript, as well as to handle AJAX requests. The output of the feasibility test
consists of a report documenting the implementation efforts to integrate Heritrix with ARCOMEM’s
modules, and the adapted Heritrix modules.

1.3 Internal Architecture of the Web Crawler

We describe here the internal architecture of IMF’s crawler that will be used as a basis for the
development of the ARCOMEM system.

Architectural principles

Figure 3 shows a preliminary view of the architectural principles. In general, the system can be
viewed as composed of three applications (shown as yellow boxes): (i) the distributed crawler
(made of several local instances, each shown as a yellow box), (ii) the repository, and (iii) the
monitoring/control system. The crawler itself is distributed Erlang software. Choosing Erlang allows
to benefit from its built-in inter-process communication primitives. Such communications actually
occur at two levels:
1. as a means to create and monitor local instances of the crawler, called crawler nodes in the following; each crawler instance is in charge of a subset of hosts;

2. and, locally, as a means to glue together the crawler modules that cooperate to achieve the many tasks involved in collecting resources from the Web (i.e., harvesting resources, parsing and discovering URLs, enforcing politeness rules, etc.).

Each crawler module can be implemented in an appropriate programming language, typically a language with low-level types for efficient data structures implementation (Java, C) or a specialized language for some quite specific tasks (e.g., Perl for text manipulation, JavaScript for Web page execution, etc.). Local storage (disks) can be used for temporary persistency of the local data structure (for instance, Drum).

Each crawler node communicates with the two other external applications.

**Document storage.** Collected resources are sent to the IMF HBase archive repository. This is a one-way transmission.

**Monitoring and control.** This is a two-way transmission that exchanges messages from the crawler node to the monitoring application (which performs some kind of aggregation over the set of crawler nodes), and from the monitoring application to the node. The former provides statistics and useful information to the monitoring task. The latter consists of actions that affect the crawler behaviour.

Distribution is based on consistent hashing (CH) in a distributed hash table (DHT) structure and uses domains as keys to distribute the load over the participating list of servers. We refer to [AMR+11] for information about the way DHTs can be used for scalable data indexing in a cluster. Basically, each crawler node (i) gets resources from the Web, (ii) sends the resource to the IMF repository, (iii) inspects the resource content to discover new URLs; and (iv) forwards each URL to the crawler node in charge of it. The latter is done by hashing the domain extracted from the URLs in the DHT ring. This design is influenced by some of the major recent contributions to the field, i.e., IRLBot and UbiCrawler [BCSV02].

---

1 http://irl.cs.tamu.edu/crawler/
A crawler node consists of a set of coordinated processes (see Figure 4 for a summary). Each process is implemented in Erlang, with built-in support for inter-process communication. The processes partly communicate around a data structure which is used to filter out unseen URL and perform other cleaning tasks (e.g., spam detection).

We give a brief overview of each process below, along with a preliminary description of the API.

**The crawling process**

This process is in charge of collecting resources from the Web. It repeatedly pops a URL from the data structure, sends a GET HTTP request, and retrieves the resource. The resource is sent to the HBase process, and its content analyzed (if hyperlink type). Each discovered URL is sent to the DHT-output process.

In a crawler node, there might be many (≈ 100) crawling processes operating in parallel. Each crawling process is dedicated to a host, and respects the crawling politeness rules for its host.

Note that we may sometimes have to execute some pages with high value and JavaScript-generated content. This is ignored in the present document.

**The HBase process**

This Erlang process (one per crawler node) is in charge of communicating with the HBase repository. It receives resources and some meta-data from a crawler process, and transmits them to HBase.

**The DHT output and input processes**

The DHT output process is part of the DHT layer that supports the cooperation of crawler nodes. This process receives a URL extracted from a freshly harvested resource, computes the hash of the URL, determines from a local routing table the crawler node in charge of this hash and sends it...
the URL. More precisely, the DHT-output communicates with the DHT-input or the target crawler node.

The DHT-input process is the counterpart of the above. It receives URLs collected by other crawlers that must be processed by the local crawler node. It then puts these URLs in the data structure.

**Data structure**

The data structure is dedicated to efficiently filter out URL which have already been visited by the local crawler node. It supports the following methods:

1. enqueue(URL)
   Add a URL to the structure.
2. dequeue(host): URL
   Retrieves an unseen URL from the structure.

For efficiency reasons, the structure may need to accumulate a lot of URLs before starting the internal filtering process. We may therefore have to wait during the early stage of the crawling process. A flush operation can be added to overcome this behaviour.

Figure 5 shows the sketch of an implementation relying on Bloom filters (a hashing structure used for compact indexing) to store a trace of already visited URLs and priority queues for the frontier. A hash table, indexed by hostnames, points to these structures. To simplify the design, the URL priorities will be enforced per host.

![Figure 5. Crawler data structure implementation using Bloom filters](image-url)
2. Intelligent Content Acquisition: Software Components

We present here the different software components that will be implemented as part of the work package 5 of the ARCOMEM project. The goal of all these components is to make a Web archiving crawler able to acquire content from the (social) Web in an intelligent and adaptive manner. We focus in this section on the presentation of individual modules that enrich the functionality of the crawler. We defer the presentation of the interfacing between the crawler and intelligent analysis modules from work packages 2, 3, and 4 to Section 3.

2.1 Application-Aware Crawling

Current-day archiving crawlers process Web sites independently of the nature of the site (a traditional content site, a social network, a wiki, a Web forum, etc.) and of the software that powers this site (e.g., the particular content management system used in this site): links are extracted from Web pages, and all pages pointed to by these pages, if they are in the scope of the archiving task, are put in a queue for further downloading. This behaviour might lead the crawler to lose resources in archiving irrelevant parts of a Web page (e.g., the edition page associated with all pages of a wiki system), in processing in the same way pages that are critical to archive and pages which should only be archived if resources allow (e.g., in some archiving tasks, news articles vs comments associated to these articles) or in missing content that is hard to reach (because, for this particular Web application, it can only be crawled through a form interface, an HTTP API, or an AJAX application). The only way an archivist or a Web crawler programmer can circumvent this problem is by adding manually lists of URLs to exclude per site, or by integrating ad hoc behaviour to the crawling software on a site per-site basis.

The goal of this software component is to make a crawler aware of the particular kind of Web application they are crawling, in terms of its general classification (wiki, social network, blog, Web forum, etc.), its technical implementation (MediaWiki, WordPress, etc.), and even its specific instance (Twitter, CNN, etc.), in order to adapt the crawling strategy to the task at hand.

A knowledge base of Web applications

The crawler will rely on a hierarchical knowledge base of Web applications, which specifies for a given kind of Web application how to recognize it and how to crawl it. This knowledge base is hierarchical: general categories of application (such as social Web site, traditional Web site, utility Web applications, etc.) can be subdivided in specific categories (wikis, blogs, Web mails, etc.) which can be separated depending on the particular software or content management system used, down to the level of specific Web sites. At the moment, we consider hand-written descriptions of such Web applications, but ultimately these could be learned from annotated Web pages. One particular goal is that such a knowledge base will be written in a declarative language, so that it can be maintained more easily and by a larger population (typically, Web archivists) than the Web crawling software itself. A given Web application usually regroups Web pages of different kinds (e.g., in a blog system, there are pages that display list of blog posts, and pages that display individual blog posts with their comments). Consequently, the knowledge base will be able to describe the different types of Web pages that appear in a given application, and how they should be processed.
Detecting Web applications

A particular Web application (or a particular kind of page inside a Web application) can be described by the following elements in the knowledge base:

- URL patterns;
- HTTP metadata;
- Textual content (keywords, key phrases, or regular expressions, to look for in the text and comment of a page);
- Structure (XPath queries describing patterns that identify the Web application);
- Reference to classifiers, learned from examples;
- (possibly) More Web graph-based features, such as the revisit patterns identified in the course of the FP7 LiWA project2.

We will work on a number of use cases suggested by the ARCOMEM application partners to check whether these features are enough to identify Web applications. The software component integrated into the crawler will then look up the knowledge base to discover the most precise match, if any, of the currently crawled Web page. Note that an important issue that will have to be tackled is that of efficiently determining which of the various applications described in the knowledge base corresponds to the current page.

Crawling actions

In a sense, the application-aware helper integrated into the crawler is a replacement for the traditional link extraction module. As such, it is supposed to extract from a Web page, given its Web application, which URLs to crawl on the Web page. But the kind of crawling actions to be performed on a Web page goes further than just a list of URLs, it can be:

- The use of an API to directly get relevant data (e.g., in Twitter);
- The description of a complicated interaction with a Web server to get data that is hidden behind a Web form or an AJAX application (e.g., which field of a form to submit, what HTTP request to perform, etc.)
- The identification of individual Web objects to extract from a Web page (e.g., individual posts in message boards) for individual storage and preservation in the ARCOMEM database.

To describe these crawling actions, the knowledge base will use the OXPath language [FGG+11], a Web navigation language developed in the course of the ERC Diadem project, that describes in a very precise way a sequence of actions to be performed on a sequence of Web pages (e.g., click here, fill in this form, etc.) together with the location of individual Web objects to extract in a Web page. At the moment OXPath does not allow to describe all possible interactions that we might want to use for crawling a Web application (e.g., HTTP POST requests that do not rely on form submissions). Thus we will work closely with the Diadem team to augment the OXPath language as required.

In addition, each of these crawling actions are associated with labels (e.g., “blog post”, “user profile”, “news article”) that will allow an archivist to better describe the scope of an archiving campaign. So the archivist can set up priorities for the archiving of specific parts of a Web site in the definition of a crawl task (e.g., in a blog system, pages containing comments might have a lower priority than pages containing the main content of the blog) (see Section 3.2).

---

2 http://www.liwa-project.eu/
All crawling actions to perform will be passed on to the URL selection module that will decide on their prioritization and eventual execution.

2.2 Crawling and Archival of Complex Content from the Web

Forms

As long as the data is submitted with a GET method, we can assume we are free to try to submit as many inputs as we like. In the case of POST requests, in general, we will not send anything to avoid causing undesirable side effects. However, for some Web sites (specifically identified by an operator or on the basis of an automatic detection of a certain type of technology), we will want to submit specific fixed data or even try automatically generated input.

Architecture: after downloading a page identified as HTML, a fetcher will look for forms and submit them to the form-filling module (format: HTML snippet). The module answers with a list of inputs (a value for each form item).

```javascript
form_filling:values(
    "<form action="/a" method="GET"><input name="city">...
</form>"
)
returns

[[{"city", "Paris"}, ...],
[{"city", "Amsterdam"}, ...]]
```

The fetcher fetches the URLs built from that. Each downloaded resource is then submitted to the form-filling module, that determines whether the page is worth being examined for links and archived. At access time, filling a form with values that were not used by the crawler will naturally cause a "not found in the archive" page to be returned.

AJAX

If we want to view the archived pages, we can identify two possibilities:

- Store each request’s answer and let the JavaScript code run at access time:
  1. fetch the main page, the embeds and run the XMLHttpRequest requests,
  2. store the answer to each GET or POST XMLHttpRequest,
  3. let the user's browser run the JavaScript normally (if the URLs in the JavaScript code are relative, they should work) (note: as always, we are faced with the problem of the myriad possible parameter values, and different sequences of actions that can yield not found pages, or incorrect content.) We could envisage to hold session information to serve different content for a URL based on the previous actions, but this looks very complicated for a limited benefit.

- Store a serialisation of the whole page after each action
  1. fetch the main page at current URL $U$ and its embeds,
2. execute different actions on page \( U \). After each action, store a serialisation of the resulting page under an `extended' URL, made of \( U \) and a sequence of OXPath actions leading to it (making sure no collision with the site's URLs can occur).

3. at access time, when browsing page \( U \), rewrite the links that trigger a JavaScript action to their corresponding extended URL. Detecting these links from the bare HTML page may prove challenging, especially if they are generated by JavaScript code.

The former seems simpler, and will be tried first.

In any case, an execution environment must be used to execute the JavaScript code and trigger actions by simulating user input. The integration and/or adaptation of existing tools such as htmlunit [http://htmlunit.sourceforge.net/] or Crowbar [http://simile.mit.edu/wiki/Crowbar] will be studied.

*Note:* Google has proposed a scheme to let Web sites use special links that can be transformed by the crawler to access similar static content. (This relies on Web sites owners providing a version of each page that is meaningful without JavaScript, possibly automatically, running a headless browser to run the JavaScript and serialise the output (e.g., htmlunit).) See [http://code.google.com/Web/ajaxcrawling/docs/specification.html](http://code.google.com/Web/ajaxcrawling/docs/specification.html). We will investigate whether it is worth implementing this in the course of the ARCOMEM project (at crawl time and when rewriting links at access time).

### Application-specific APIs

In addition to being used to find links, the output of queries to APIs should certainly be archived as any other content retrieved during a capture. Similar to application-aware crawling, URLs pointing to APIs can be detected and processed by the crawler in a special way. The fetching phase may require adaptations, just like the link extraction or Web objects extraction. These specific rules and code will be written per application at first, as they are likely to vary widely.

As long as the data is served over HTTP, we can use the URL to name the content. To name Web objects extracted from an answer, the same scheme as for XML-based content (including HTML, as it can be converted to XHTML) will be used. If a need to extract Web objects from data encoded in a different way surfaces, instead of defining a selection language adapted to the specific format (e.g. for JSON, use field names and indexes into lists), we can define a transformation from the format to XML.

### 2.3 Interfacing with the ARCOMEM Database

The crawler components are interacting with the ARCOMEM database with high frequency. In particular, the final output of a crawling campaign (i.e., a set of Web objects in the form of WARC files) will be saved for application use to the Web Archive. Moreover, intermediate cleansed objects will be also stored for offline analysis. Finally, the crawler needs information from the annotations stored in the ARCOMEM database that may influence the direction of future crawls. Thus, it is imperative that we define clear and efficient ways for the crawler to interface with the database.

The ARCOMEM database stores semantic information that is collected during offline processing of Web objects. The crawler queries these annotations in order to receive information that will further guide the crawling process in combination with the online processing. Currently, both the exact data model used for the annotations as well as the respective schema of the ARCOMEM database is under consideration. We are exploring the solution of adopting an RDF schema to describe metadata and a triplet store scheme for storing and querying. Thus, the exported functionality will enable the crawler to perform simple queries to identify if a specific entity, topic, etc. (i.e., a subject, object or triplet predicate) exists or is linked in the current object. This will be in the form of a
get() call, whose parameters can either be an ID or a keyword that describes an annotation object. In any case, while performance is an issue, the ARCOMEM database is interfaced by many modules. The above query functionality can be exposed through RESTful calls. The ARCOMEM database constitutes the resource, which will be accessed via a common interface based on the HTTP standard methods. In the course of the project, the solution chosen for the interface between the crawler and the database will be tested to assure real-time response times expected.

**Storage of captured resources: HBase**

Resources are stored in an HBase table [http://hbase.apache.org/](http://hbase.apache.org/). HBase is a NoSQL database written in Java and modelled after Google's BigTable [http://labs.google.com/papers/bigtable.html](http://labs.google.com/papers/bigtable.html). Like all DBMS in this class, it sacrifices some query expressiveness (compared to SQL DBMSs) to achieve extreme scalability. It is also very flexible in the columns definition: each table is a sparse map associating a value to (key, column family, qualifier, date)-tuples.

Storing the output of crawls requires the ability to store large numbers of resources (especially when adding individual Web objects), representing a large overall volume. In addition, different type of extra data must be attached to different type of resources (storing extracted text only makes sense for HTML or PDF documents, for instance, not pictures). HBase covers all these needs.

In order to provide high availability of the data, HBase keeps its data in distributed file system called HDFS. This approach also diminishes the impact of a node failure on the overall data base system performance.

HBase can be queried from a client over the network, or run distributed Hadoop computations. For the former, Java classes are provided or a RESTful interface using XML or protobuf can be used. Avro and Thrift interfaces are also being developed.

In the case of IMF, the Java APIs are used for the communication between the software and the database, since they are very easy to use, especially when using the distributed processing Hadoop framework. The data itself is stored in one table which is divided into two column families. One column family stores the actual crawled content and the second one stores the meta data associated with the actual crawled content. The primary key for each row is then the crawled URL. This organisation is very convenient considering the parallel processing and HBase's intrinsic column oriented way of storing data, e.g., we are able to carry out fast analyses of the data nature by extremely efficient sequential read of the metadata information.

HBase allows addition of column families after the initial schema definition as well. In the meanwhile, we are considering storing for each crawled resource besides the original data, for instance, extracted links pointing out from the page, or the list of pointers to resources embedded in the page, e.g., images. In general, such additional information may be stored at crawl time if available or might constitute a result of a post processing done arbitrarily upon the crawled data. Therefore, on demand the data model can be enriched.

The APIs for communication with the database are very similar to those known from the SQL/ODBC world. For instance, when inserting into the database, a data object is first created. Then, the values for the particular column qualifiers for the afore-mentioned column families are set as raw byte values and sent to the server. The inserts are cumulated in a buffer of a predefined size to minimize the network communication overhead.

The querying is done using either a point query approach (one particular row is retrieved from the DB using the URL and API GET) or through sequential reading of a range of URLs with the Scanner API. In both cases the user can control which versions of the datum – a Web resource is to be retrieved.
The following example demonstrates the usage of the Java APIs to retrieve all the versions for the given URL: `http://www.nytimes.com/index.html` into a multidimensional Map where the key is the column qualifier pointing to another Map where the key is the timestamp (time of crawl) and the value the stored resource's content. Similarly, the meta information can be accessed using the same URL but the "meta" column family instead of the "content" one.

```java
Get get = new Get(Bytes.toBytes("http://www.nytimes.com/index.html"));
get.addFamily(Bytes.toBytes("content"));
Result r = hTable.get(get);
NavigableMap<byte[], NavigableMap<Long,byte[]>> resultMap =
    r.getMap().get(Bytes.toBytes("content"));
```

### 2.4 Prioritization and Focusing of the Crawl

The task of prioritization and focusing a crawl is the step in the crawl processing chain which combines the different analysis results and the crawl definition for filtering and ranking the URLs of a seed list. The filtering of URLs is necessary to focus a crawl and to avoid unrelated content in the archive. For content that is relevant to a certain degree URLs need to be prioritized to focus the crawler tasks to first crawl high relevant pages and to schedule less relevant pages to a later stage.

For Web crawls a number of strategies and therefore URL ordering metrics exist. Standard methods like Breadth-First, back link count or PageRank are examples. It has been shown in a number of publications like [CGP98, BCM+05] that PageRank and Breadth-First are good strategies to crawl “important” content on the Web. Since these generic approaches do not cover specific information needs, focused or topical crawls have been developed like [CBD99, AAY01, MPS04]. Since current work on focused crawling has a vague notion of topicality in the next section we describe which parameters will be available within the ARCOMEM project that are suitable for guiding the crawl.

### The crawl definition

In general, ranking of URLs should be driven by the crawl specification given by the archivist. The crawl specification consists, among others, of a number of content related parameters:

- search string, e.g. [event:“Rock am Ring 2011”] and ([person:“Coldplay”] or [person:“Beatsteaks”])
- social media categories, e.g., social networks
- demographics, e.g., age=20 - 40
- content: e.g., language=German
- media, e.g., all
- users, e.g., all

Most of these parameters describe attributes of a page, of a user or of a site. These attributes can easily be used for filtering and ranking as they do not need any interpretation. A challenge for the ranking is the absence of these attributes. Attributes like the age of a user require accessibility to the user profile. If these attributes are not accessible, they will be ignored in the ranking.

The importance of each of these parameters for the ranking depends on the aim of the crawl campaign. So, in a crawl campaign where the target is to document and analyse the impact of rock festivals on students, the content created by elderly people is less interesting. Therefore the demographic aspects get a higher weight than the social media category. Vice versa, a campaign focusing on micro-blogging sites will give the social media category a higher weight than the...
demographics. Another example is a campaign to document the opinions of major social Web contributors to an event. In this case the user parameter will have more importance than the demographics or the social media category.

Since the intention of an archivist behind a crawl specification cannot be derived automatically, the crawl parameters should be weighted by the archivist to give the ranking algorithm more hints on the importance of a parameter.

Semantic impact

In most cases the search string will have a high impact on the ranking of a seed list. The archivist specifies, with the help of the search string, in his own words the semantic focus of a crawl campaign. In the crawl specification above the search string states that the archivist is interested in pages about the event “Rock am Ring” and the musicians “Coldplay” and “Beatsteaks”. In a strict interpretation the string implies that a page must be strongly related to the event and to one of the musicians. In practice it will not be possible to detect these semantic relations of a page or microblog from the search string with such a high level of confidence. The ETOE (Entities, Topics, Opinions and Events, see deliverable D3.1) analysis will only provide confidence values for each detected ETOE and a relevance value for the analysed page with respect to the crawl specification. As a consequence of a strict interpretation the crawl coverage might not be best way to go. So some degree of freedom in the interpretation is necessary. On the other side a too relaxed interpretation could results in few resources with relevant content in the archive. Therefore the level of relaxation will depend on the intention of the archivist and should therefore be specified by him or her. Furthermore, the archivist needs to regularly analyse the crawled content and to adapt the crawl specification and especially the weights accordingly.

Impact of the social Web

Beside the direct interpretation of the search string and the degree of freedom for the interpretation, the “wisdom of the crowds” can be used to prioritize the ranking of URLs. The underlying idea is that users who discuss an event or topic on social Web sites have a better understanding of the topic and thus have a more complete overview of related pages. Therefore they recommend certain pages to other users and might get feedback from them. By taking these recommendations into account for the ranking, the human perception of a topic can be reflected in the archive. It supports and complements the semantic analysis as it will not always be possible to judge the relevance of a page with high confidence.

Since the social Web can easily be influenced and misused for pushing certain pages or sites, the reputation of a user has to be taken into account.

The table below provides a first simplified view of the impact of the social Web relevance and user reputation on a ranking that is initially based on the semantic relevance. With social Web relevance we mean on how relevant a page is seen by a user or the community. The table is intended as a starting point for the discussion in the project to develop a deeper understanding on what is meant with user reputation, social Web relevance and semantic relevance.

<table>
<thead>
<tr>
<th>Semantic relevance</th>
<th>Social Web relevance</th>
<th>User reputation</th>
<th>Ranking impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Very high ranking</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Strong increase of ranking</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>More research necessary (tendency: no change of ranking)</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Strong decrease of ranking</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Slight increase of ranking</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>No change of ranking</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>No change of ranking</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Decrease of ranking</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td>Unavailable</td>
<td>More research necessary (tendency: increase of ranking)</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>Unavailable</td>
<td>More research necessary (tendency: increase of ranking)</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>Unavailable</td>
<td>No change of ranking</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td>Unavailable</td>
<td>No change of ranking</td>
</tr>
</tbody>
</table>

Combining everything into a final ranking

For the final ranking many parameters need to be combined and weighted, which at least partially influence each other. Some parameters are complex in their nature and their potential impact on the ranking requires more research. Therefore early versions of the ranking method will use Breadth-First strategy in combination with a subset of the parameters mentioned as a starting point. Throughout the project more parameters will be added when they become available.

2.5 Export of Crawl Definitions

A final module that will be developed focuses on the export of ARCOMEM crawling results to a standard Heritrix installation. The desired output is a crawl job configuration file in the format accepted by the Heritrix archiving crawler [IA11].

There are obviously limitations to this: no intelligent analysis can be performed during the crawl, it is impossible to fine-tune the prioritization of the crawl, and Heritrix is unable to perform some crawling actions such as HTTP POST requests. However, in the case where the archiving task is such that the Web sites and URLs to archive do not change significantly or frequently over time, this feature will allow any archiving institution to use their usual archiving setup, making use of the ARCOMEM intelligent crawler only when it is needed to (re-)establish the scope of the crawl to be performed.
3. Online and Offline Interfaces to Content Analysis Modules

Content analysis modules pertaining to social Web analysis (work package 2), entity, event, and topic extraction (work package 3), and opinion mining (work package 4) will be able to influence the crawler in two possible ways: first, in an online mode, by analyzing the content of crawled Web resources and providing feedback to the crawler as it crawls a given Web page, helping it prioritize future crawling actions; second, in an offline mode, by processing the whole output of a crawl and building from this a new *intelligent* crawl definition. The analysis carried out online must be done relatively fast (from seconds to minutes) in order to have an impact on the crawl in progress, and has for input a single Web resource (though the analysis module is free to look for other information from the ARCOMEM database). In contrast, offline analysis can last as long as one is willing to wait (typically, of the order of days). It can process the whole content of the ARCOMEM database and study in depth a given crawl or even the dynamics of different crawls. But the ways offline analysis can influence a subsequent crawl are less subtle. We describe next both interfaces.

3.1 Online Interface: Hooks to Online Analysis Modules

Inputs (from the crawler to content analysis modules)

As a result of a crawling action, and after processing by the application-aware helper, the crawler provides the following content to analysis modules:

- Timestamp of the request
- HTTP headers as key-value pairs (if applicable)
- Raw content of the resource (e.g., image, raw HTML Web page)
- Categorization of the resource and of the corresponding Web site by the application-aware helper, as a list of tags describing the Web application (e.g., *social-site*, *twitter*, *twitter-profile*)
- Individual Web objects extracted by the application-aware helper (e.g., individual comments on a blog post page), with, for each:
  - Content of the Web object, as both raw text and HTML fragment (when applicable)
  - Categorization of the Web object by the application-aware helper, as a list of tag (e.g., *comment*)
- Links (or, more generally, subsequent crawling actions described as OXPath expressions) extracted from the Web resource, with, for each:
  - URL or OXPath expression
  - Anchor text or other textual context
  - Categorization of the link by the application-aware helper

Outputs (from content analysis modules to the crawler)

The analysis module provides feedback on the resource crawled and the links extracted from it as follows:

- A global assessment of the relevance of the whole resource, as a number between 0 and 1
- For each Web object, an *optional* assessment as a number between 0 and 1
• For each link (or crawling action), an optional assessment as a number between 0 and 1

A score of 1 has the semantics that it is fully in the scope of the current crawling campaign, of 0 if it is fully out of scope. Crawling actions with a score of 0 will not be performed, and crawling actions will be prioritized according to their score. If no assessment is provided of the links, a score will be set by default from the assessment of the resource itself.

Time available

As already discussed, an online analysis module has some limited time to process a resource. Observe first that the feedback will mostly be on crawling actions on the same Web server as the current resource. It would therefore be ideal if it could be sent to the crawler during the time the server needs to wait (for crawling ethics reasons) before issuing a new request to the same Web server (i.e., of the order of the second). Alternatively, the feedback can be sent later on to the Web crawler (say, after up to an hour), assuming that some other crawling actions need to be performed on this server in the mean time. This would be at the expense of potentially delaying the archiving of the content and of losing some crawling resources archiving undesirable content. It is probably not reasonable to expect feedback given even later to be helpful for crawl prioritization, and lengthy analysis should be deported to the offline mode.

Architecture of the interface

Since there is no real-time requirement on the crawler, it will communicate asynchronously with the analysis modules. This makes it less dependent on the analysis module speed. Of course, the sooner these modules send the outcome of their analysis, the sooner it will be taken into account and the more effective the crawl will be.

After fetching a page and performing a basic analysis of it (type determination, link extraction...), the page will be stored into the HBase repository and a message will be sent to all applicable analysis modules. This message will contain either the reference of the page (URL and date) to let it be fetched from HBase, or directly the page and metadata. The former could be simpler, but may lead to excessive load on the HBase system.

In the other direction, the crawler will wait for the input of scored crawling actions at any time.

We give a simplified example of a possible interface using JSON over HTTP, but other interfaces might be possible depending on the requirement of online modules. For instance, notifying the Social link extraction module that a page was just crawled could be achieved with the message:

```
{
    "notification_type": "page_retrieved",
    {
        "url": "http://example.com/",
        "date": "2011-05-30 12:00:00 CET"
    }
}
```

After retrieving the information from the ARCOMEM database (assuming the information is not provided directly in the message), it may wish to submit new weighted URLs to the crawler, with the following:

```
{
    "operation": "url_addition"
}  
```
3.2 Offline Interface: Intelligent Crawl Definitions

In the offline mode, content analysis modules influence subsequent crawls by updating the definition of the crawl. We say the crawl definition is intelligent because 1) the scope of the crawled content can be defined with some scoring between 0 and 1, allowing to prioritize the crawl; 2) the crawl definition can refer to Web application categories that will be identified during the crawl; 3) it can also include semantic information that will be used by online analysis modules.

- Seed URLs or crawling actions (in particular, a crawling action might specify to query a search engine with some key strings to initialize the set of seeds), with scoring
- URL regexps defining the scope, with scoring (a score of 0 implying an exclusion)
- Web application categories (e.g., twitter-profile, blog-comment-page) with scoring
- Keywords or key phrases to look for in text resources and textual context of crawling actions, with scoring
- Additional semantic information about the scope in a key=value format that is not processed by the crawler itself, but that can be used by online analysis modules to give feedback to the crawler (e.g., age_min=30, language=German, etc.)
4. Data Model

We describe here the data model of the information produced and processed by content acquisition modules. Produced information will typically be stored in the ARCOMEM database, and processed information read from the ARCOMEM database (except for information directly exchanged between modules, but this will be limited to a minimum). We do not mention data elements that are internal to the crawler and therefore are not not part of the ARCOMEM database, such as the data structure needed to manage the frontier, or the Web application knowledge base. We describe each individual entity in bold font, with references to other entities in italics.

- **Crawled Web resource**
  - URL or OXPath expression acting as an identifier
  - Timestamp of the request
  - HTTP headers as key-value pairs (if applicable)
  - HTTP POST data (if applicable)
  - Raw content of the resource
  - MIME type of the resource (in particular, image, video, HTML, text) possibly different from the declared HTTP MIME type
  - Categorization of the resource
  - Web objects extracted from the Web page
  - Crawling actions identified from this Web page

- **Categorization** of a resource, crawling action, or Web object
  - List of tags, referring to the Web application knowledge base

- **Web object**
  - Reference to original Web resource the Web object was extracted from
  - XPath expression identifying the Web object inside the Web resource
  - Raw text content (if applicable)
  - HTML fragment (if applicable)
  - Categorization of the Web object

- **Crawling action**
  - Reference to original Web resource the crawling action was found on
  - URL or OXPath expression
  - Anchor text or other textual context
  - Categorization of the link

- **Intelligent crawl definition**
  - List of seed crawling actions, with scoring
  - URL regexps, with scoring
  - List of categorizations, with scoring
  - Keywords or key phrases, with scoring
  - Additional semantic information in a key=value format
Implementation of the data model in the ARCOMEM database

The way this data model will be stored in the ARCOMEM database will be documented in the ARCOMEM internal documentation, elaborating on table names, attribute names, and value ranges associated to each data item. This global schema is also to be integrated with the schema of other information stored in the ARCOMEM database, especially for the purposes of work packages 2, 3, 4.
Citations and References


