Intelligent Archiving of the Social Web

Pierre Senellart, Télécom ParisTech

DIADEM 1.0 Workshop, October 3, 2011

The ARCOMEM project (FP7 IP, 2011–2013)

Objective

Leverage the social Web to build rich Web archives about specific topics, using intelligent mechanisms: semantic analysis, focused crawling, social network mining, etc.

Examples

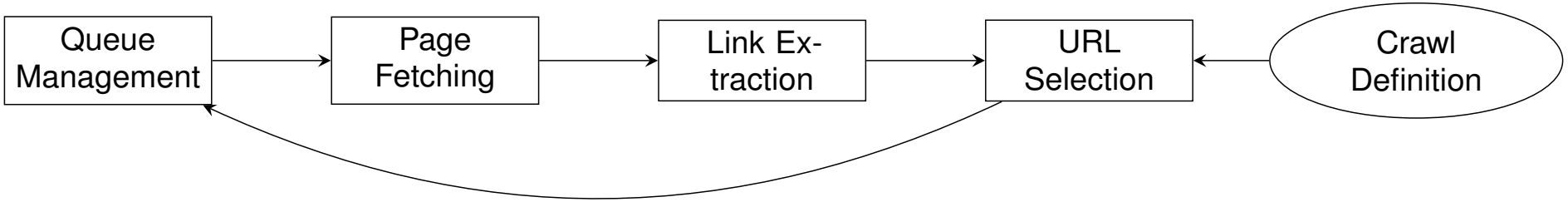
Understand and preserve for future generations:

- What people on the Web think about the Greek economic crisis
- How a music festival is perceived in social networks

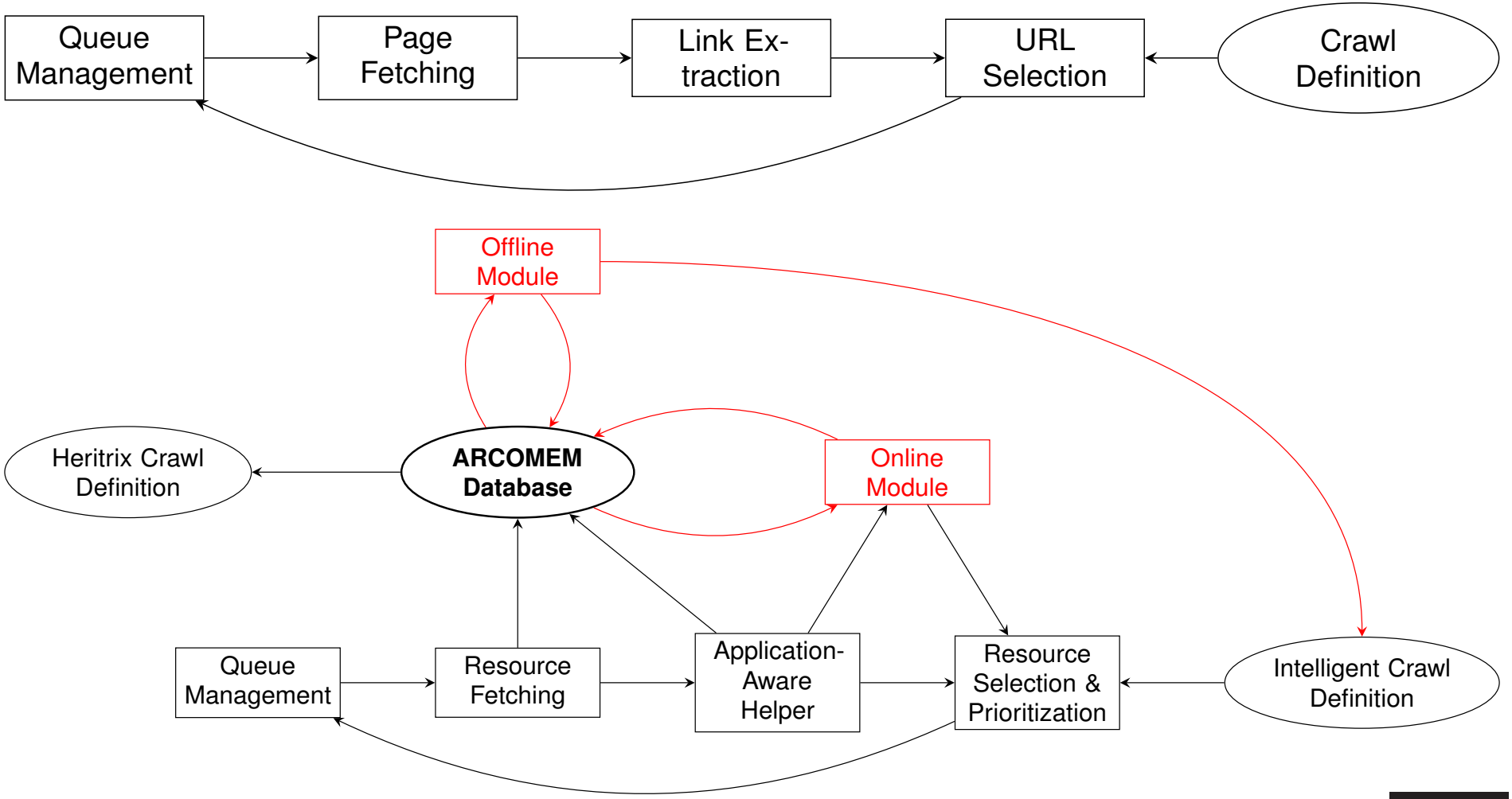
Remark: Archivists are concerned with preserving Web content exactly as it appeared, with as few loss as possible in the process.

The ARCOMEM consortium

Architecture of a classical crawler



Architecture of the ARCOMEM crawler



Analysis modules

- Entity recognition
- Event extraction
- Topic recognition
- Sentiment analysis
- Social graph mining

The modules are responsible for focusing the crawl: changing the crawl definition, give (asynchronous) scored feedback on Web content crawled and crawling actions to be performed.

Crawling Web applications

The optimal strategy for crawling different kinds of Web sites and Web applications varies a lot:

- In a Wiki, do not follow the “Edit this page” link
- On Twitter, you should follow AJAX events to create full pages
- On Facebook, nothing can be crawled:

  ```
  User-agent: *
  Disallow: /
  ```

  but the Facebook API can be used to get part of the content
- A Dotclear blog can be archived efficiently by following the monthly archives, and disregarding other links
- Certain Web sites need to be accessed through Web forms

In addition, social Web sites contain Web objects (blog posts, tweets, user profiles, etc.), to be archived by themselves.

Application-aware helper

Crawlers should be aware of the Web application they are crawling! We assume a knowledge base, describing a hierarchy of Web applications (site type, CMS used, specific sites), featuring:

- detection of the type of the currently crawled Web application
- description of crawling actions to perform, depending on the Web application

This KB can be manually created (by developers? by archivists?) or, ultimately, mined from the Web.

A navigation and extraction language for ARCOMEM

Requirements for a language describing crawling actions:

- URLs to follow
- JavaScript and AJAX actions to perform
- Web forms to submits
- parts of Web pages to extract
- unbounded number of iterations
- (possibly) RESTful API interaction

A navigation and extraction language for ARCOMEM

Requirements for a language describing crawling actions:

- URLs to follow
- JavaScript and AJAX actions to perform
- Web forms to submits
- parts of Web pages to extract
- unbounded number of iterations
- (possibly) RESTful API interaction
- The language should remain declarative (hope to have expressions written by a non-expert, or learned from examples)
- Efficient processing (no slowing down of the crawler, at least not more than needed by layout rendering)

A navigation and extraction language for ARCOMEM

Requirements for a language describing crawling actions:

- URLs to follow
- JavaScript and AJAX actions to perform
- Web forms to submits
- parts of Web pages to extract
- unbounded number of iterations
- (possibly) RESTful API interaction
- The language should remain declarative (hope to have expressions written by a non-expert, or learned from examples)
- Efficient processing (no slowing down of the crawler, at least not more than needed by layout rendering)

OXPath is our best bet!

Minor things we miss in OXPath

- Fine control on the AJAX waiting delays
- POST requests without form submissions
- XPath navigation in JSON documents (does it work in XML documents?)
- Regular expressions to break up text nodes?
- More generally, what features (say, from XPath 2.0) can be added without hurting efficiency?
- A way to feed back Web content into OXPath patterns
- A browser less buggy than HTMLUnit ☺

Minor things we miss in OXPath

- Fine control on the AJAX waiting delays
- POST requests without form submissions
- XPath navigation in JSON documents (does it work in XML documents?)
- Regular expressions to break up text nodes?
- More generally, what features (say, from XPath 2.0) can be added without hurting efficiency?
- A way to feed back Web content into OXPath patterns
- A browser less buggy than HTMLUnit ☺

Eager to get our hands on the Mozilla/Webkit version!
We can also help!

Marrying OXPath with a crawler

Ideally, every network interaction should go through the crawler, this helps:

1. archival of content
2. prioritization of crawling actions
3. policy management (crawling ethics)
4. parallelization

A few directions to have this work:

- The crawler acts as a HTTP proxy for the OXPath processor
- The state of the OXPath processor can be dumped and restored at a later point
- Or perhaps some finer interaction. Don't you have a similar need for DIADEM?

The OXPath hammer

- OXPath is a pretty big hammer. . .
- Not always useful, some crawling actions can just be described by URLs and XPath
- We also want to avoid the cost of the layout engine as much as we can

This raises the following two questions:

- Can we detect when a layout engine is not needed? (No DOM-modifying JS code? No automatically launched DOM-modifying code?)
- Can we detect whether an OXPath expression can actually decomposed into URL fetching and XPath evaluation sequences, without any JS execution?

Other interests of mine relevant to DIADEM

- JavaScript analysis for understanding Web sites
- Probabilistic databases for information extraction
- Optimizing the number of requests to a given Web site
- Extraction of the main content of a Web page using external semantic clues (RSS feeds, anchor text, etc.)
- Combining form analysis, probing, and result page analysis into a deep Web understanding loop