Not So Creepy Crawler: Easy Crawler Generation with Standard XML Queries

Franziska von dem Bussche
vondembu@cip.ifi.lmu.de

Klara Weiand
klara.weiand@ifi.lmu.de

Benedikt Linse
linse@pms.ifi.lmu.de

Tim Furche
tim@furche.net

François Bry
bry@lmu.de

University of Munich, Oettingenstr. 67, 80538 Munich, Germany

ABSTRACT

Web crawlers are increasingly used for focused tasks such as the extraction of data from Wikipedia or the analysis of social networks like last.fm. In these cases, pages are far more uniformly structured than in the general Web and thus crawlers can use the structure of Web pages for more precise data extraction and more expressive analysis.

In this demonstration, we present a focused, structure-based crawler generator, the *"Not so Creepy Crawler"* (NC^2). What sets NC^2 apart, is that all analysis and decision tasks of the crawling process are delegated to an (arbitrary) XML query engine such as XQuery or Xcerpt. Customizing crawlers just means writing (declarative) XML queries that can access the currently crawled document as well as the metadata of the crawl process. We identify four types of queries that together suffice to realize a wide variety of focused crawlers.

We demonstrate NC^2 with two applications: The first extracts data about cities from Wikipedia with a customizable set of attributes for selecting and reporting these cities. It illustrates the power of NC^2 where data extraction from Wiki-style, fairly homogeneous knowledge sites is required. In contrast, the second use case demonstrates how easy NC^2 makes even complex analysis tasks on social networking sites, here exemplified by last.fm.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Design, Experimentation, Languages

Keywords

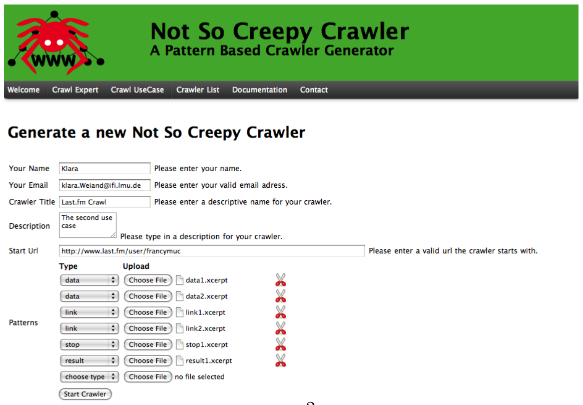
Web crawler, data extraction, XML, Web query

1. INTRODUCTION

Things haven't been going well for your company lately and you know what's at stake when your boss tells you: *"We have to give a party for our investors, and you have to make sure, that they all like the music we play to get them into the right mood. All the information should be on last.fm, where I have added the investors to my social network."*

Fortunately, you have attended a few WWW conferences in the past and know to look to crawlers and data extraction.

Unfortunately, large-scale crawlers, used in Web search engines, do not provide the granularity to solve this task.



Generate a new Not So Creepy Crawler

Your Name	Klara	Please enter your name.
Your Email	klara.Weiand@ifi.lmu.de	Please enter your valid email adress.
Crawler Title	Last.fm Crawl	Please enter a descriptive name for your crawler.
Description	The second use case	Please type in a description for your crawler.
Start Url	http://www.last.fm/user/francymuc	Please enter a valid url the crawler starts with.

Patterns

Type	Upload
data	data1.xcerpt
data	data2.xcerpt
link	link1.xcerpt
link	link2.xcerpt
stop	stop1.xcerpt
result	result1.xcerpt
choose type	no file selected

Figure 1: Expert nc^2 crawler interface

Focused crawlers [4, 7] which aim at accumulating high quality collections of data on a predefined topic are not suitable either as they cannot easily identify pages of investors and generally do not allow to compare data from different crawled Web pages. On the other hand, data extraction tools [1, 2] have long been used successfully to extract specific data (such as the music tastes of an investor) from Web pages, but they either do not provide crawling abilities or allow only limited customization of the crawling.

In this demonstration, we introduce the *"Not so Creepy Crawler"* (NC^2), a novel approach to structure-based crawling that combines crawling with standard Web query technology for data extraction and aggregation. NC^2 differs from previous crawling approaches in that all data (object and metadata) is stored and managed in XML format. The crawling process is entirely controlled by a small number of XML queries written in any XML query language: some queries extract data (to be collected), some links (to be followed later), some determine when to stop the crawling, and some how to aggregate the collected data.

This allows easy, but flexible customization through writing XML queries. By virtue of the loose coupling between an XML query engine and the crawl loop, the XML queries can be authored with standard tools, including visual pattern generators [2]. In contrast to data extraction scenarios, these same tools can be used in NC^2 for authoring queries of any of the four types mentioned above.

You quickly author the appropriate queries and generate and run a new NC^2 crawler using the Web interface shown in Figure 1[1] Let the party begin!

[1]And available at `http://pms.ifi.lmu.de/ncc`.

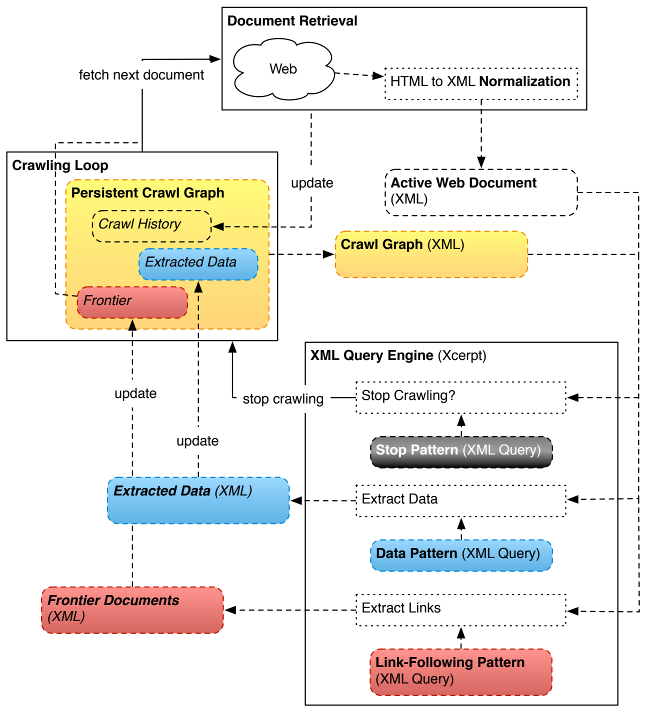


Figure 2: Architecture "Not So Creepy Crawler"

2. CRAWLING WITH XML QUERIES

2.1 "Not So Creepy Crawler": Architecture

The basic premise of NC^2 is easy to grasp: A crawler where all the analysis and decision tasks of the crawling process are delegated to an XML query engine. This allows us to leverage the expressiveness and increasing familiarity of XML query languages and provide a highly configurable crawler generator, which can be configured entirely through declarative XML queries.

To this end, we have identified those analysis and decision tasks that make up a focused, structure-based crawler, together with the data each of these tasks requires. Figure 2 gives an overview of the architecture of NC^2 with focus on the various analysis and decision tasks.

2.1.1 *XML patterns*

Central and unique to a NC^2 crawler is uniform access to both object data (such as Web documents or data already extracted from previously crawled Web pages) and metadata about the crawling process (such as the time and order in which pages have been visited, i.e., the crawl history). Our *crawl graph* not only manages the metadata, but also contains references to data extracted from pages visited previously. It is worth noting that the tight coupling of the crawling and extraction process allows us to retain only the relevant data from already crawled Web documents.

This data is queried in a NC^2 crawler by three types of XML queries (shown in the lower right in Figure 2):

(1) *Data patterns* specify how data is extracted from the current Web page. A typical extraction task is "extract all elements representing events if the current page or a page linking to it is about person X". To implement such an extraction task in a data pattern, one has to find an XML

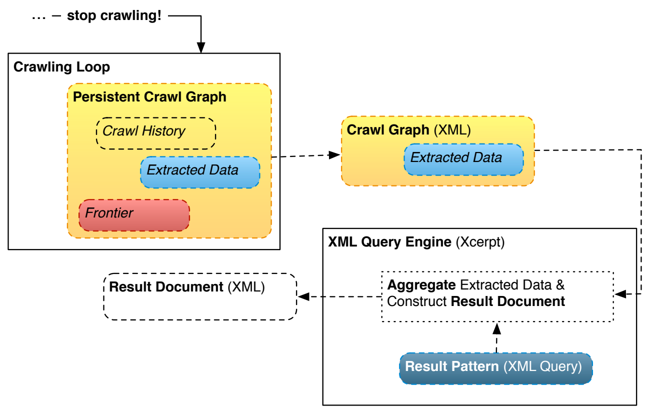


Figure 3: Result document construction

query that characterizes "elements representing events" and "about person X". As argued above, finding such queries is fairly easy if we crawl only Web pages from a specific Web site such as a social network.

(2) *Link-following patterns* extract all links from the current Web document that should be visited in future crawling steps (and thus be added to the crawling frontier). Often these patterns also access the crawl graph, e.g., to limit the crawling depth or to follow only links in pages directly linked from a Web page that matches a data pattern.

(3) *Stop patterns* are boolean queries that determine when the crawling process should be halted. Typical stop patterns halt the crawling after a given amount of time (i.e., if the time stamp of the first crawled page is long enough in the past), number of visited Web pages, number of extracted data items, or if a specific Web page is encountered.

There is one more type of pattern, the *result pattern*, of which there is usually only a single one: It specifies how the final result document is to be aggregated from the extracted data. Figure 3 shows this finalization phase: Once a stop pattern matches and the crawling is halted, the result pattern is evaluated against the crawl graph and the extracted data, e.g., to further aggregate, order, or group the crawled data into an XML document, the result of the crawling.

All four patterns can be implemented with any XML query language. In this demonstration we use Xcerpt [6, 3].

2.1.2 *System components*

How are these patterns used to steer the crawling process? Crawling in NC^2 is an iterative process. In each iteration the three main components (rectangles with solid borders in Figure 2) work together to crawl one more Web document:

(1) The *crawling loop* initiates and controls the crawling process: It tells the document retrieval component to fetch the next document from the crawling frontier (the list of yet to be crawled documents).

(2) The *document retrieval* component retrieves and normalizes the HTML document and tells the crawling loop to update the crawl history in the crawl graph (e.g., to set the document as crawled and to add a crawling timestamp).

(3) The *XML query engine* (in the demonstrator, Xcerpt) evaluates the stop, data, and link-following patterns on both the active document and the crawl graph (containing the information which data patterns matched on previously crawled pages and the crawl history). Extracted links and data are sent to the crawling loop which updates the crawl graph.

(4a) If none of the stop patterns matches (and the frontier is not empty) the iteration is finished and crawling starts again with the next document in step (1).

(4b) If one of the stop patterns matches in step (3), the crawling loop is signalled to stop the crawling. As depicted in Figure 3, the XML query engine evaluates the result pattern on the final crawl graph and the created XML result document is returned to the user.

2.2 Implementation

As described above, the implementation of NC^2 is independent of the actual XML query language used. For this demonstrator we use Xcerpt [6, 3] as its query-by-example style eases query authoring where we have an example Web page and try to formulate a query accordingly. However, replacing Xcerpt with, e.g., XQuery is, from the view point of NC^2, as easy as changing a configuration file.

In the above description (and the current implementation), the persistent crawl graph is implemented as an in-memory data structure that is serialized each time a new document is crawled. This proves to be sufficient for small crawl graphs. For larger crawl graphs, those parts of the query patterns that are evaluated against the crawl graph should be evaluated incrementally against the updates triggered by data or link extraction and history updates [5].

3. DEMO SETUP AND DESCRIPTION

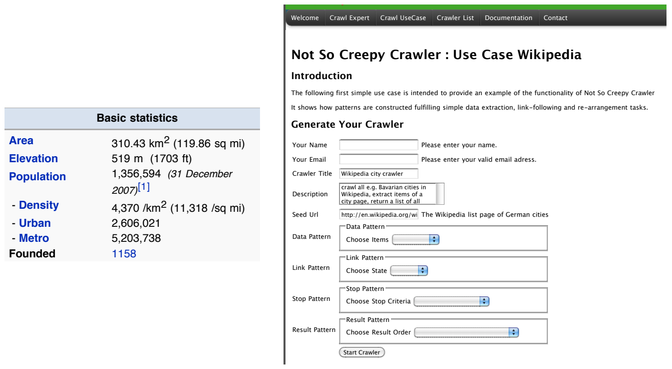
The demonstration is built around two applications in the area of knowledge extraction and social networks that demonstrate the power and ease of pattern-based crawling. The NC^2 interface is publicly accessible over a Web interface at `http://pms.ifi.lmu.de/ncc`. The interface allows the easy generation of new crawlers by providing a seed URL and the requisite patterns (see Section 2.1). During the crawling process, a generated crawler can be examined online, crawling results are available on the website or via email. Crawler generation with the Web interface is possible in two modes: *(a)* expert mode, in which the user can load her own patterns, and *(b)* demo mode, which provides predefined patterns for our use cases which can be changed or extended by the user.

3.1 Application #1: Extracting City Information from Wikipedia

The pattern-based crawling approach is particularly useful on large websites that contain Web pages with similar structure for the same kind of information. Wikipedia is among the largest such sites that offer somewhat structured knowledge. The most valuable structure of that knowledge is contained in so-called info-boxes, each type of info-box adhering to a particular schema. Different types of info-boxes are used for persons, companies, US presidents, operating systems, historical eras, political parties, countries, skyscrapers, etc. The application described in this section deals with cities, but can be easily adapted to any of the other Wikipedia categories.

Assume we would like to find out more about Bavarian cities: *"Find all Bavarian cities in Wikipedia, extract items (such as the population, the names and/or the elevation) from the city pages and return a list of all resulting cities ordered by city name or population."*

Wikipedia info-boxes for cities contain, amongst others,



Basic statistics	
Area	310.43 km^2 (119.86 sq mi)
Elevation	519 m (1703 ft)
Population	1,356,594 *(31 December 2007)*[1]
- Density	4,370 /km^2 (11,318 /sq mi)
- Urban	2,606,021
- Metro	5,203,738
Founded	1158

Not So Creepy Crawler : Use Case Wikipedia

Introduction

The following first simple use case is intended to provide an example of the functionality of Not So Creepy Crawler

It shows how patterns are constructed fulfilling simple data extraction, link-following and re-arrangement tasks.

Generate Your Crawler

Your Name		Please enter your name.
Your Email		Please enter your valid email adress.
Crawler Title	Wikipedia city crawler	
Description	crawl all e.g. Bavarian cities in Wikipedia, extract items of a city page, return a list of all	
Seed Url	http://en.wikipedia.org/wi	The Wikipedia list page of German cities
Data Pattern	Data Pattern: Choose Items	
Link Pattern	Link Pattern: Choose State	
Stop Pattern	Stop Pattern: Choose Stop Criteria	
Result Pattern	Result Pattern: Choose Result Order	

Figure 4: Wikipedia info-box and nc^2 demo interface

entries for their name, state, country, area, elevation, population, its time-zone, postal code and website (see left hand of Figure 4). In this use case we are only interested in the names and the population of the cities, but given the example data extraction patterns, users can easily adjust the crawler to extract additional information.

The right hand of Figure 4 shows the demo mode of the NC^2 interface. In demo mode the user is given a more limited choice between several different data, link, stop and result patterns, that can be further customized by the user. The user can select info-box items (such as the population or name of a city) from a dropdown field and the corresponding pattern is shown in the input form. In this way, beginners get easily acquainted to the formulation of queries in a do-it-yourself style. In *expert-mode* (see Section 3.2), fully customized crawling tasks are possible, as the user can upload any pattern.

The following is an example of a data extraction pattern in Xcerpt that extracts the population of a city from its info-box (observe that in Wikipedia info-boxes the label of the population property is an adjacent td to the actual value):

```
in{ resource { "document.xml","xml"},
  and { desc h1 [[
    attributes { class [ "firstHeading" ] },
    var Name ]],
    desc tr [[
      td{{ desc /.*Population.*/ }},
      td{{ /(var A → [0-9]+),*(var B → [0-9]+),
               *(var C → [0-9]+)/ }}
      ]] } }
```

Besides the crawl patterns introduced in the previous section, the web interface expects a seed URL to initialize the frontier. We pick the list of all German cities in Wikipedia (we could also start with the list of all cities).

In addition to the data extraction pattern, the application also uses a very basic link-following pattern to only crawl cities in Bavaria. The user can select different stop patterns to stop the crawling process after a given amount of time, after a given number of extracted data items or after a given number of websites crawled.

3.2 Application #2: Last.fm Crawl

This second use case solves a quite similar problem as the vision described in the introduction: *Given a last.fm user name, a list of artists that the user and his last.fm friends like is created, augmented with information about which users are fans of the respective artists.* last.fm, the largest social music platform on the internet, provides music-related services as well as social features. For this applica-

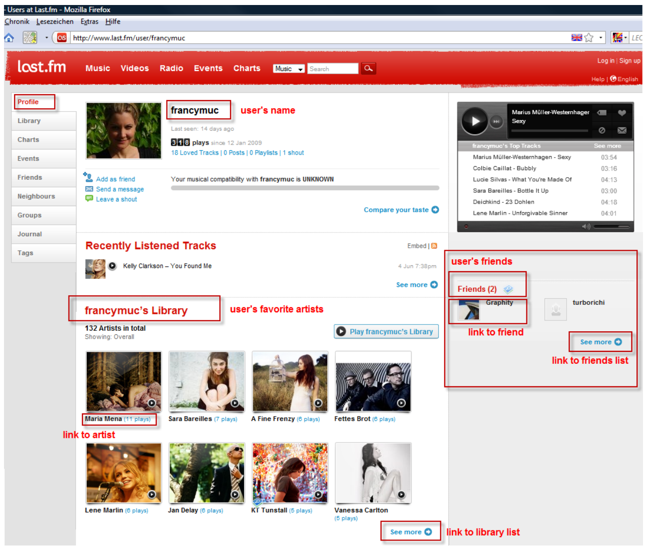


Figure 5: A last.fm profile page. The data relevant to this crawling task is highlighted.

tion, we will make use of the information who a user's last.fm friends are and what artists the users have listened to.

The expert mode crawling setup for this use case is shown in Figure 1: In the crawling task, two different types of pages are relevant: user profile pages and artist pages. In contrast to the link structure in the first use case, the structure of links that must be followed to reach all these relevant pages from our seed URL is not flat: As is common in social websites not all information about a user can be reached directly from his profile page. Instead, only some items are linked directly, others can only be reached via intermediate pages. For example, last.fm user profiles list up to six randomly selected friends. A link *"see more"* leads to a complete list of friends which in turn may be paginated. The same is true for a user's favorite artists. This relatively complex link structure must be represented in the link following patterns.

Traditional crawlers typically avoid to crawl URLs multiple times in order to prevent infinite loops. They keep a list of seen URLs and do not add the same URL to the frontier twice. However, for this task, the duplicate information is essential as the aggregation of information in this crawling task requires that some URLs (those pointing to artists that are common to several users) are treated more than once. Therefore, NC^2 lets the user indicate in the pattern and in which circumstances duplicates should be avoided.

To illustrate workflow and data structures employed in this use case, a *pre-result* containing data for two friends who share a favorite artist is shown below. It shows a list of those friends names. The node_id attributes identify the Web pages in the crawl graph that each data item originates from, the matched_id attribute the data extraction pattern that matched with the item:

```
<pre-result>
   <data node_id="1" matched_id="1">
      <name>francymuc</name>
   </data>
   <data node_id="2" matched_id="1">
      <name>turborichi</name>
   </data>
   <data node_id="5" matched_id="2">
      <name>Maria Mena</name>
   </data>
<pre-result>
```

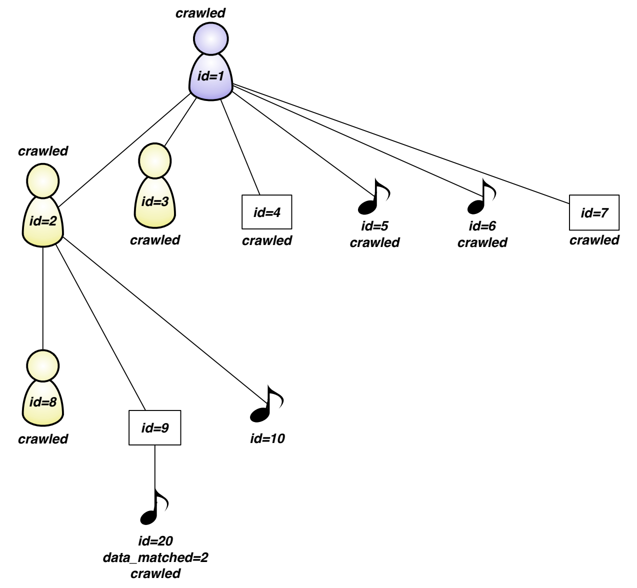


Figure 6: The crawl graph

The corresponding crawlgraph which holds the link structure and some additional annotations is shown in Figure 6: The crawling starts at the person page with id 1 and follows links both to other person pages (denoted by little men) and to artists (denoted by notes). Since some of these pages (like 20) are not reachable directly, also intermediary pages are followed as specified in the link-following pattern. If a page has been crawled, it is annotated as crawled, all pages without this annotation form the frontier. The artist page 20 shows an annotation like in the pre-result: the data extraction pattern 2 matched with this page.

Returning to the original query task, we can determine which artists are liked by a user from the crawl graph: They are those artists whose pages are reachable from the user over any number of other pages except other users.

4. REFERENCES

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, 2003.

[2] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *VLDB*, 2001.

[3] F. Bry, T. Furche, B. Linse, A. Pohl, A. Weinzierl, and O. Yestekhina. Four lessons in versatility or how query languages adapt to the web. In F. Bry and J. Maluszynski, *Semantic Techniques for the Web*, LNCS 5500, 2009.

[4] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *WWW*, 1999.

[5] M. El-Sayed, E. A. Rundensteiner, and M. Mani. Incremental maintenance of materialized XQuery views. In *ICDE*, 2006.

[6] S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Extreme Markup Languages*, 2004.

[7] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti. Structure-driven crawler generation by example. In *SIGIR Conf.*, 2006.