YAWN: A Semantically Annotated Wikipedia XML Corpus

Ralf Schenkel Fabian Suchanek Gjergji Kasneci
Max-Planck-Institut für Informatik, Saarbrücken, Germany
{schenkel,suchanek,kasneci}@mpi-inf.mpg.de

Abstract: The paper presents YAWN, a system to convert the well-known and widely used Wikipedia collection into an XML corpus with semantically rich, self-explaining tags. We introduce algorithms to annotate pages and links with concepts from the WordNet thesaurus. This annotation process exploits categorical information in Wikipedia, which is a high-quality, manually assigned source of information, extracts additional information from lists, and utilizes the invocations of templates with named parameters. We give examples how such annotations can be exploited for high-precision queries.

1 Introduction

1.1 Motivation

Much of the existing work on structure-aware XML retrieval [AY$^+$02, STW05, Sch02, TW02] has anticipated the existence of a huge number of heterogeneous XML documents with descriptive (i.e., self-explaining and semantically rich) tags. This has always been predicted as the natural consequence of the flexibility and variety of XML, where every author of a Web page can invent a different schema for her data on the Web. Given a semantically rich structural query like `//person[about(//work, physics) and about(//born,Germany)]`[1] (i.e., find people who work in physics and were born in Germany), such search engines would either consider structural similarity metrics of the query and documents [AY$^+$02, Sch02] or semantic similarity of the tags used in the query and in the documents, using an ontology as background knowledge [STW05, The03].

However, this revolution still has to happen. XML, even though widely used nowadays, usually is either generated from a structured database or used to store textual information with some structure. In the latter case, documents are content-rich, but tags are generic, while in the former case, there may be meaningful tag names, but there is often not much textual content. Therefore, today's typical XML search engines either ignore structural information completely, focussing on keyword queries alone, or deal with semantically weak structures as in `//article[about(.,XML)]//section[about(./`

[1]This query is formulated in NEXI [TS04], the query language of the INEX benchmark [INE06], but could be expressed similarly in XPath with FullText extensions.

`/paragraph,retrieval)]` (see [FLMK06] for an overview of many XML search engines).

This paper aims at bridging the gap between semantics-aware XML search engines and real world data by adding semantics to XML data, extending the ideas already used in SphereSearch [GSW05] and, more recently, the work by Chu-Carroll et al. [CC$^+$06]. Based on the well-known and widely used Wikipedia collection, this paper presents *YAWN*[2], making the following important contributions: (1) It shows how Wikipedia pages can be converted to XML, and (2) it presents algorithms to annotate these pages with concepts from WordNet, resulting in a huge annotated XML corpus with a highly heterogeneous structure and semantically rich tags. To achieve the latter, we exploit three sources of semantics: categorical information that has been added to most pages by the author (like "Albert Einstein is a physicist"), lists of similar pages, which are a common concept in Wikipedia (like lists of actors, songs, and companies), and invocations of predefined templates with human-readable parameters that encode factual information.

The rest of the paper is organized as follows. In the remainder of this section we discuss related work. In Section 2, we shortly review Wikipedia and its markup language and show how to convert Wikipedia articles to XML. In Section 3 we explain the annotation of pages based on categories and lists, and Section 4 presents an approach to exploit template invocations. Finally, Section 5 demonstrates some applications.

1.2 Related Work

There are a number of database-oriented XML benchmarks such as XMark [S$^+$02] and XMach [BR01] that focus on performance aspects of XML databases. They usually provide no meaningful content and are therefore not suited for information retrieval. The INEX community has produced two XML benchmark collections, namely the INEX IEEE collection and the INEX Wikipedia collection [DG06], which combine huge corpora with queries, lists of relevant results, and a methodology to evaluate the quality of search results. However, these corpora do not include heterogeneous and semantically rich tags. The INEX Heterogeneous Track has started to collect different XML collections, but there are no self-explaining tags yet. A first step towards richly annotated data was made in the INEX Multimedia Track with the Lonely Planet collection with documents from the travel domain [INE05].

There have been several attemps to define a standard XML format for documents stored in Wikis, the most recent ones being the Wikipedia DTD [Wika] that proposes a set of tags similar to HTML, and [Doc] that defines XML tags for a subset of the Wiki markup. However, to the best of our knowledge, none of these covers the complete feature set of Wiki markup and is publicly available.

The Semantic Web community has recently launched a number of projects to add semantics to Wikis [Aum05a, Aum05b, BG06, Sou05, VKV$^+$06]. These important activities typically aim at adding semantic information while the pages are designed, requiring that

[2]Yet Another Wikipedia Annotation project

the designer of a page has an understanding of semantics. In contrast, our aproach exploits information that is already present in Wikipedia pages, without the need for any user interaction.

Information extraction from text and HTML data is an area with intensive work. Agichtein [Agi05] gives a survey of information extraction techniques with a focus on scalability in large collections with millions of documents. Approaches in the literature mostly either follow a rule-based paradigm [AGM03, CMM02, G$^+$04], or employ learning techniques and/or linguistic methods [AFG03, CS04, Cun02, E$^+$04, NP02]. Our algorithms, unsupervised and statistical in their nature, fit in the second class. The list extraction method used in Section 3.3 is similar to list extraction methods used in other information extraction systems like KnowItAll [ECD$^+$05] and SCRAP [FFT05]. Rule-based information Extraction from XML documents has been considered by Abolhassani et al. [AFG03]; in contrast, we follow a purely automatic approach. Annotated XML collections and their use for information retrieval were considered by Graupmann et al. [GSW05] and Chu-Carroll et al. [CC$^+$06]; the techniques presented in this paper perfectly integrate with these works.

2 Converting Wiki Markup to XML

2.1 Wiki Markup

With more than 1,4 million articles as of October 2006, Wikipedia[3] is the largest general purpose encyclopedia that is freely available. The content of pages in Wikipedia, like in other Wikis, is formulated in *Wiki markup*, a combination of standard HTML tagging with specific constructs for structuring, tables, links etc. There is no formal specification of the language and its semantics yet, hence we had to reverse-engineer them from help documents in Wikipedia (which do not point out some uncommon features and usages). Figure 1 shows a simple example for a document in this language. Important building blocks of the Wiki markup language include

- structuring text by defining several levels of sections with different numbers of equals signs (=),
- different levels of emphasis for text parts with different numbers of quotes (`'`),
- defining bulleted and numbered lists of different nesting depths with `*` and `#` characters,
- defining tables with rows and columns, including headers and captions,
- defining links within the Wikipedia collection by including the name of the target page in `[[ ]]`
- defining links to the Web, by including the link in `[ ]`, and

[3]`http://www.wikipedia.org/`

- including images and other media, by defining a Wiki link with an appropriate namespace like `Image:`.

```
==Introduction==
''Wiki markup'' is used in [[Wikipedia]].

==Language Components==
* tables
* lists
* and a lot more

==See also==
[http://www.wikipedia.org]
```

Figure 1: An example WikiMarkup page

In addition, Wiki markup can be arbitrarily mixed with HTML tags (which is most often used for layout purposes), and some Wiki markup symbols (like images and tables) may contain additional layout hints. Mathematic formulae can be added with a LaTeX-like language. Wiki markup also provides a template language where invocations of a template `TEMP` (in the form {`TEMP`}) are replaced with the definition of this template, possibly incuding some simple conditional expressions like if-then-else. Template invocations may also include values for parameters that then replace the parameter in the template's definition. A Wiki2HTML converter (written in PHP) generates HTML pages (that neither have semantically rich tags nor are guaranteed to be well-formed XML) from the Wiki markup input, which is then displayed on the Wikipedia Web site.

The different components of Wiki markup can be combined almost arbitrarily. However, this huge flexibility is at the same time a big problem, as authors often tend to overstrain the features. As an example, many authors make frequent use of tables for layout, resulting in a deep nesting of tables and (sub-)sections. Secondly, the fault-tolerant converter does not encourage correct markup; while this is adequate to create HTML, this makes it difficult to create well-formed XML.

2.2 Generating XML

This section shows how we generate XML from the Wiki markup of the Wikipedia pages. We focus on the content of the pages, accepting that some layout information is lost in this process. The complete textual content of Wikipedia is available as a huge XML file (as of April 2006, this was about 6GB), which contains one element for each Wikipedia page with some meta information and its content in Wiki markup. Additionally, a large fraction of the referenced images are also available for download.

Our Wiki2XML converter runs in two phases, where each phase corresponds to a SAX-style scan of the input XML document. In the first phase, the existing pages are collected and redirections (i.e., pages that are simply links to other pages) are resolved. In the second phase, an XML document is generated for each page that is encountered. The file name of this document is formed by (1) replacing all white spaces by '+', (2) replacing all non-ASCII characters by their UTF-8-encoding, and (3) adding a '$' character after each capital letter in the page name but the first one, which is always a capital. This document is put into a folder whose name is derived from the page's name (the first two characters). As an example, the XML document for the page 'Heinrich Böll' has the file name `Heinrich+B$=C3=B6ll.xml` and is put into the directory `He`.

As the original Wiki2HTML converter is quite tolerant to syntax errors, there is a substantial number of Wikipedia pages with syntactic problems (the Wiki Syntax Project[Wikb] lists 8,400 for the April 21, 2005 dump, including 50 defective section headings, 80 HTML tags, 250 tables, 600 double quotes, 950 triple quotes, and 3400 square brackets). Each generated XML document consists of three parts: (1) the preamble that sets the character encoding and includes a pointer to an XSLT for presenting the page, (2) the `article` element with its `header` child, which in turn has children that specify meta data like the page title and id, the last revision, and the categories of this page, and (3) the `body` child of the `article` element that contains the XML representation of the page's content.

The conversion of the Wiki markup to XML is conceptually done as follows:

- Sections, subsections etc. are converted to `section`, `subsection` etc. elements, each with an `st` child that contains the section title.
- Both numbered and bulleted lists are converted to `list` elements with `entry` children corresponding to the different list entries. Nested lists are represented by `subentry`, `subsubentry` etc. tags.
- Tables are represented by a `table` tag. Inside this tag, there is one `row` element for each row of the table, which in turn has one `col` tag for each column. For header rows, the tag `header` is used.
- Links to other pages in Wikipedia are converted to `link` elements that correspond to an XLink to the target page's XML version; optionally, links to nonexisting Wikipedia pages can be marked. Links to web pages are converted to `weblink` tags with an XLink to the link target.
- Links to images are converted to `image` elements that correspond to an XLink to the image file that is located in a directory derived from the image name's MD5 hash (like in the original Wikipedia image collection).
- Markup for emphasis is converted to `b` and `it` elements.

Figure 2 shows the XML generated for the example Wiki markup from Figure 1.

The exact conversion of Wiki markup to well-formed XML is more complex (and in fact sometimes impossible without human interaction) if the markup is mixed with arbitrary HTML tags (this is also pointed out in [Doc]). This is especially a problem with some tables that are defined in a mixture of HTML and Wiki markup, but also simple tags like `<br>` render a generated document malformed. To solve this problem, we eliminate all

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="application/xml" href="../../wikipedia.xslt"?>
<article xmlns:xlink="http://www.w3.org/1999/xlink/">
  <header>
    <title>Wiki markup</title>
    <id>42</id>
    <revision>
      <timestamp>2006-10-05 14:22</timestamp>
    </revision>
    <categories>
      <category>Markup languages</category>
    </categories>
  </header>
  <body>
    <section>
      <st>Introduction</st>
      <p><b>Wiki markup</b> is used in
      <link xlink:href="../Wi/Wikipedia.xml" xlink:type="simple">
        Wikipedia
      </link>.</p>
    </section>
    <section>
      <st>Language Components</st>
      <list>
        <entry>tables</entry>
        <entry>lists</entry>
        <entry>and a lot more</entry>
      </list>
    </section>
    <section>
      <st>See also</st>
      <weblink xlink:href="http://www.wikipedia.org" xlink:type="simple">
        http://www.wikipedia.org
      </weblink>
    </section>
  </body>
</article>
```

Figure 2: Generated XML for the WikiMarkup from Figure 1

HTML tags from the Wiki markup in a preprocessing step. As those tags are typically used only for layout purposes, we do not lose any semantics, but can generate much better XML.

Each document is tested for well-formedness after it was generated, making sure that the collection contains only syntactically correct documents. In our experiments, less than 0.5% of the generated documents were malformed and therefore dropped from the collection.

The XML dialect we use is similar to the one used by the INEX Wikipedia collection. However, we have better support for some details, including nesting of sections (in INEX, all nesting levels of sections are mapped to `sec` elements). Additionally, as the INEX Wikipedia collection has kept the HTML tags, it contains a noticable amount of malformed or otherwise abnormal XML pages. However, as both collections use the same identifier for pages that is derived from Wikipedia, both collections are to some extent compatible on the document level, which allows to reuse some of the queries and assessments.

3 Semantic Annotation of Wikipedia Pages

We aim at finding high-quality semantic annotations for Wikipedia pages by a combination of exploiting manually created, high-quality information from categories that are assigned to most pages, and deriving additional information from highly structured documents such as lists (of persons, locations, etc.). To make the results applicable for a large suite of applications, we find annotations within the scope of a predefined ontology; we use WordNet [Fel98], the currently most extensive and widely used general-purpose thesaurus for the English language, but the results are transferable to any hierarchical ontology. The ontology provides us with a standard vocabulary for the annotations than can also be exploited for querying the annotated documents. Additionally, as our annotation algorithms are heuristic, we explicitly maintain an estimated confidence in the annotations wherever applicable.

3.1 Overview of WordNet

WordNet [Fel98] is a fairly comprehensive common-sense thesaurus carefully handcrafted by cognitive scientists. WordNet distinguishes between *words* as literally appearing in texts and the actual *word senses*, the concepts behind words. As of the current version 2.1, WordNet contains 81,426 synsets for 117,097 unique nouns. (WordNet also includes other types of words like verbs and adjectives, but we consider only nouns in this paper.) Often a single word has multiple senses, each of which comes with an estimation of its commonality and a brief description in a sentence or two and also characterized by a set of synonyms, words with the same sense, called *synsets* in WordNet. In this paper, we use the term *concept* for word senses, hence each concept corresponds to exactly one synset. WordNet provides relationships between concepts like hypernyms (i.e., broader senses), hyponyms (i.e., more narrow senses), and holonym (i.e., part of) relationships; for this paper, we focus on hypernym relationships.

Conceptually, the hypernym relationship in WordNet spans a directed acyclic graph with a single virtual source node 'ROOT' (that we introduced to get a connected graph) and seven first-level basic synsets (`entity`, `state`, `abstraction`, `event`, `act`, `group`, `possession`) that are children of the source node. Figure 3 shows an excerpt of that graph. For each concept in WordNet, there exists at least one, but usually several distinct root-to-concept paths (like for the concept 'singer' in the excerpt).

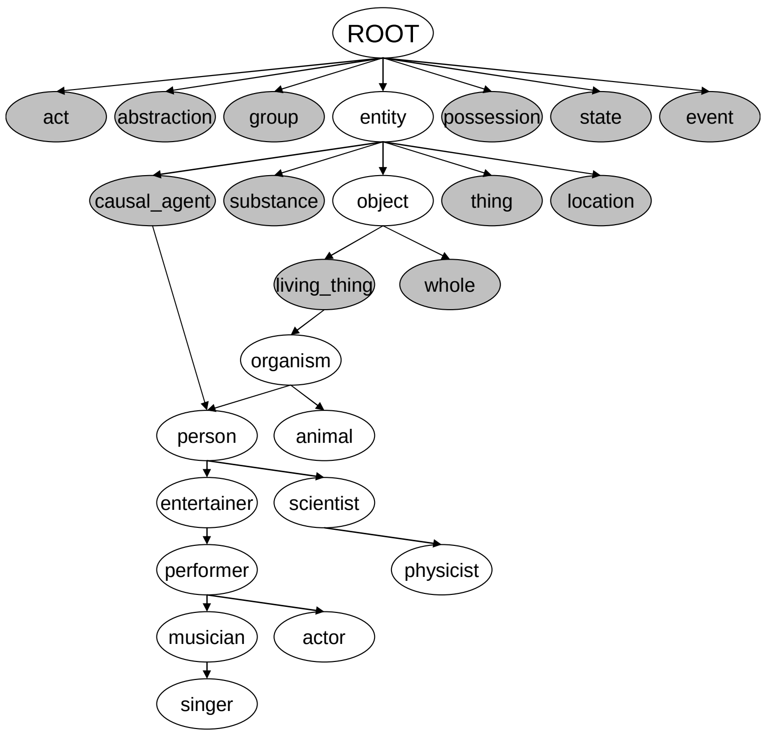

Figure 3: Excerpt of the WordNet DAG

3.2 Exploiting Categories

The majority of Wikipedia pages is assigned to one or multiple categories. The page about Albert Einstein, for example, is in the categories `German_language_philosophers`, `Swiss_physicists`, and 34 more. Not all categories, however, imply that the entity described on the Wikipedia page is an instance of some concept. Some categories serve administrative purposes (like `Articles_with_unsourced_statements`), others yield non-conceptual information (like `1879_births`) and again others indicate merely thematic vicinity (like `Physics`). The administrative and non-conceptual categories are very few (less than a dozen) and can be excluded by hand. To distinguish the conceptual categories from the thematic ones, we employ a shallow linguistic parsing of the category name. For example, a name like `Naturalized_citizens_of_the_United_States` is broken into a pre-modifier (`Naturalized`), a head (`citizens`) and a post-modifier (`of_the_United_States`). Heuristically, we found that if the head of the category name is a plural word, the category is most likely a conceptual category. We used the Pling-Stemmer [SIW06] to reliably identify and stem plural words. This gives us a (possibly empty) set of conceptual category for each Wikipedia page.

As we want to use WordNet as foundation for our annotations, every conceptual Wikipedia category has to be linked to a corresponding WordNet concept. We experimented with different heuristics, including context-aware [STW03] and compactness-based [MTV$^+$05] methods, and discovered that the simplest heuristics yields the correct link in the overwhelming majority of cases: We determine the WordNet concepts that the head of the category name refers to and link the category to the most common concept among them.

3.3 Exploiting Lists

Wikipedia contains many lists, which are an extensive, manually created and therefore high-quality source of information. In the Wikipedia Snapshot from April 2006 that we used for our experiments, there are 18,436 different lists.

We do not consider the original Wiki markup lists, but their XML versions, i.e., the output of the XML generation process described in Section 2; this allows a better handling of structure in the lists. As an example, Figure 4 shows an excerpt of the list of Germans from Wikipedia. To uniquely identify the elements of such a list, we assign a unique XPath expression to each element that consists only of name tests, child axes and position predicates. In the example, the `link` element that is a child of the second `entry` element in Figure 4; it is identified by the XPath expression `/article[1]/body[1]` `/section[1]/list[1]/entry[2]/link[1]`. Note that Wikipedia lists are not always designed as nested Wiki markup lists; there are many lists that are in fact tables. Our algorithm is not limited to Wiki markup lists and supports any type of regular structure.

It is evident that the example list is well structured, and it would be easy for a human to find out that all links point to pages about actors or, more generally, persons. A manual appraoch to exploit lists would therefore require that a user identifies patterns in a list (either explicitly by an XPath expression or by highlighting them in an interface) and assigns them to a WordNet concept like 'actor'. However, while such a manual approach could be useful for selected small but important subsets of all lists (like annotating all persons, which can be found in a list of 692 similarly structured lists), this tedious task does not scale to the whole Wikipedia collection.

3.3.1 Automatic Grouping of XPath Expressions

We propose an automated algorithm that exploits the fact that many pages have already been annotated with concepts derived from their categories, and only some pages are left to annotate. The algorithm, a variant of previously proposed algorithms for list extraction like [FFT05], proceeds in three steps: (1) it identifies parts of the list that are structurally similar (so-called *group candidates*), (2) it selects those group candidates where a large fraction of links points to pages with coherent annotations (so-called *groups*), and (3) it finds annotions that are common among them, and heuristically assigns these annotations to all pages in the group. In the example, if all but the third link point to pages that are

```
<article>
  ...
  <body>
    <section>
      <st>Actors</st>
      <list>
        <entry>
          <link xlink:href="../Ma/Mario+A$dorf.xml">
          Mario Adorf</link>, (born 1930), actor
        </entry>
        <entry>
          <link xlink:href="../Ha/Hans+A$lbers.xml">
          Hans Albers</link>, (1891-1960), actor
        </entry>
        <entry>
          <link xlink:href="../Mo/Moritz+B$leibtreu.xml">
          Moritz Bleibtreu</link>, (born 1971), actor
        </entry>
  ...
```

Figure 4: Excerpt from the XMLified Wikipedia list of Germans

labeled as 'actor', the algorithm would assign that concept to the page of 'Moritz Bleibtreu' as well.

In a preprocessing step, we first temporarily extend the annotations of all pages that are linked in the list. For each concept annotated to a page, we add all concepts on the root-to-concept paths of this concept in WordNet to the annotations of a page. This allows us to identify different annotations that are similar at a higher level of abstraction in WordNet, like 'actor' and 'singer', which are both subconcepts of 'performer'.

Group candidates are identified by grouping elements with similar XPaths together. We say that two elements have a similar XPath if both paths have the tag sequence and differ only in a single position. We label each group candidate with an XPath pattern that has the same tag sequence and the same positions as the elements in the group candidate, but a wildcard '*' at the position where the elements differ. As we maintain annotations only for pages (which are identified by `link` elements in the lists), it is sufficient to consider only group candidates where the last tag is `link`. We identify each XPath in a group candidate with the page its link element points to. In the example, the group candidate including the three `link` elements has the label `/article[1]/body[1]/section[1] /list[1]/entry[*]/link[1]`. We eliminate group candidates that are too small, i.e., consist of less than 5 elements.

To determine if a group candidate with n elements is a proper group, we count, for each concept c, the number f_c of times c where occurs as an annotation in the group. We accept the group if there is at least one concept where $\frac{f_c}{n} \geq \delta$, i.e., where the concept occurs in the annotations of at least a (configurable) fraction of all pages. Note that n includes pages without any annotations. Setting δ close to 1.0 gives a higher annotation quality, but can potentially reduce recall, whereas setting δ close to 0 incurs a high danger of wrong annotations. In our experiments, $\delta = 0.75$ yielded good results. Each such concept c is then assigned to all pages in the group that are not already annotated with c; the confidence of the annotation is set to $\frac{f_c}{n}$.

As a result of this process, we have for each page a set of candidate annotations. However, there are some redundant annotations in this set, i.e., concepts that are more general than another concept in the set (like 'actor' and 'person'). To minimize the set of annotations, we consider the subgraph of WordNet formed by the concepts assigned to this page. We keep only concepts that are sinks of the graph, removing more general concepts that are implied by them.

3.3.2 Outlier Detection

Even though most lists have a regular structure, sometimes outliers occur, i.e., small glitches in an otherwise perfectly regular structure. As an example, consider the excerpt of a list of songs shown in Figure 5. While, for most entries, the first link points to the singer's page and the second to the song's, this regularity is broken for one song that has two singers (shown in boldface). If we apply our algorithm in this setting, 'David Bowie' would be accidentally annotated as 'song'. Similar outliers can be caused by omitting links to pages that do not yet exist in Wikipedia (like some unknown singer) or by mistakes of the page editor.

```
<list>
  <entry>
    <link xlink:href="../Jo/John+L$ennon.xml" xlink:type="simple">
          John Lennon</link> :
    <link xlink:href="../Im/Imagine+(song).xml" xlink:type="simple">
          Imagine</link>
  </entry>
  <entry>
    <link xlink:href="../Ne/Nena.xml" xlink:type="simple">
          Nena</link> :
    <link xlink:href="../99/99+L$uftballons.xml" xlink:type="simple">
          99 Red Balloons</link>
  </entry>
  ...
  <entry>
    <link xlink:href="../Qu/Queen+(band).xml" xlink:type="simple">
          Queen</link> &
    <link xlink:href="../Da/David+B$owie.xml" xlink:type="simple">
          David Bowie</link> :
    <link xlink:href="../Un/Under+P$ressure.xml" xlink:type="simple">
          Under Pressure</link>
  </entry>
</list>
```

Figure 5: Example for an outlier in a list of songs

We apply a simple heuristics to detect such outliers that works as follows. First, we manually define the compatibility of a carefully selected set of base concepts towards the top of the WordNet DAG; these concepts are shown in grey in Figure 3. Compatible sets of base concepts are {living_thing,causal_agent,group} and {whole,thing}, all other combinations of base concepts are incompatible. Whenever a page p should be annotated with a new concept c, all base concepts $B(c)$ on paths from c toward the root node are computed and compared with the base concepts that have been assigned to p earlier in the process. The new concept is assigned to p if and only if each base concept in $B(c)$ is compatible with each already assigned base concept of p. This heuristic cleaning eliminates most erroneous annotations.

3.4 Adding Semantic Tags to Pages

Annotations characterize a Wikipedia page and therefore should be stored with the page, enabling queries of the form "find pages of singers that...". We therefore add tags that correspond to the annotations for a page right after the `article` element (see Figure 6. The tag names are derived from the WordNet synset that correspond to the annotated concept. The tags are augmented (in form of attributes) with the confidence, the ID of the WordNet concept, and the source of the annotation. In a NEXI-style query language, the example query fragment would be posed as `//singer[about(.,...)]`, exploiting the new annotation.

```
<article>
  <guitarist confidence="1.0" wordnetid="09498828" source="categories">
    <singer confidence="0.75" wordnetid="9908715" source="1 lists">
      <header>
        <title>Brian May</title>
        <id>42069</id>
        ...
```

Figure 6: Excerpt from the semantically annotated 'Brian May' page

At the same time, the annotations of a page are also an important source of information in other pages that link to the page; this can be exploited for queries like "find concerts where the band Queen played". To support this, we add the same tags also to links to the page in other pages (see Figure 7). Once we have such an annotation of links, the example query fragment could be formulated as `//concert[about(//band,``Queen'')]`, exploiting the fact that any link to the Queen page will be annotated as `band` (as opposed to links to the Queen Mary ship or Queen Elizabeth II. of England).

```
<guitarist confidence="1.0" wordnetid="09498828" source="categories">
  <singer confidence="0.75" wordnetid="9908715" source="1 lists">
    <link xlink:href="../Br/Brian+M$ay.xml" xlink:type="simple">
    Brian May</link>
  </singer>
</guitarist>
```

Figure 7: Semantically annotated version of a link to the 'Brian May' page

Note that the generated XML gets more complex with these additional tags. However, these documents will never be seen by any human user, as systems will typically convert results to a more readable format anyway (and can decide if they show the annotations to a user).

4 Exploiting Implicit Semantics of Template Invocations

A rich source of semantics that is already included in Wikipedia are template invocations. However, unlike in the approaches presented in the previous sections, we exploit template invocations not for annotating a Wikipedia page as a whole, but for annotating pieces of information on that page. Templates are often used to generate a standard layout for structured information that is common to many pages. As an example, Figure 8 shows the invocation of the `Infobox_band` template that generates a table with some standard information on a musical band; this information is provided as parameters to the template. Similar templates exist for persons, countries, companies, rivers, software, and many more.

```
{{Infobox_band |
  band_name    = Queen |
  image        = [[Image:Queen.png|250px|right]] |
  years_active = 1971 - Present |
  status       = Active |
  country      = [[United Kingdom]]
  }}
```

Figure 8: Example call of the `Infobox_band` template

For most templates, the name of a parameter is a clear indication of its semantics. We therefore exploit template invocations in a Wikipedia page to enrich the generated XML document with semantic annotations based on the template parameters. To do so, we try to map each parameter name to a WordNet concept, using the heuristics explained in Section 3.2. We then generate an element with the same name as the template. For each parameter, this element has a child element with the parameter name as name and the current parameter value as value. Figure 9 shows the XML that is generated for the example template invocation from Figure 8; note that not all parameter names could be mapped to WordNet concepts.

In contrast to the INEX Wikipedia collection that simply represents the template invocations with their parameters with a generic tag `<template>`, we believe that our approach is much more in the spirit of XML, allowing more natural XPath- and NEXI-style queries such as `//article[about(.,'band') and contains(//country,'USA') and contains(//status,'active')`. Note that such queries can also be posed and answered if a user does not exactly know the schema, by relaxing tag names and other structural query conditions [AY$^+$02, STW05, Sch02, TW02], or possibly by support of a DTD-aware graphical interface [vZBvOW06].

```
<Infobox_band>
  <band_name>Queen</band_name>
  <image confidence="1.0" wordnetid="3782824" source="template">
    <imagelink xlink:type="simple"
               xlink:href="../../images/3/32/Queen.png"/>
  </image>
  <years_active>1971 - Present</years_active>
  <status confidence="1.0" wordnetid="13131686" source="template">
    Active
  </status>
  <country confidence="1.0" wordnetid="8023668" source="template">
    <link xlink:href="../Un/United+K$ingdom.xml" xlink:type="simple">
      United Kingdom
    </link>
  </country>
</Infobox_band>
```

Figure 9: XML representation of the template call from Figure 8

5 Applications

5.1 Concept-Based Information retrieval

An important application of annotations is concept-based information retrieval. Graupmann et al. [GSW05] have shown that annotating important classes of information like persons, locations, and dates can help to improve result quality, especially precision of results. As YAWN is not limited to a few classes, but annotates with a huge set of diverse concepts from WordNet, it seems likely that these annotations can lead to further enhancements for the retrieval.

We have not yet done a throrough evaluation of the quality of annotations in general and their impact on result quality. However, to give a first impression of the use and effectiveness of annotations, we made some preliminary experiments with our XML search engine TopX [TSW05] on a small, annotated Wikipedia fragment, with NEXI-style [TS04] structured queries. We converted the first 10,000 Wikipedia documents (excluding redirections that contain only a pointer to another document) from the April 2, 2006 dump file into our XML format with semantic annotations, using the techniques presented in the sections before. The conversion failed for 49 documents, usually due to syntax problems in the input Wiki markup or unusual combinations of tables and sections (like starting a new section within a header of a table, or starting a new table within the title of a section). In our preliminary implementation, this took less than one hour on a standard notebook.

We consider three types of queries: (1) queries that exploit only the annotation of complete pages, optionally with content constraints, (2) queries that exploit the annotation of pages and links, optionally with content constraints, and (3) queries that additionally exploit annotations derived from template invocations. We collected five to ten queries of each type and compared the performance of TopX with annotation-aware queries to queries without annotation-awareness, i.e., content-only queries. Table 1 summarizes the results for the average precision@10 of the three query classes with and without annotation awareness. Note that we did not measure recall; we expect that some results will not be found due to errors in the annotation process. We now discuss some anecdotical results for the three query types.

Query type	Precision@10[keywords]	Precision@10[annotations]
1 (pages)	0.85	0.24
2 (pages+links)	0.96	0.14
3 (pages+links+templates)	1.0	0.0

Table 1: Experimental results with TopX

The simplest (albeit useful) example for the first query type are list-style queries like 'find scientists that won the Nobel prize' that would be formulated as keyword query `scientist nobel prize` without annotation awareness. If we exploit annotations, we can reformulate the query as `//scientist[about(.,Nobel prize)]`, yielding a lot less nonrelevant results. For the example collection, TopX achieves a precision@10 of 0.2 for the keyword query, compared to 0.8 for the annotation-aware query. Here, the additional annotation serves as a prefilter for pages that simply mention the Nobel prize, but are not a about scientists (like the page about Jimmy Carter who won the peace Nobel prize).

A typical example that shows the usefulness of the second query type that exploits page and link annotations is the query that asks for musicians who have performed a song where 'space' occurs in the title, which could be formulated as `//musician[about(//song,space)]`. Without the annotation of the link, a search engine would find any occurrence of the term within the page, not only in the context of a song. As a result, the precision of the annotation query is perfect (1.0), whereas the corresponding keyword query finds no relevant results within the top 10.

The last query type is most powerful as it can exploit all three types of annotations. As an example, consider the query 'find mayors of towns in the (German state) Hesse'. While a keyword-based query has no chance to retrieve relevant results, the annotation-aware query `//town[about(//state,Hesse)]//mayor` that exploits tags introduced by the template `Infobox_Town_DE` yields only relevant results in the test collection. However, as not all city pages may use such a template, the recall is probably not perfect. To increase recall (at the cost of precision), using ontological tag expansion and structural similarity measures in the engine could help; this is subject of future work.

5.2 Other Applications

There are many more applications for an annotated XML corpus beyond the obvious information retrieval case sketched in the previous subsection, for example to help with clustering and classification of Wikipedia pages.

The annotations can further be exploited for query formulation, by letting a user pick concepts from WordNet and combine them to form a structured query, similar to the DTD-aware query formulation presented in [vZBvOW06]. The diverse structure introduced by the annotations will most certainly be a rich source for structural feedback [ST06] that exploits the structure of relevance results to generate more precise queries. And, finally, being the first real-life heterogeneous XML collection with both rich tags and content, we think that our new collection can serve as a large-scale benchmark for systems that exploit semantic annotation for retrieval, classification, clustering, etc, extending existing benchmarks with structural diversity.

6 Conclusions and Outlook

This paper presented YAWN, a project to create an XML version of Wikipedia with semantic information. We showed how to extract semantics from categories, lists, and template invocations, yielding a huge XML corpus annotated with semantically rich tags.

For future work, we plan to extensively evaluate the quality of the annotations and their effect for information retrieval, possibly within the INEX benchmark, and to consider some of the applications sketched in the previous section. Besides that, we will examine how we can integrate the collected information into our ontology and exploit it for other data sources, too. Finally, we plan to offer the annotated XML collection for public download.

References

[AFG03] Mohammad Abolhassani, Norbert Fuhr, and Norbert Gövert. Information Extraction and Automatic Markup for XML Documents. In Blanken et al. [BGS+03], pages 159–174.

[Agi05] Eugene Agichtein. Scaling Information Extraction to Large Document Collections. *IEEE Data Eng. Bull.*, 28(4):3–10, 2005.

[AGM03] A. Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In *SIGMOD 2003*, pages 337–348, 2003.

[Aum05a] D. Aumüller. Semantic Authoring and Retrieval in a Wiki. In *European Semantic Web Conference ESWC2005*, 2005.

[Aum05b] D. Aumüller. SHAWN: Structure Helps a Wiki Navigate. In *BTW-Workshop "WebDB Meets IR"*, 2005.

[AY+02] Sihem Amer-Yahia et al. Tree Pattern Relaxation. In *EDBT 2002*, pages 496–513, 2002.

[BG06] M. Buffa and F. Gandon. SweetWiki: Semantic Web Enabled Technologies in Wiki. In *ACM 2006 International Symposium on Wikis*, pages 69–78, 2006.

[BGS+03] Henk M. Blanken, Torsten Grabs, Hans-Jörg Schek, Ralf Schenkel, and Gerhard Weikum, editors. *Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *Lecture Notes in Computer Science*. Springer, 2003.

[BR01] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *BTW 2001*, pages 264–273, 2001.

[CC$^+$06] Jennifer Chu-Carroll et al. Semantic search via XML fragments: a high-precision approach to IR. In *SIGIR 2006*, pages 445–452, 2006.

[CMM02] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites. In *SIGMOD 2002*, page 624, 2002.

[CS04] William W. Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *KDD 2004*, pages 89–98, 2004.

[Cun02] H. Cunningham. GATE, a General Architecture for Text Engineering. *Comput. Humanit.*, 36:223–254, 2002.

[DG06] Ludovic Denoyer and Patrick Gallinari. The Wikipedia XML Corpus. *to appear in SIGIR Forum*, 2006.

[Doc] DocBook XML Export. `http://meta.wikimedia.org/wiki/DocBook_XML_export` (last change JUN-19-06).

[E$^+$04] Oren Etzioni et al. Web-scale information extraction in KnowItAll (preliminary results). In *WWW 2004*, pages 100–110, 2004.

[ECD$^+$05] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[Fel98] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[FFT05] Bettina Fazzinga, Sergio Flesca, and Andrea Tagarelli. Learning Robust Web Wrappers. In *DEXA 2005*, pages 736–745, 2005.

[FLMK06] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Gabriella Kazai, editors. *4th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2005)*, volume 3977 of *Lecture Notes in Computer Science*. Springer, 2006.

[G$^+$04] Georg Gottlob et al. The Lixto Data Extraction Project – Back and Forth between Theory and Practice. In *PODS 2004*, pages 1–12, 2004.

[GSW05] Jens Graupmann, Ralf Schenkel, and Gerhard Weikum. The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In *VLDB 2005*, pages 529–540, 2005.

[INE05] INEX Multimedia Track, 2005. `http://inex.is.informatik.uni-duisburg.de/2005/tracks/media/index.html`.

[INE06] Initiative for the Evaluation of XML Retrieval (INEX), 2006. `http://inex.is.informatik.uni-duisburg.de/2006/`.

[MTV$^+$05] Dimitrios Mavroeidis, George Tsatsaronis, Michalis Vazirgiannis, Martin Theobald, and Gerhard Weikum. Word Sense Disambiguation for Exploiting Hierarchical Thesauri in Text Classification. In *PKDD 2005*, pages 181–192, 2005.

[NP02] G. Neumann and J. Piskorski. A Shallow Text Processing Core Engine. *Journal of Computational Intelligence*, 18(3):456–576, 2002.

[S$^+$02] A. Schmidt et al. XMark: A Benchmark for XML Data Management. In *VLDB 2002*, pages 974–985, 2002.

[Sch02] Torsten Schlieder. Schema-Driven Evaluation of Approximate Tree-Pattern Queries. In *EDBT 2002*, pages 514–532, 2002.

[SIW06] Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum. LEILA: Learning to Extract Information by Linguistic Analysis. In *2nd Workshop on Ontology Population (OLP2) at ACL/COLING*, 2006.

[Sou05] A. Souzis. Building a Semantic Wiki. *IEEE Intelligent Systems*, 20:87–91, 2005.

[ST06] Ralf Schenkel and Martin Theobald. Structural Feedback for Keyword-Based XML Retrieval. In *ECIR 2006*, pages 326–337, 2006.

[STW03] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Ontology-Enabled XML Search. In Blanken et al. [BGS$^+$03], pages 119–131.

[STW05] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Semantic Similarity Search on Semistructured Data with the XXL Search Engine. *Information Retrieval*, 8(4):521–545, December 2005.

[The03] Anja Theobald. An Ontology for Domain-Oriented Semantic Similarity Search on XML Data. In *BTW 2003*, 2003.

[TS04] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *INEX Workshop 2004*, pages 16–40, 2004.

[TSW05] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An Efficient and Versatile Query Engine for TopX Search. In *VLDB 2005*, pages 625–636, 2005.

[TW02] Anja Theobald and Gerhard Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *EDBT 2002*, pages 477–495, 2002.

[VKV$^+$06] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic Wikipedia. In *WWW*, pages 585–594, 2006.

[vZBvOW06] Roelof van Zwol, Jeroen Baas, Herre van Oostendorp, and Frans Wiering. Bricks: The Building Blocks to Tackle Query Formulation in Structured Document Retrieval. In *ECIR 2006*, pages 314–325, 2006.

[Wika] Wikipedia DTD (draft). `http://meta.wikimedia.org/wiki/Wikipedia_DTD` (last change APR-09-06).

[Wikb] Wiki Syntax Project. `http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Wiki_Syntax`.