WrapIt: Automated Integration of Web Databases with Extensional Overlaps

Mattis Neiling*, Markus Schaal*, and Martin Schumann

Free University of Berlin, Department of Economics,
Institute for Information Systems, Garystraße 21, D-14195 Berlin, Germany
mneiling@wiwiss.fu-berlin.de, schaal@cs.tu-berlin.de,
info@schumannsoftware.de

Abstract. The world wide web does not longer consist of static web pages. Instead, more and more web pages are created dynamically from user request and database content. Conventional search engines do not consider these dynamic pages, as user input cannot be simulated, thus providing often insufficient results.
A new approach for online integration of web databases will be presented in this paper. Providing only one sample HTML result page for a source, result pages for new requests will be found by structural recognition. Once structural recognition is established for one source, other web databases of the same universe (e.g. movie databases) can be integrated on the fly by content-based recognition. Thus, the user receives results from various sources.
Global schemata will not be produced at all. Instead, the heterogeneity of the single sources will be preserved. The only requirement is given by the existence of an extensional overlap of the databases.

1 Introduction

The world wide web is increasing at an enormous rate. Already now it can be viewed as the largest source of information. According to Najork and Viennese [MN01], the following estimation applied in October 2000: 2,5 billion pages were directly accessible and 550 billion pages were dynamically generated. Especially, more and more databases are made available via the web. In order to access information from these web databases, queries are entered into certain input forms (text boxes, check boxes, etc.). The query result is returned by dynamically generated *result pages*. Users need to access each source individually in order to formulate appropriate queries and to identify the results.

* Part of this work was supported by the Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316)

Automated searching across several web databases requires knowledge about the structure of each individual site. Nowadays, individual wrappers are developed for each source separately. This solution is not very satisfying, as new sources appear every day and legacy sources may change their layout and programming style. Instead of creating one wrapper per source, a spider is used for querying sources. Following this wrapper-free approach, a method for recognition of result pages is required. Related work (e.g. Crescenci et. al. [CMM01]) has proposed the use of several (at least two) sample *result* pages per source in order to recognize other result pages. Our approach extends this idea to multiple sources, requiring only *one* sample page for *one* source.

Our idea is as follows: The structure of result pages of the first source can be derived by comparing the sample page with new query results (so-called *structural recognition*). Once the structure of the first source is known, domain-specific content can be extracted from this source. Then, the same query is sent to another source and result pages are recognized there by content-based matching (so-called *content-based recognition*).

Thus, wrapper-free search can be extended towards automated integration of previously unknown or changing web databases.

2 State of the Art

Semi-automated wrapper generation has been recognized as challenge in the scientific community. At least two approaches exist, namely Lixto (cf. Baumgartner et. al. [BFG01]) and W4F (World Wide Web Wrapper Factory, cf. Sahuguet et. al. [SA99]).

Lixto generates Wrappers by user interaction. A graphical user interface (Interactive Pattern builder) enables the user to select text parts within HTML pages. Each text part is an example for a kind of data which can be extracted. Lixto represents the HTML page internally as a tree where text parts correspond to partial trees. Additional knowledge has to be provided by the user in terms of so-called Lixto or Elog rules for the extraction of the actual data.

As it is the case for Lixto, W4F also uses a query language (HEL, HTML Extraction Language) for producing extraction rules. From these rules JAVA source code is produced. This source code is the actual wrapper and contains not only the extraction methods, but also methods for communication with the web server.

Besides semi-automated wrapper generation, there is also an approach for automated wrapper generation, namely Roadrunner (cf. Crescenci et. al. [CMM01]). Very similar to the approach described here, Roadrunner starts out with two sample result pages and analyzes the structure of such pages by parsing the HTML pages in parallel. Identical elements are ignored. Different text strings are used in order to localize content, while different HTML tags are used in order to analyze optional or repeating structures. The Roadrunner approach leads to a grammar for result pages.

While our approach builds upon the concepts realized in Roadrunner, we have gone one step further by extending the search to other sources without having previous knowledge except for their Unified Resource Locator (URL).

3 Our Approach

Content in the world wide web is provided by different web sites in a variety of different kinds. The results and the communication steps for achieving these results vary strongly. Some web sites may answer a query directly with the result, others show a summary first. The automated distinction between the *result pages* (containing the details) and other pages (such as summaries, etc.) is our core issue. The recognition of *result pages* is a central issue for automated searching of previously unknown web databases.

The recognition of result pages requires knowledge about the structure and/or content of such a page. We have used two different methods for two different situations:

Structural Recognition: In cases, where the source is already known and structure is given by sample result pages, we follow the Roadrunner-approach [CMM01] and perform a search for pages of similar structure. For a given user request, the result pages are those, which have a similar structure. However, we do not compute a grammar for result pages, but generate so-called Content-Tagged Structure Trees (CTSTs)

Content-based Recognition: In cases, where the source is not previously known, a user request will also be given to the previously known sources. Similar results (relating to the same real-world object) may then be recognized by comparing the contents of the result pages of the known source with all candidates from the previously unknown source. Once a result page has been recognized by this method, it can be further used as a sample page for structural recognition.

Content-based recognition requires an extensional overlap between a known source and an unknown source. Moreover, the user request must produce results from the extensional overlap. Therefore, we distinguish between administrative and user level use of our prototype *WrapIt*. Content-based recognition is reserved for the administrative level only where experts are required to enter requests that are likely to refer to the extensional overlap. For user level use, knowledge about extensional overlaps is not required and pages are structurally recognized by comparison with sample pages.

Figure 1 depicts the complete approach as an abstract data flow diagram. Web site A (left side) is assumed to be previously known, while web site B is the unknown site. In both cases, a simple spider is used for the generation of HTML candidate pages. The spider tries to enter the user request pattern Y into appropriate form fields and follows links.

For structural recognition, the candidate pages are matched pairwise with the sample result page, leading to so-called Content-Tagged Structure Trees

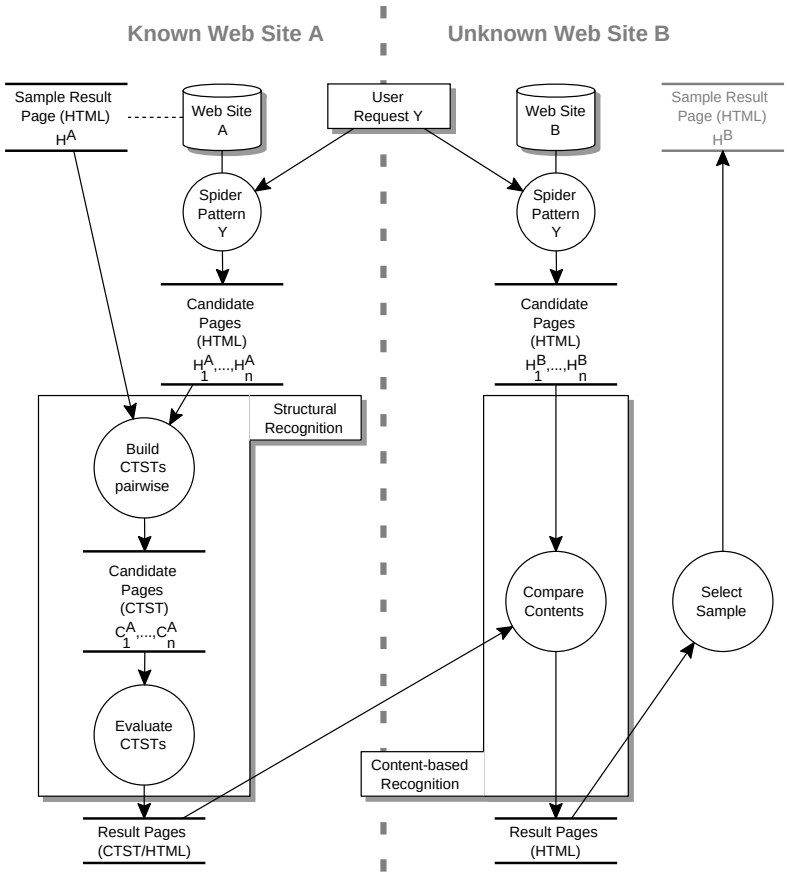


Fig. 1. Abstract Data Flow Diagram of *WrapIt*; the concept of the CTST's, the Content-Tagged Structure Trees, is introduced in section 3 and further discussed in section 4.

(CTSTs). This matching is performed upon the DOM-tree[1] and can be rather sophisticated as repetitive and optional structures have to be recognized. Details will be discussed in Section 4.

The CTSTs are subsequently evaluated and ranked heuristically with respect to a so-called S-measure, which gives high ranking for candidate pages who have a similar structure but textual distinctions. CTSTs with sufficiently high ranking are considered being result pages and the textual distinctions are stored as term list for content-based recognition. Details will be discussed in Section 5.

For content-based recognition, the HTML candidate pages (of web site B) are directly searched for the previously stored content-tags with usual information

[1] Document Object Model for HTML according to `http://www.w3c.org/dom`

retrieval techniques. The closest results are considered being result pages for web site B. Details will be discussed in Section 6.

4 Structural Matching

The structural analysis is used for the recognition of result pages when a sample page is already known. For any pair of HTML sample page H^A and HTML candidate page H_i^A the respective DOM-trees D^A and D_i^A are built first. The DOM-tree represents the hierarchical tree-structure of HTML source code explicitly. Metatags are ignored and the root is always the `body`-Tag. Textual information is always located in the leaves of the tree. The path to the leaves represents the formatting.

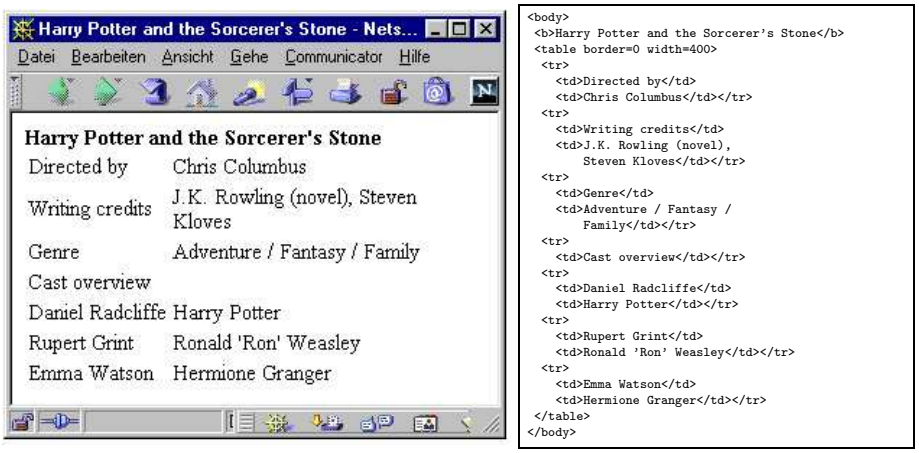


Fig. 2. Screen dump of a sample page H^A of the movie *Harry Potter and the Sorcerer's Stone* (left) and the corresponding HTML-code (right)

The Content-Tagged Structure Tree (CTST) for such a DOM-tree D_i^A with respect to a referential DOM-tree D^A will be generated as follows. By sequential pairwise comparison of these paths the leaves of D_i^A are tagged with respect to the following cases:

1. The path is present in both trees and the leave content is identical. Such leaves are marked with a DELETION tag. Most likely, they represent irrelevant information that appears on any instance of a result page at this web site (like ads, headings, non-specific attributes).
2. The path is present in both trees, but the leave content is different. Such leaves are marked with a CONTENT tag. Most likely, they represent actual content being retrieved from the database upon user request.

3. The path is present only in D_i^A. Such paths are marked with an OPTIONAL tag.
4. The path represents a repeated structure. Such paths are marked with an ITERATION tag.




```
<body>
 <b>The Matrix</b>
 <table border=0 width=500>
  <tr>
    <td>Directed by</td>
    <td>Andy Wachowski,
        Larry Wachowski</td></tr>
  <tr>
    <td>Writing credits</td>
    <td>Andy Wachowski
        & Larry Wachowski</td></tr>
  <tr>
    <td>Genre</td>
    <td>Action / Thriller /
        Sci-Fi</td></tr>
  <tr>
    <td>Cast overview</td></tr>
  <tr>
    <td>Keanu Reeves</td>
    <td>Thomas A. Anderson/Neo</td></tr>
  <tr>
    <td>Laurence Fishburne</td>
    <td>Morpheus</td></tr>
  <tr>
    <td>Carrie-Anne Moss</td>
    <td>Trinity</td></tr>
 </table>
 <br>
 <U>Oscar Award Oscar 2000, Best Effects</u>
</body>
```

Fig. 3. Screen dump of a candidate page H_1^A of the movie *Matrix* (left) and the corresponding HTML-code belonging (right)

A small example will illustrate the process of tree matching as described here.

Example 1. Starting with two simple HTML pages H^A and H_1^B about movies as depicted in Figures 2 and 3, the HTML code is transformed into standard DOM-trees D^A and D_1^B.

The structure of these DOM-trees is derived from the structure of the HTML code of as follows:

A node is created for each pair of HTML-tags like `<b>...<\b>`, `<tr>...<\tr>`, or `<td>...<\td>`.

Further nodes lying between these tags are child nodes with respect to the enclosing node, i.e. all that is between these tags is placed at succeeding nodes respectively. Textual information (the content) is always placed at terminal nodes which will be called leave nodes also.

Structure recognition (and CTST creation) is now performed on DOM-trees. We compare the paths to the leaves between H^A and H_i^A as follows:

If a path is present in both trees with identical content, then there is no distinguishing information given by that. Lets consider path

`<body>/<table>/<tr>/<td>`

with leave content

```
Directed by
```

This node is to be tagged with a DELETION tag. But if only the path matches properly and the content is different as it is the case for leave nodes

```
<body>/<b>Harry Potter ...
<body>/<b>Matrix
```

respectively, an identical structural element with different content has been found. Such a leave node is then to be marked with a CONTENT tag.

If paths to leave nodes occur only in one of both DOM-trees, the node is tagged with an OPTIONAL tag, as in the case of

```
<body>/<U>Oscar Award Oscar 2000, Best Effects
```

in H_i^A. Last but not least, repeated structures are recognized as well. Paths occurring at least twice within the DOM-tree are marked with the REPETITION tag. Note, that such paths may not necessarily lead to leave nodes, as it is the case for the path

```
<body>/<table>/<tr>
```

in both DOM-trees.

Both result pages are displayed as screen dumps and HTML code in the Figures 2 and 3. The resulting CTST-tree is shown in Figure 4 as screen dump.

Intuitive pictograms have been employed for the visualization of the different tags. The A-symbol represents the CONTENT tag, the *scissors*-symbol represents the deletion tag, etc. .

Intuitive pictograms have been employed for the visualization of the different tags of the CTST (compare fig 4):

- the **A**-symbol represents the CONTENT tag,
- the *scissors*-symbol represents the DELETION tag,
- the *balloon with a question mark* represents the OPTIONAL tag, and
- the *rotation*-symbol represents the REPETITION tag.

5 CTST evaluation

Each CTST C_i^A will be evaluated with the heuristic measure S :C$\rightarrow$ $\mathbf{R}$, where C is the space of all CTSTs with

$$S(C_i^A) = \#ContentTag - \#OptionalTag \tag{1}$$

Numerous other measures can be thought of. This measure has been experimentally shown to effectively distinguish between result pages and other pages by choosing 0 as treshold value, i.e.

$S(C_i^A) > 0$**:** Most likely, the candidate page is a result page.

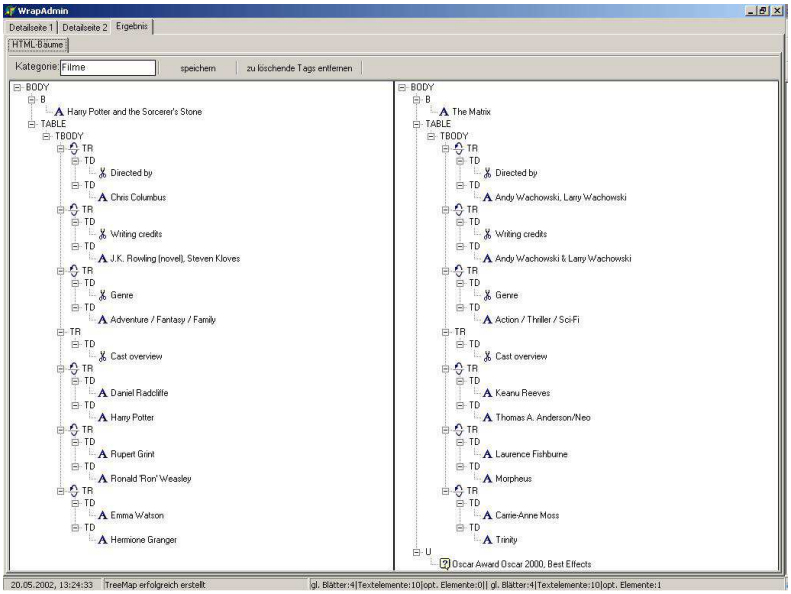


Fig. 4. Screen dump of the resulting CTST's of the HTML-pages of fig. 2 (left hand side) and fig. 3 (right hand side)

$S(C_i^A) = 0$**:** Most likely, both pages are identical.
$S(C_i^A) < 0$**:** Most likely, the candidate page is no result page.

Actually, we have not found values near to zero (in most cases[2]). Further experiments will show whether this measure together with threshold value 0 is indeed a good classifier for result pages.

6 Content Matching

In order to recognize result pages from unknown web sites, the content has to be analyzed. For a specific query Y the result pages for known web sites are computed first by structural analysis. Similar results from known and unknown sites are likely to refer to the same real world object.

In a pre-processing step, term lists are to be retrieved from both pages whose similarity is to be measured. The measurement of similarity is then performed upon these term lists instead of the original page. The method for term list extraction is described next:

[2] Regrettably there is also a counter example which has been documented in the result section (cf. Example 3 in Section 8).

For known sources: Content for the term lists is taken from those leaves in the CTST that are labelled with the CONTENT or OPTIONAL tag respectively. The term list contains all words contained in this leaves and some phrases

For unknown sources: In this case, the actual content is not localized. The term list will be generated from all leave nodes of the DOM-tree.

After the pre-processing step, a term list L_k^A is retrieved from some result page H_k^A of known web site A (using the CONTENT and OPTIONAL tags in the CTST) and a second term list L_j^B is retrieved from some candidate page H_j^B of unknown web site B (using the DOM-tree). The frequency of each term within the respective page is stored in addition to the term itself.

For measuring similarity, we adopt the well-known *vector space model* for *documents* [Sal89]. In the *vector space model* documents consist of terms, e.g. words or phrases. The similaririty of two term lists can be computed with the TF–IDF weighting scheme. TF–IDF abbreviates Term Frequency– Inverse Document Frequency, a widely used method for comparing text documents. W. Cohen applies this approach successfully to the mediation of web data sources cf. [Coh98].

In the vector space model, a *document vector* v consists of nonnegative real numbers, the i-th component of v corresponds to the i-th term t_i of a term enumeration. In our approach, the *documents* d are given by the result pages H^A, H_k^A for the web database A and the result pages H^B, H_k^B for the web database B, respectively. The value of $v(i)$ corresponding to a term t_i is defined for a result page d by

$$v(i) = v_d(i) = \begin{cases} 0 & : \quad t_i \notin d \\ (1 + \log TF(d,i)) \cdot \log IDF(i) & : \quad t_i \in d \end{cases} \tag{2}$$

where the $TF(d,i)$ denotes *term frequency* of the i-th term t_i in the result page d. $IDF(i)$ is the relative *inverse document frequency* of the i-th term t_i within a given collection DOC of result pages. $IDF(i)$ is defined as follows

$$IDF(i) = \frac{|DOC|}{|\{d \in DOC \mid t_i \in d\}|} \,. \tag{3}$$

The similarity of two documents is defined by the inner product of the vector space:

$$Sim(d_1, d_2) = v_{d_1}^T v_{d_2} = \sum_i v_{d_1}(i) \cdot v_{d_2}(i) \tag{4}$$

The similarity of two documents gets the higher the more terms coincide. Rare terms t being in both documents d_1, d_2 lead to high values for the corresponding component of $v(i)$. By this, rare terms are heavy indicators for similarity, while widely used terms receive only little weight.

For efficient computation of $IDF(t)$, we use a domain-specific dictionary DIC instead of DOC. DIC is generated from the known web sites by posing queries with requests from a predefined list. The dictionary DIC is then build from the contents of all result pages that have been found. An entry in DIC consists of a term t_i and its cumulated frequency $DIC(i)$ in the assessed result

pages DOC. If the number of the assessed result pages $N = |DOC|$ is stored separately, the formula (3) becomes

$$IDF(i) = \frac{N}{DIC(i)}. \tag{5}$$

For the calculation of the similarity between two HTML-pages H_k^A, H_j^B only terms t_i occurring both in L_k^A and L_j^B are considered, since either $v_A(t)$ (referring to the term list L_k^A) or $v_B(t)$ (referring to the term list L_i^B) equals zero in the other cases (cf. Equation 2). Formula (4) can thus be rewritten for efficient computation of the similarity between HTML-pages H_k^A and H_j^B as follows (with (2) and (5)):

$$\begin{aligned} &Sim(H_k^A, H_j^B) = Sim(L_k^A, L_j^B) \\ &= \sum_{i:t_i \in L_k^A \cap L_j^B} v_{L_k^A}(i) \cdot v_{L_j^B}(i) \\ &= \sum_{i:t_i \in L_k^A \cap L_j^B} \underbrace{\big(1 + \log TF(L_k^A, i)\big)}_{\text{value for } t_i \text{ in } L_k^A} \cdot \underbrace{\big(1 + \log TF(L_j^B, i)\big)}_{\text{value for } t_i \text{ in } L_j^B} \cdot \underbrace{\left(\log \frac{N}{DIC(i)}\right)^2}_{IDF\text{–weight for } t_i} \end{aligned} \tag{6}$$

where $TF(L_j^B, i)$ denotes the frequency of the i-th term t_i in term list L_j^B. In this way we compute the similarity directly from the term lists, without explicit use of the vector representation as described above. This procedure has a complexity of $O(n)$, where n denotes the (maximal) size of the term lists (number of terms). However, typically n becomes not large for result pages (approximately 100 terms).

By employing the TF–IDF model, candidate pages from unknown sources can be ranked by their similarity to the result pages of the known web site. Lets assume, that candidate page H_j^B (unknown source) has the highest similarity value when compared to some H_k^A (known source). All other pairs of candidate and result pages have lower values. Then, both pages are assumed to refer to the same real-world object. Result page H_j^B (unknown source) is stored as new sample page H^B such that web database B is a known source from now on.

7 Remarks on the Spider

User requests are usually sent to the web server via forms. The request is formulated by text entered in one or several text boxes and by choices from predefined lists. Gratifyingly, most web databases offer quick search requiring input for only one text box. However, we have only studied web databases were the user query can be entered in the first text field of the form.

Except for following links, this is the only task for the spider to be performed. A more sophisticated approach to the analysis of forms is given by Raghavan and Garcia-Molina [RGM01].

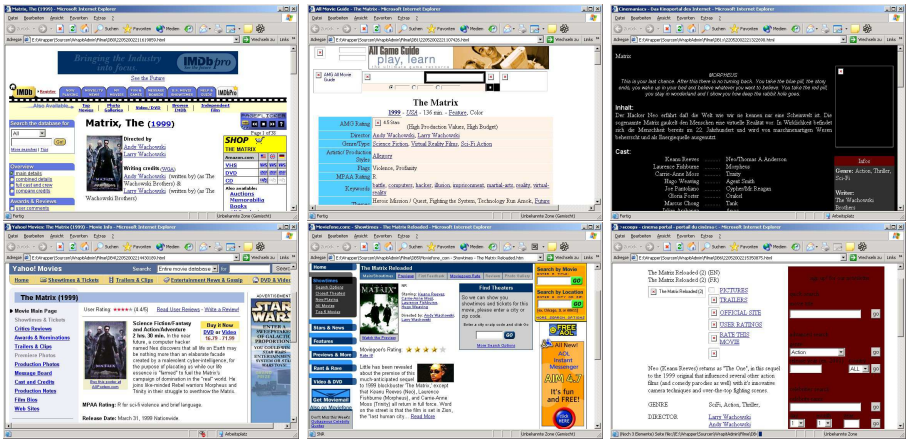


Fig. 5. Thumbnails of the result pages for the request *matrix* from 6 databases, as shown in table 1. It can be seen, that layout and structure differ.

8 Experimental Results

Our prototype *WrapIt* is implemented as standalone application with Borland Delphi 5 and uses classes shipped with the Microsoft Internet Explorer 6.0, namely the library `mshtml.dll`. We use these classes to display HTML-files and for the spider querying the web database and crawling through the results.

All tests presented here have been carried out on a PC equipped with the processor AMD Athlon 1.7XP+ (1.433 GHz) with 512MB RAM on Windows XP Professional. Since massive web querying is performed, the connectivity to the Internet might be of interest, too. The PC was connected to the Internet by use of ADSL with 768 Kbit/s downstream and 128 Kbit/s upstream bandwidth through a Ethernet-card.

We applied tests within two universes of discourse (or domains), namely movie databases and online book sellers. Since the prototype is implemented based upon Internet Explorer 6.0, the user can follow the querying, page loading and browsing by the program by watching the actual HTML page, thus he can trace the progress of the program. Alternatively, when performing an administrator run, a log window can be viewed, where the querying, page loading and matching results of the program are documented.

First of all we present the results for the administrative usage level, where previously unknown web databases are to be analyzed by the content-based recognition described in section 6.

Example 2 (content-based recognition). For the administrative usage level the administrator feeds one sample HTML-page from a web database A to the program (by browsing to the page and saving it with the embedded IE 6.0), in our movie example it consists of the result page of the movie *Lord of the Rings* from `http://www.imdb.com`. Additionally, a list of URL's of movie databases is

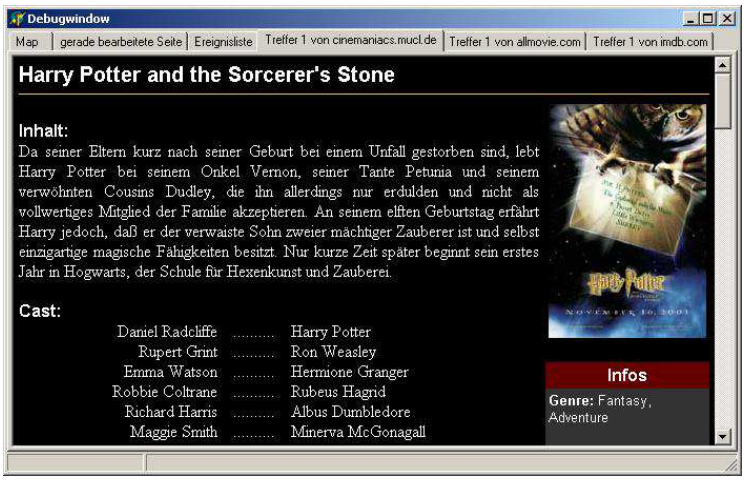


Fig. 6. Screen dump of the prototype *WrapIt*, displaying a result page for the request *potter* (Example 3)

stored in an `.ini`-file, in our case for 9 databases (including the known source `http://www.imdb.com`). The administrative user request in our example is *Matrix*, since new and Oscar-awarded movies are likely to appear in most of the movie databases.

The results are presented in table 1. We found the movie in all but one database[3] (`http://rasp.nexenservices.com`). The overall runtime was about eight minutes, since the queries were send sequentially.

Example 3 (structural recognition — movies). Once the sample HTML pages are found and stored as demonstrated in example 2, user requests can be handled by structural recognition. We present the results of two tests in Table 2 and 3. For the first test the runtime was between 70 seconds and two minutes. The S-measure of the CTSTs was a clear indicators for result pages, i.e. they were correctly identified (cf. table 2, querying of three databases). In a second test run, all seven databases were queried (cf. table 3). Here, the results were less satisfying with the S-measure below 0 even for semantically correct results in four cases. Further research should analyze the reasons for this misclassification

[3] The reason is more technical: since this web-site uses redirection, our spider was not able to load the redirected page— mostly because the I.E. 6.0 usually delivers the first and only sometimes the redirected page

[5] The highest similarity-value (according to (6)) of the result pages of the database B_i compared with the sample result page from A is reported.

[5] Structural recognition took place for `www.imdb.com`, with the objective to find a new sample page usable for the content-based recognition at other databases; The CTST of the new sample page about the movie *Matrix* had a S-Value of 48 relative to the previous given sample page about the movie *Lord of the Rings*.

Table 1. Results of the test for the administrative usage level, applying content-based recognition to get new sample HTML-pages from previously unknown movie databases (example 2)

web database (URL, without `http://`)	hits	time [s]	new sample page	Sim[4]
A: `www.imdb.com`[5]	20	79.3	The Matrix	–
B_1: `cinemaniacs.mucl.de/frames`	3	39.8	The Matrix	53.2
B_2: `rasp.nexenservices.com/index.php`	1	19.9	no page	–
B_3: `www.allmovie.com`	7	32.2	The Matrix	251.7
B_4: `www.cinema.de/suche`	5	39.4	The Matrix	29.0
B_5: `movies.yahoo.com/movies`	13	65.0	The Matrix	216.3
B_6: `movies.eonline.com`	17	87.9	Matrix News	33.1
B_7: `www.scoops.be/home.asp?lang=1`	4	48.1	The Matrix	98.3
B_8: `tvguide.com/movies`	3	52.7	The Matrix	63.2

Table 2. Results of the test for the user search, applying structural recognition on HTML-pages from 3 movie databases (example 3)

user request	web database (URL)	hits	time [s]	highest ranked result page	S-value
Amélie	www.imdb.com	12	56.7	Fabuleux destin d'Amélie Poulain, Le	49
Amélie	cinemaniacs.mucl.de	3	10.2	Le fabuleux destin d'Amélie Poulain	57
Amélie	www.allmovie.com	1	11.2	no result page	
Men in Black	www.imdb.com	12	58.9	Man In Black 2	34
Men in Black	cinemaniacs.mucl.de	3	10.2	no result page	
Men in Black	www.allmovie.com	12	11.2	Men In Black	123
Harry Potter	www.imdb.com	13	87.1	Harry Potter and the Chamber ...	61
Harry Potter	cinemaniacs.mucl.de	3	10.4	Harry Potter and the Chamber ...	78
Harry Potter	www.allmovie.com	4	19.5	Harry Potter and the Chamber ...	150

and come up with a refined S-measure. The runtime was about three and seven minutes respectively for each request.

Example 4 (structural recognition — book sellers). Content-based recognition has been successfully applied for two databases in the book sellers domain starting with a sample page about the book *Harry Potter* from `www.amazon.com` . Details are omitted here. Structural recognition can be applied based upon sample pages from both domains. Again, the response time was quite large with many HTML-pages to be loaded and analyzed (cf. table 4). The S-measure was a good classifier here, except for the case of `www.alibris.com`, where values of -1 and -2 occurred.

The performance seems to be rather unsatisfactorily — the answering times range between a half and two minutes. However, we have to perform many web

[6] We report the highest value of S, the similarity measure of structural recognition according to (1).

Table 3. Results of the second test for the user search on 7 movie databases, applying structural recognition (example 3)

user request	web database (URL)	hits	time [s]	highest ranked result page	S^6
First Contact	www.imdb.com	5	22.9	First Contact	48
First Contact	cinemaniacs.mucl.de	1	9.6	no result page	
First Contact	www.allmovie.com	9	10.7	no result page	
First Contact	movies.yahoo.com	4	25.8	Star Trek: First Contact	1.8
First Contact	movies.eonline.com	2	28.1	Star Trek: First Contact	-220
First Contact	www.scoops.be	3	38.6	Star Trek: First Contact	4.9
First Contact	www.cinema.de	1	40.6	no result page	10
Back to the Future	www.imdb.com	11	56.0	Back to the Future Part III	5.0
Back to the Future	cinemaniacs.mucl.de	1	7.9	no result page	
Back to the Future	www.allmovie.com	37	104.3	Back to the Future Part III	255
Back to the Future	movies.yahoo.com	4	25.8	Back to the Future Part II	22
Back to the Future	movies.eonline.com	10	60.3	Back to the Future	-18
Back to the Future	www.scoops.be	10	52.0	Back to the Future Part III	25
Back to the Future	www.cinema.de	1	74.5	no result page	

Table 4. Results of searching book seller databases by structural recognition (example 4)

request	URL (without http://)	hits	time [s]	highest ranked result page	S-value
A Brief History of Time	www.amazon.com	20	159.3	A Brief History of Time	36
A Brief History of Time	www.barnesandnoble.com	19	98.0	The Illustrated A Brief ...	54
A Brief History of Time	www.alibris.com	23	266.4	A Brief History of Time	-1
A New Kind of Science	www.amazon.com	12	133.8	A New Kind of Science	36
A New Kind of Science	www.barnesandnoble.com	3	24.5	A New Kind of Science	54
A New Kind of Science	www.alibris.com	0	10.2	no result page	–
First Contact	www.amazon.com	27	199.2	First Contact	58
First Contact	www.barnesandnoble.com	3	23.7	From First Contact through Reconstruction	11
First Contact	www.alibris.com	40	169.1	Star Trek: First Contact	-2

queries on numerous databases, all of these being executed in sequel by our present prototype. At this time we have not implemented parallel querying of databases, which could lead to an enormous performance gain. Another reason for the poor performance of our prototype is given by extremely complex HTML pages, typically with 500–1500 HTML-elements containing between 100–250 text parts. Thus the DOM-tree gets large and the *realtime* creation and comparison of CTSTs becomes very costly. Different strategies can be thought of in order to overcome this problem. Besides parallelization, cutting of computation during DOM- and CTST-creation at the earliest possible time might be a good method if incompatible paths are detected at an early stage.

Satisfactorily, at second, the tests on the two domains (books and movies) establish a proof of concept, as the approach works well. In fact, structural recognition was successful for most web databases, with only one sample HTML-page

given. Additionally, the detection of new sample HTML-pages from previously unknown data sources was also successful in many cases using our newly invented content-based recognition. Once a sample page was found, the database could be queried along with all the others. Hence, the database was integrated.

9 Summary

We have presented a new wrapper-free approach for the automated integration of web databases. More specifically, the recognition of result pages is done by structural recognition for known web databases and by content-based recognition for new web databases.

Structural recognition deploys the structural similarity of pages generated by the same web database. The query results are compared with a given sample page. This is done by use of their HTML structure. Content-based recognition is based upon the extensional overlap between web databases. Result pages from different web databases but referring to the same real-world object are matched by term-frequency based comparison.

For the chosen examples (movies and books) our experiments produced satisfying results. However, the approach depends heavily upon the initial sample pages, the chosen user request and, last but not least, upon the kind of web database content. We are convinced that our approach is a good foundation for further development of wrapper-free automated integration of web databases.

Last but not least, *WrapIt* can easily adapt to changes in web databases.If there is no fundamental change in the extension of the data, the evolution of web databases, including changes in communication, data modelling, structure, layout etc. can be managed automatically by our approach.

References

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Declarative information extraction, Web crawling, and recursive wrapping with lixto. *Lecture Notes in Computer Science*, 2173, 2001.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 109–118, Orlando, September 2001. Morgan Kaufmann.

[Coh98] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD*, Seattle, Washington, 1998.

[MN01] Janet L. Wiener Marc Najork. Breadth-first search crawling yields high-quality pages. In *Proceedings of Tenth International World Wide Web Conference*, Hong Kong, May 2001.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 129–138, Orlando, September 2001. Morgan Kaufmann.

[SA99] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, 1999.

[Sal89] Gerald Salton, editor. *Automatic Text Processing*. Addison-Wesley, Reading, Massachusetts, 1989.