DeepBot: A Focused Crawler for Accessing Hidden Web Content

Manuel Álvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, Víctor Carneiro
University of A Coruña - Department of Information and Communications Technologies
Facultade de Informática
Campus de Elviña s/n, A Coruña 15071, Spain
Phone: +34 981 167 000
{mad,jrs,apan,fidel,fbellas,viccar}@udc.es

ABSTRACT

The crawler engines of today cannot reach most of the information contained in the Web. A great amount of valuable information is "hidden" behind the query forms of online databases, and/or is dynamically generated by technologies such as Javascript. This portion of the web is usually known as the Deep Web or the Hidden Web. We have built DeepBot, a prototype of hidden-web focused crawler able to access such content. DeepBot receives a set of domain definitions as an input, each one describing a specific data-collecting task and automatically identifies and learns to execute queries on the forms relevant to them. In this paper we describe the techniques employed for building DeepBot and report the experimental results obtained when testing it with several real world data collection tasks.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases.

H.2.8 [**Database Management**]: Database Applications - Data mining.

H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Algorithms, Design, Experimentation.

Keywords

Crawler, Hidden Web, Web Forms.

1. INTRODUCTION

A key component in the architecture of current web search engines are the "crawler" programs used to automatically traverse the Web, retrieving pages to build a searchable index of their content. Crawlers receive as input a set of "seed" pages and recursively obtain new ones by locating and traversing their outbound links.

Conventional web crawlers cannot reach to a very significant fraction of the web, which is usually called the "Hidden Web" or the "Deep Web".

Several works have studied and characterized the Hidden Web [4], [5]. They concluded that it is substantially larger than the publicly indexable web and that it usually contains data of higher quality and with a higher degree of structure.

The problem of crawling the "Hidden Web" can be divided into two challenges:

- *Crawling the "server-side" Hidden Web*. Many websites offer query forms to access the contents of an underlying database. Conventional crawlers cannot access these pages because they do not know how to execute queries on those forms.
- *Crawling the "client-side" Hidden Web*. Many websites use techniques such as client-side scripting languages and session maintenance mechanisms. Most conventional crawlers are unable to handle this kind of pages.

This paper overviews the architecture of DeepBot, a prototype system for crawling the Hidden Web, and describes in detail the techniques it uses for accessing the content behind web forms. The techniques used to deal with the client-side Deep Web were described in greater detail in [1].

The main features of DeepBot are:

- For accessing the "server-side" Deep Web, DeepBot can be provided with a set of *domain definitions*, each one describing a certain data-gathering task. DeepBot automatically detects forms relevant to the defined tasks and executes a set of pre-defined queries on them.
- DeepBot's crawling processes are based on automated "mini web browsers", built by using browser APIs (our current implementation is based on Microsoft Internet Explorer). This enables our system to deal with client-side scripting code, session mechanisms, and other complexities related with the client-side Hidden Web.

The paper is organized as follows. Section 2 overviews the architecture of DeepBot and the main components that participate in accessing the server-side Hidden Web. Section 3 describes the domain definitions used to specify a data collection task. Section

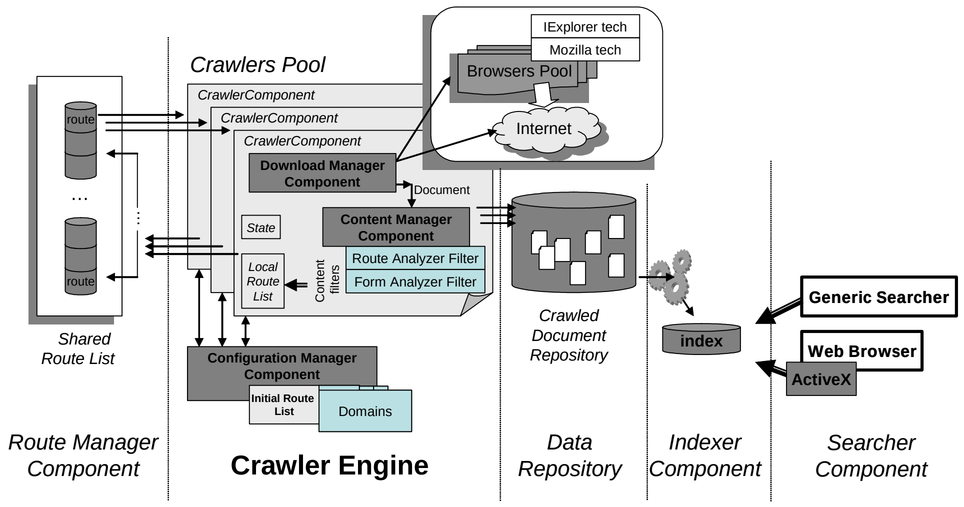


Figure 1. Crawler Architecture

4 describes how DeepBot detects query forms relevant to a certain task and how it learns to execute queries on them. Section 5 describes our experiments with the system. Section 6 discusses related work and section 7 concludes the paper.

2. ARCHITECTURE

The architecture of the system is shown in Figure 1. As well as in conventional crawlers, the functioning of DeepBot is based on a shared list of *routes* (pointers to documents), which will be accessed by a certain number of concurrent crawling processes, distributed into several machines. The main singularities of our approach are:

- In conventional crawlers, routes are just URLs. Thus, they have problems with sources using session mechanisms. Our system stores, with each route, a session object containing all the required information (cookies, etc.) to restore the execution environment in which the crawling process was running in the moment of adding the route to the master list.
- Conventional engines implement crawling processes by using HTTP clients. Instead, our system uses lightweight automated *mini web browsers* (built by using the APIs of most popular browsers) as execution environment for automated navigation. These *mini web browsers* access to pages by generating actions on a web browser interface, in the same way a human user would generate them when browsing. For specifying a navigation sequence in the automated mini-browsers, we use NSEQL [13], a language which allows representing the list of interface events a user would need to produce on the browser to reach the desired page.
- When the system reaches a new page, in addition of using its anchors to generate new routes, it also examines each HTML form and ranks its relevance with respect to a set of pre-configured *domain definitions*, each one describing a specific data-collection task. If the system finds that the form is relevant, it is used to execute a set of queries defined by the domain, thus reaching to new pages.

The architecture also includes components for indexing and searching the crawled contents, using state of the art algorithms (our current implementation is based on Apache Lucene). The NSEQL sequence needed to access each document is also stored. This sequence is used by *the ActiveX for automatic navigation Component*, which receives as a parameter a NSEQL program, downloads itself into the user browser and makes it execute the given sequence. This is used to access the documents returned as result of a search against the index, when they cannot be directly accessed in the source by using its URL, due to session issues.

3. DOMAIN DEFINITIONS

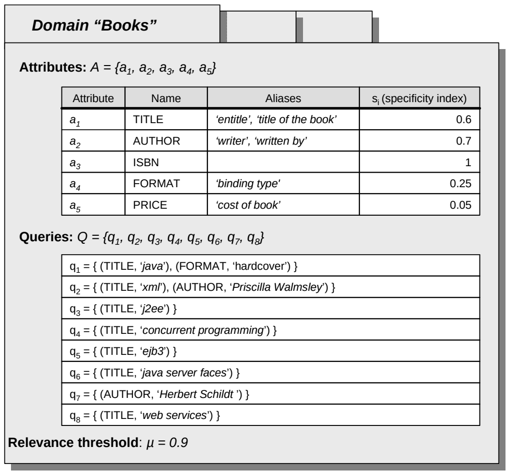
In this section, we describe the domain definitions used to define a data-collection task. A *domain definition* is composed of the following elements:

- A set of attributes $A=\{a_1, a_2, \ldots, a_n\}$. Each attribute a_i has associated a *name*, a set of *aliases* $\{a_i\_alias_1, \ldots, a_i\_alias_k\}$, and a *specificity index* s_i.
- A set of *queries* $Q=\{q_1, q_2, \ldots, q_m\}$ we want to execute on the discovered relevant forms. Each query q_j is a list of pairs (*attribute*, *value*), where *attribute* is an attribute of the domain and *value* is a string (it can be empty).
- A *relevance threshold* denoted as μ.

An *attribute* represents a field that may appear in the query forms that are relevant to the data-collection task.

The *aliases* represent alternative labels that may identify the attribute in a query form. For instance, the attribute AUTHOR, from a domain used for collecting data about books, could have aliases such as “*writer*” or “*written by*”. It is important to notice that the study in [5] concluded that the aggregate schema vocabulary of web forms in the same domain tends to converge at a relatively small size. They also detected a Zipf-like distribution of attribute frequencies (thus, a small set of “dominant” attributes are much more frequent than the rest of attributes). This supports the feasibility of creating effective domain definitions in a fast way: exploring a few sources in the domain is usually enough to find the most important attributes and aliases.

The *specificity index* (denoted s_i) of an attribute a_i is a number between 0 and 1 indicating how probable is that a query form containing such attribute is actually relevant to the domain. For



Domain "Books"

Attributes: *A = {a$_1$, a$_2$, a$_3$, a$_4$, a$_5$}*

Attribute	Name	Aliases	s_i (specificity index)
a_1	TITLE	*'entitle', 'title of the book'*	0.6
a_2	AUTHOR	*'writer', 'written by'*	0.7
a_3	ISBN		1
a_4	FORMAT	*'binding type'*	0.25
a_5	PRICE	*'cost of book'*	0.05

Queries: *Q = {q$_1$, q$_2$, q$_3$, q$_4$, q$_5$, q$_6$, q$_7$, q$_8$}*

q_1 = { (TITLE, '*java*'), (FORMAT, 'hardcover') }
q_2 = { (TITLE, '*xml*'), (AUTHOR, '*Priscilla Walmsley*') }
q_3 = { (TITLE, '*j2ee*') }
q_4 = { (TITLE, '*concurrent programming*') }
q_5 = { (TITLE, '*ejb3*') }
q_6 = { (TITLE, '*java server faces*') }
q_7 = { (AUTHOR, '*Herbert Schildt* ') }
q_8 = { (TITLE, '*web services*') }

Relevance threshold: *μ = 0.9*

Figure 2. Example domain definition for Books

instance, in an example domain for collecting book data, the attribute ISBN would have a very high value (e.g. 0.95), since a query form allowing queries for the ISBN attribute is almost certainly a form allowing to search books; the PRICE attribute would have a low value such as 0.05, since a query form containing it could be related to any kind of product.

Finally, the domain also includes a *relevance threshold* μ. The specificity indexes and the threshold will be used to determine if a given form is relevant to a domain.

Figure 2 shows an example domain definition for the task of collecting pages containing data about books on the subject of Java and XML programming. The relevance threshold for this domain is set to 0.9.

4. PROCESSING FORMS WITH THE FORM ANALYZER

In this section, we describe how the crawler processes each found form. The performed steps are:

- For every domain, the system tries to match its attributes with the fields of the form, using visual distance and text similarity heuristics (see subsection 4.1).
- By using the output of the previous step, the system determines if the form is relevant with respect to the domain (described in subsection 4.2).
- If the form is relevant, the crawler uses it to execute the queries defined in the domain. For each query, we obtain a new route to add to the list of routes. The new route will be dealt with as any other route fetched by the crawler (subsection 4.3).

4.1 Associating Form Fields and Domain Attributes

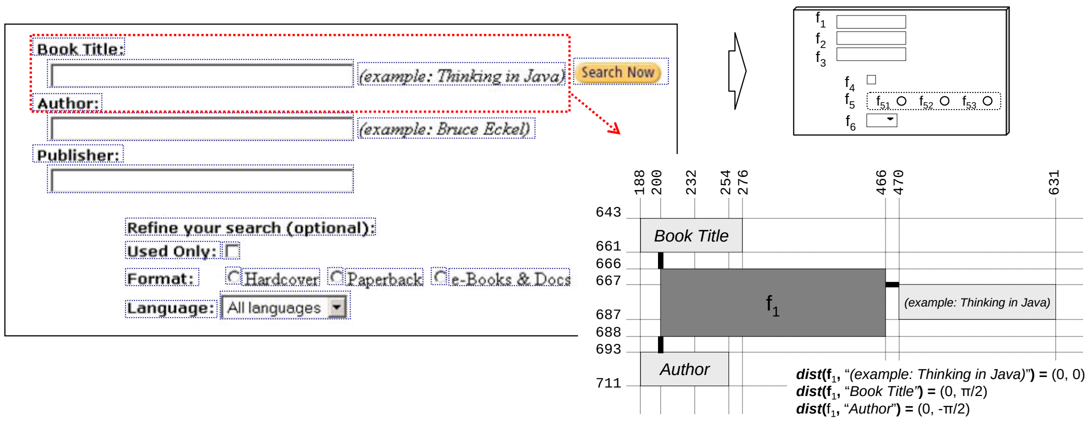
Given a form *f* located in a certain HTML page and a domain *d* describing a data-collecting task, our goal at this stage is to determine whether *f* allows executing queries for the attributes of the domain *d* or not. Our method consists of two steps:

1. Determining which texts are associated with each field of the form. This step is based on heuristics using visual distance measures between the form fields and the texts surrounding them.
2. Trying to relate the fields of *f* with the attributes of *d*. The system performs this step by obtaining text similarity measures between the texts associated with each form field and the texts associated with each attribute in the domain definition *d*.

*Measuring visual distance*s. At this step, we consider the texts in the page and compute their visual distance with respect to each field of the form *f*. The visual distance between a text element *t* and a form field *f* is computed as follows:

1. The browser APIs are used to obtain the coordinates of a rectangle enclosing *f* and a rectangle enclosing *t*. If *t* is into an HTML table cell, and it is the unique text inside, then the coordinates of the table cell rectangle are assigned to *t*.
2. We obtain the minimum distance between both rectangles. Distances are not computed in pixels but in more coarse-grained units (we use cells of the approximated visual size of one character).
3. We also obtain the angle of the shortest line joining both rectangles. The angle is approximated to the nearest multiple of *π/4*.

Figure 3a shows one example query form corresponding to an Internet bookshop. We show the distance and angles obtained for some of its texts and fields.



***dist*(f$_1$, "*(example: Thinking in Java)*") =** (0, 0)
***dist*(f$_1$, "*Book Title*") =** (0, π/2)
***dist*(f$_1$, "*Author*") =** (0, -π/2)

Figure 3a. Example query form and visual distances and angles for field *f₁*

Fields	***Texts***	***(dist,θ)***
f_1 √	**(example: Thinking in Java)**	(0,0)
√	**Book Title:**	(0, π/2)
	Author:	(0, -π/2)
	(example: Bruce Eckel)	(1, -π/2)
	Publisher:	(3, -π/2)
f_2 √	**(example: Bruce Eckel)**	(0, 0)
√	**Author:**	(0, π/2)
	Publisher:	(0, -π/2)
	(example: Thinking in Java)	(2, π/2)
	Book Title:	(3, π/2)
f_3 √	**Publisher:**	(0, π/2)
	Refine your search (optional):	(1, -π/2)
	(example: Bruce Eckel)	(2, π/2)
	Author:	(3, π/2)
	Used Only:	(3, -π/2)
	Format:	(4, -π/2)
	Hardcover	(4, -π/2)
	Paperback	(4, -π/2)

Fields	***Texts***	***(dist,θ)***
f_4 √	**Used Only:**	(0, π)
√	**Refine your search (optional):**	(0, π/2)
	Hardcover	(0, -π/2)
	Format:	(1, π)
	Language:	(2, -π/2)
f_5	e-Books & Docs	(0, 0)
	Used Only:	(0, 3π/4)
	Language	(0, -3π/4)
√	**Format:**	(1, π)
	Refine your search (optional):	(1, π/2)
f_{51} √	**Hardcover**	(0, 0)
	Used Only:	(0, 3π/4)
	Language:	(0, -3π/4)
	Format:	(1, π)
	Refine your search (optional):	(1, π/2)

Fields	***Texts***	***(dist,θ)***
f_{52}	Hardcover	(0, π)
√	**Paperback**	(0, 0)
	Refine your search (optional):	(1, π/2)
f_{53}	Paperback	(0, π)
√	**e-Books & Docs**	(0, 0)
	Refine your search (optional):	(2, 3π/4)
f_6 √	**Language:**	(0, π)
	Hardcover	(0, π/2)
	Paperback	(0, π/2)
	Format:	(1, π)
	Used Only:	(1, π/2)
	Refine your search (optional):	(3, π/2)

f_1 [(example: Thinking in Java)] [Book Title:]; **f_2** [(example: Bruce Eckel)] [Author:]; **f_3** [Publisher:];
f_4 [Used Only:] [Refine your search (optional):];
f_5 [Format:]; **f_{51}** [Hardcover]; **f_{52}** [Paperback]; **f_{53}** [e-Books & Docs]; **f_6** [Language]

Figure 3b. Texts associated to each field in the form of Figure 3a

Associating texts and form fields. For each form field, our goal is to obtain the texts "semantically linked" with it in the page. For instance, in the Figure 3a the strings semantically linked to the first field are "*Book Title*" and "*(example: 'Thinking in Java')*". For pre-selecting the "best texts" for a field *f*, we apply the following steps:

1. We add all the texts having the shortest distance *d* with respect to *f* to the list.
2. Those texts having a distance lesser than *k•d* with respect to *f* are added to the list ordered by distance (*k* is a configurable factor usually set to *5*). This step discards those texts that are significantly further from the field.
3. Texts with the same distance are ordered according to its angle. The preference order for angles privileges texts aligned with the fields (that is, angle multiple of *π/2*); it also privileges left with respect to right and top with respect to bottom, because they are the preferred positions for labels in forms.

As output of the previous step we have an ordered list of texts, which are probably associated to each form field. Then we post-process the lists as follows:

1. We ensure that a given text is only present in the list of one field. The rationale for this is that at the following stage of the form ranking process (which consists in matching form fields and "searchable" attributes), we will need to associate unambiguously a certain text with a given form field.
2. We ensure that each field has at least one associated text. The rationale for this is that, in real pages, a given form field always has some associated text to allow the user to identify its function. For instance, if the list of a field f_1 contained the texts t_1 and t_2 (in that order), and the list of a field f_2 only contained the text t_1, then we would choose to remove t_1 from the list of f_1, since removing it from the list of f_2 would leave the field with an empty list.

Figure 3b shows the process for the example form of Figure 3a. For each field[1] of the form, we show the ordered list of texts obtained by applying the visual distance and angle heuristics. The texts remaining in the lists after the post-processing steps are boldfaced in the figure.

Associating form fields and domain attributes. At this step we try to detect the form fields which correspond to attributes of the target domain. We distinguish between two kinds of fields:

- *Bounded fields*. We term as bounded those fields offering a finite list of possible query values, such as *select-option* fields, *checkbox* fields or *radio buttons*.
- *Unbounded fields*. We term as unbounded those fields whose query values are not limited, such as *text boxes*.

The basic idea to rank the "similarity" between a field *f* and an attribute *a* is to measure the textual similarity between the texts associated with *f* in the page (obtained as shown in the previous step) and the texts associated with *a* in the domain (the attribute name and the aliases). When the field is *bounded*, the system also takes into account the text similarities between the possible values of *f* in the page[2] and the query input values specified for *a* in the domain queries. Text similarity measures are obtained using a method proposed in [7] that combines TFIDF and the Jaro-Winkler edit-distance algorithm.

As result, we obtain a table with the estimated similarities between each form field and each attribute. Then, we discard the pairs from the table that do not reach a minimum similarity *threshold*. If the table contains more than one entry for the same attribute, we choose for each attribute the entry with a higher similarity but trying to assure that no field with an entry above the threshold is left unassigned.

[1] Note how the system models the FORMAT 'checkbox' field as a field with three subfields. f_5 refers to the whole set of checkboxes while f_{51}, f_{52} and f_{53} refer to individual *checkboxes*.

[2] Obtaining these values is a trivial step for select-option tags, since their possible values appear in the HTML code enclosed in *option* tags. For *checkbox* and *radio* tags we apply visual distance techniques similar to the ones previously discussed.

Assigment	*Form Field*	*Domain Attribute*	*c_i (confidence)*
A_1	***f_1***	***a_1* = TITLE**	0.71
A_2	***f_2***	***a_2* = AUTHOR**	1
	f_3	*(unassigned)*	
	f_4	*(unassigned)*	
A_3	***f_5***	***a_4* = FORMAT**	1
	f_6	*(unassigned)*	

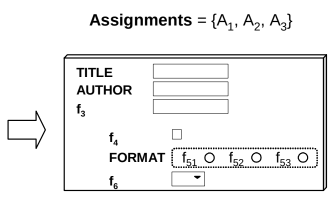


Figure 4. Assignments obtained for the form in Figure 3a, using the domain definition shown in Figure 2

The output of this stage is a set of assignments between form fields and domain attributes. Each of these assignments has a certain *confidence*, which the system sets to the similarity obtained between the field and the attribute.

Figure 4 shows the assignments obtained for the form in Figure 3a, using the domain definition of Book Shopping shown in Figure 2.

4.2 Determining the Relevance of a Form to a Domain

The output of the previous stage is a set of assignments $\{A_1,\ldots, A_k\}$ between form fields and domain attributes. Each assignment has a certain *confidence*, expressed as a number between 0 and 1. We notate the confidence of assignment A_i as c_i.

The method we use to determine if a form is relevant to a domain consists of adding the confidences of each assignment, pondered by the *specificity index* of the attribute involved in it, and checking if the sum exceeds the *relevance threshold* μ. That is, the system checks if the inequality $\sum_{i=1..k} c_i s_i > \mu$ is verified.

For instance, considering the domain definition shown in Figure 2, and the assignments in Figure 4, we would obtain $0.71 \cdot 0.6 + 1 \cdot 0.7 + 1 \cdot 0.25 = 1.376 > \mu = 0.9$

4.3 Executing Queries

Once the system determines that a form is relevant to a certain domain *d*, a new route must be added for each query specified in *d*. Executing a query involves filling in the form according to the query and submitting it.

The first task can be easily done from the assignments which associate form fields and domain attributes.

The second task has its own complications. Although the lightweight mini-browsers the system uses as crawling processes may directly issue a SUBMIT event on the form once it has been filled in, this simple strategy does not work in some websites. This is due to the frequent use of client-side scripting languages to manage form submission. To overcome these difficulties, the system proceeds as follows:

1. The system searches for *input* elements in the form of the types *submit*, *image* or *button* (in that order). Each element is used to try to submit the form by generating a *click* event on it. After each try, the system checks if the event caused a new navigation in the browser. If it was not the case, it tries the next element.

2. If the previous step is unsuccessful (typically because the searched types of *input* elements do not exist), the system concludes that the way used to submit the form is clicking on an anchor with some associated client-scripting code (typically Javascript). Therefore, the system looks for anchors located visually close to the form and having associated some client-side script in either the *href* or the *onClick* attributes. The anchors obtained are ordered according to its visual proximity to the form and to the text similarity between their associated texts and a set of pre-defined texts commonly used to indicate form submission (e.g. '*search*', '*go*', '*submit*',…). The system tries to generate a click event on the anchors in the list and checks if the event caused a new navigation in the browser.

3. If all the previous steps fail, the system generates a SUBMIT event on the form.

5. EXPERIENCE

To evaluate the performance of our approach, we tested it on three different domains: Books Shopping, Music Shopping and Movies Shopping websites.

The process for creating the domain definitions was the following: for each domain, we manually explored 10 sites at random, from the respective Yahoo Directory[3] category and used them to define the attributes and aliases. The specificity indexes and the relevance threshold were also manually chosen from our experience visiting these sites. The resulting domain definitions are shown in Figure 5.

Once the domains were created, we used DeepBot to crawl 20 websites of the respective Yahoo Directory category. The websites visited by DeepBot for each domain are shown in the extended version of this paper [2]. The websites used to define the attributes and aliases are grouped in a dataset named *Training*, while the remaining sites are grouped in a dataset named *Advanced*.

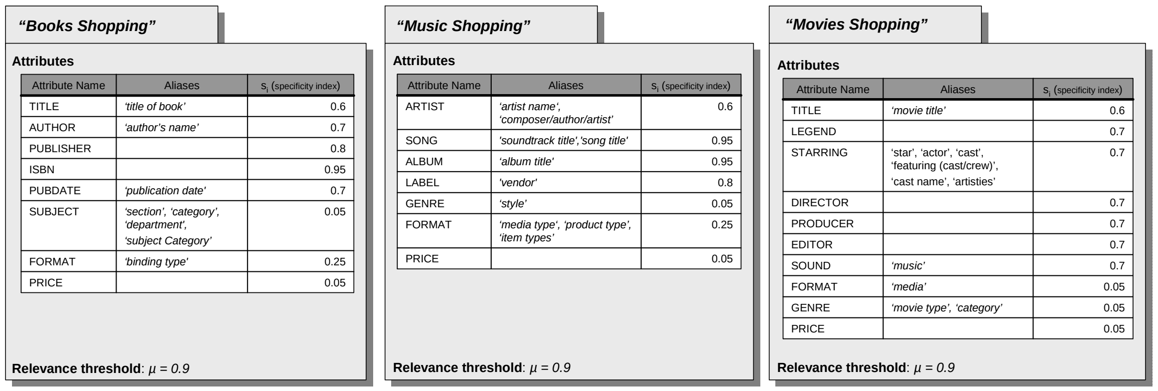
To check the accuracy of the results obtained, we manually analyzed the websites and compared the results with those obtained by DeepBot. We measured the results at each stage of the process: associating texts with form fields, associating form fields with domain attributes, establishing the relevance of a form to a domain, and executing the queries on the relevant forms.

To quantify the results, we used standard Information Retrieval metrics: precision, recall and F_1-measure. For instance, in the stage of associating form fields and domain attributes, the metrics are defined as follows; we defined the following variables to use in (1).

- $FieldAttributeA_{DeepBot}$: set of the associations between form fields and domain attributes discovered by DeepBot.

- $FieldAttributeA_{Real}$: set of the associations between form fields and domain attributes discovered by the manual analysis.

[3] http://dir.yahoo.com



"Books Shopping"

Attributes

Attribute Name	Aliases	s_i (specificity index)
TITLE	*'title of book'*	0.6
AUTHOR	*'author's name'*	0.7
PUBLISHER		0.8
ISBN		0.95
PUBDATE	*'publication date'*	0.7
SUBJECT	*'section', 'category', 'department', 'subject Category'*	0.05
FORMAT	*'binding type'*	0.25
PRICE		0.05

Relevance threshold: *μ = 0.9*

"Music Shopping"

Attributes

Attribute Name	Aliases	s_i (specificity index)
ARTIST	*'artist name', 'composer/author/artist'*	0.6
SONG	*'soundtrack title','song title'*	0.95
ALBUM	*'album title'*	0.95
LABEL	*'vendor'*	0.8
GENRE	*'style'*	0.05
FORMAT	*'media type', 'product type', 'item types'*	0.25
PRICE		0.05

Relevance threshold: *μ = 0.9*

"Movies Shopping"

Attributes

Attribute Name	Aliases	s_i (specificity index)
TITLE	*'movie title'*	0.6
LEGEND		0.7
STARRING	'star', 'actor', 'cast', 'featuring (cast/crew)', 'cast name', 'artisties'	0.7
DIRECTOR		0.7
PRODUCER		0.7
EDITOR		0.7
SOUND	*'music'*	0.7
FORMAT	*'media'*	0.05
GENRE	*'movie type', 'category'*	0.05
PRICE		0.05

Relevance threshold: *μ = 0.9*

Figure 5. Domain definitions: Books, Music and Movies

$$\Pr ecision_{FieldAttributeA} := \frac{\left|\text{FieldAttributeA}_{\text{DeepBot}} \cap \text{FieldAttributeA}_{\text{Real}}\right|}{\left|\text{FieldAttributeA}_{\text{DeepBot}}\right|} \qquad (1)$$

$$\operatorname{Re} call_{FieldAttributeA} := \frac{\left|\text{FieldAttributeA}_{\text{DeepBot}} \cap \text{FieldAttributeA}_{\text{Real}}\right|}{\left|\text{FieldAttributeA}_{\text{Real}}\right|}$$

$$F_1 - measure_{FieldAttributeA} := \frac{2 \times \Pr ecision_{FieldAttributeA} \times \operatorname{Re} call_{FieldAttributeA}}{\Pr ecision_{FieldAttributeA} + \operatorname{Re} call_{FieldAttributeA}}$$

The metrics for the remaining stages were defined in a similar manner. See the extended version of this paper [2] for detail.

5.1 Experimental Results

Table 1 summarizes the obtained experimental results. For each domain, it shows the values obtained for the *Training* dataset (S_1, sites used to define the domains), the *Advanced dataset* (S_2, the remaining sites) and in the *Global* dataset (S_1+S_2, *Training + Advanced*).

In order to calculate the metrics for form-domain and field-attribute associations, "quick search" and authentication forms have not been considered. The results include only multi-field forms of the kind usually employed for "advanced search" forms. In addition, the results for the field-attribute associations have been measured independently of the previous stage (text-field associations).

The obtained results are quite promising: all the metrics show high values and some of them even reach 100%. Now we discuss the reasons behind the mistakes committed by DeepBot at each stage.

Recall in associating forms and domains reached 100% in every case but in the *Advanced* dataset of the Music and Movies domains (which reached 95%). In the music domain, the reason was that the *ProMusicFind* source used an alias for the ARTIST attribute which did not match with any of the aliases defined in the domain. In addition, the form only had two fields so, even though the system correctly assigned the other one to a domain attribute ("*Album Title*"), it was not enough to exceed the relevance threshold. In the movies domain, the query form from source *IGN.COM* only had two searchable fields (*title, genre*) matching with attributes in our domain definition. Although the system correctly matched both, it was not enough to reach the threshold.

The precision and recall values obtained for the associations between texts and form fields exceeded 80% except in the *Advanced* dataset of the Books domain (0.73 precision and 0.79 recall). The majority of the errors in this dataset came from a single source (*Blackwell's Bookshop*). If we did not have into account this source, the metrics would take values similar to those reached by the other ones.

The failures at this stage came mainly from *bounded* fields that did not have any globally associated text in the form (the form only included the texts corresponding to its values). That is contrary to one of our heuristics, which assumed that every form field should have at least one associated text to "explain" the function of the field to the user.

Finally, *Recall* and *Precision* also reach high values (> 90% except in one case) in the associations between form fields an domain attributes. The mistakes at this stage occurred because the domain did not include the alias used in the form for some attribute.

6. RELATED WORKS

In recent years, several works have addressed the problem of accessing the Hidden Web using a variety of approaches.

The system more similar to ours is HiWE [14]. HiWE is a task-specific crawler able to automatically recognizing and filling in forms relevant to a given domain. HiWE also uses visual distance measures to find the texts associated to each field in a form, and text similarity measures to match fields and domain attributes.

When analyzing forms, HiWE only associates one text to each form field. The text is chosen in the following way: first, HiWE finds the four closest texts to the field; second, it chooses one of them according to a set of heuristics taking into account the relative position of the candidate texts with respect to the field (texts at the left and at the top are privileged), and their font sizes and styles.

To learn how to fill in a form, HiWE matches the text associated with each form field and the labels associated to the attributes defined in its LVS table (a concept that plays a similar role to our domain definitions). In this process, HiWE has the following restriction: it requires the LVS table to contain an attribute definition matching with each unbounded form field.

Now we discuss the differences between HiWE and our system. The process followed by DeepBot has several advantages:

- DeepBot may use a form, even though it has some fields that do not match any attribute of the domain. For instance, the domain definition in Figure 2 does not have any attribute matching with the "*Publisher*" field in Figure 3a.

Table 1. Experimental results

	Books Shopping			Music Shopping			Movies Shopping		
	S_1	S_2	S_1+S_2	S_1	S_2	S_1+S_2	S_1	S_2	S_1+S_2
Submitted Forms									
Precision	*13/13* **1.00**	*11/11* **1.00**	*24/24* **1.00**	*10/10* **1.00**	*9/9* **1.00**	*19/19* **1.00**	*12/12* **1.00**	*9/9* **1.00**	*21/21* **1.00**
Form-Domain Associations									
Precision	*13/13* **1.00**	*11/11* **1.00**	*24/24* **1.00**	*10/10* **1.00**	*9/9* **1.00**	*19/19* **1.00**	*12/12* **1.00**	*9/9* **1.00**	*20/20* **1.00**
Recall	*13/13* **1.00**	*11/11* **1.00**	*24/24* **1.00**	*10/10* **1.00**	*9/10* **0.90**	*19/20* **0.95**	*12/12* **1.00**	*9/10* **0.90**	*21/22* **0.95**
F_1-measure	**1.00**	**1.00**	**1.00**	**1.00**	**0.95**	**0.97**	**1.00**	**0.95**	**0.97**
Field-Attribute Associations									
Precision	*54/55* **0.98**	*50/50* **1.00**	*104/105* **0.99**	*37/37* **1.00**	*31/33* **0.94**	*68/70* **0.97**	*45/46* **0.98**	*33/33* **1.00**	*78/79* **0.99**
Recall	*54/54* **1.00**	*50/53* **0.94**	*104/107* **0.97**	*37/37* **1.00**	*31/37* **0.84**	*68/74* **0.92**	*45/45* **1.00**	*33/35* **0.94**	*78/80* **0.98**
F_1-measure	**0.99**	**0.97**	**0.98**	**1.00**	**0.89**	**0.94**	**0.99**	**0.97**	**0.98**
Text-Field Associations									
Precision	*129/142* **0.91**	*101/137* **0.73**	*230/279* **0.82**	*93/110* **0.83**	*107/132* **0.81**	*199/242* **0.82**	*154/179* **0.86**	*163/184* **0.89**	*317/363* **0.87**
Recall	*129/132* **0.98**	*101/127* **0.79**	*230/259* **0.88**	*92/94* **0.98**	*107/109* **0.98**	*199/203* **0.98**	*154/168* **0.92**	*163/181* **0.90**	*317/349* **0.91**
F_1-measure	**0.94**	**0.76**	**0.85**	**0.90**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**

- DeepBot correctly detects when a field has more than one associated text; this can result in better accuracy when matching form fields and domain attributes.
- In addition, the decision of assigning a text to a field is not based only on conditions "local" to the field: the context provided by the whole form is also taken into account in our heuristics. For instance, in our example form of Figure 3a, HiWE would erroneously assign the text "*Hardcover*" to the second radio button element (f_{52}), since the text is the closest one and it is located at the left of the field. Nevertheless, our system correctly assigns the text "*e-Books & Docs*" to f_{53}, "*Paperback*" to f_{52} and "Hardcover" to f_{51}.
- Finally, another advantage is that DeepBot fully supports Javascript sources.

Reference [3] presents another system for domain-specific crawling of the Hidden Web. Nevertheless, they only deal with *full text* search forms; these forms have a single field allowing search by keyword on unstructured collections. In turn, our system focuses on the multi-attribute forms typically used to query structured data.

Reference [12] addresses the problem of automatically generating keyword queries to crawl all the content behind a form. New techniques are proposed to automatically generate new search keywords from previous results, and to prioritize them in order to retrieve the content behind the form, using the minimum number of queries. The ability to automatically generate new queries would be an interesting new feature for DeepBot, so this work is complementary to ours. Nevertheless, the presented techniques would need to be adapted since they do not deal with multi-attribute forms.

The problem of extracting the full content behind a form has been also addressed in [11]. This system does not deal with forms requiring *textbox* fields to be filled in.

The Hidden Web can also be accessed using the *meta-search* paradigm instead of the *crawling* paradigm. In *meta-search* systems [6,15,9,8,10], a query from the user is automatically redirected to a set of underlying relevant sources, and the obtained results are integrated to return a unified response. The meta-search approach is more lightweight than the crawling approach, since it does not require indexing the content from the sources; it also guarantees up to date data. Nevertheless, users will get higher response times since the sources are queried in real-time.

7. CONCLUSIONS

In this paper, we have described the architecture of DeepBot, a crawling system able to access the contents of the Hidden Web. Our approach is based on a set of domain definitions, each one describing a data-collecting task. From the domain definition, the system uses several heuristics to automatically identifying relevant query forms and learning how to execute queries on them. We have tested our techniques for several real-world data-collecting tasks, obtaining a high degree of effectiveness.

8. ACKNOWLEDGMENTS

This research was partially supported by the Spanish Ministry of Education and Science under project TSI2005-07730.

Alberto Pan's work was partially supported by the "Ramón y Cajal" programme of the Spanish Ministry of Education and Science.

9. REFERENCES

[1] Álvarez, M., Pan, A., Raposo, J., Hidalgo, J. Crawling Web Pages with Support for Client-Side Dynamism. Published in Lecture Notes in Computer Science 4016, pp. 252-262, 2006. Issue corresponding to Proceedings of the 7th International Conference on Web Age Information Management. 2006.

[2] Álvarez, M., Raposo J., Pan, A., Cacheda, F., Bellas, F., Carneiro, V. DeepBot: A Focused Crawler for Accessing Hidden Web Content. http://www.tic.udc.es/~mad/publications/deepbotfocused_extended.pdf.

[3] Bergholz, A., Chidlovskii, B. Crawling for Domain-Specific Hidden Web Resources. In Proceedings of the 4th Int. Conference on Web Information Systems Engineering.2003.

[4] Bergman, M. The Deep Web. Surfacing Hidden Value. http://brightplanet.com/technology/deepweb.asp. 2001.

[5] C.-C. Chang, K., He, B., Patel, M., Zhang, Z. Structured Databases on the Web: Observations and Implications. SIGMOD Record, 33(3). 2004.

[6] C.-C. Chang, K., He, B., Zhang, Z. MetaQuerier over the Deep Web: Shallow Integration Across Holistic Sources. In Proceedings of the VLDB Workshop on Information Integration on the Web. 2004.

[7] Cohen, W., Ravikumar., P., Fienberg, S. A Comparison of String Distance Metrics for Name-Matching Tasks. In Proceedings of IJCAI-03 Workshop. 2003.

[8] Gravano, L., Ipeirotis, P., Sahami, M. QProber: A System for Automatic Classification of Hidden-Web Databases. In ACM Transactions on Information Systems, vol. 21(1), 2003.

[9] He, H., Meng, W., Yu, C., and Wu, Z. Automatic Integration of Web Search Interfaces with WISE-Integrator. In VLDB Journal, Vol.13, No.3, pp.256-273. 2004.

[10] Ipeirotis P., Gravano L. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. Proceedings of the 28th Very Large DataBases Conference. 2002.

[11] Liddle, S., Embley, D., Scott, Del., Yau Ho, Sai. Extracting Data Behind Web Forms. Proceedings of the 28th Intl. Conference on Very Large Databases. 2002.

[12] Ntoulas, A., Zerfos et al.. Downloading Textual Hidden Web Content Through Keyword Queries. Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries. 2005.

[13] Pan, A., Raposo, J., Álvarez, M., Hidalgo, J. and Viña, A. Semi-Automatic Wrapper Generation for Commercial Web Sources. In Proceedings of IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context. 2002.

[14] Raghavan S., Garcia-Molina, H. Crawling the Hidden Web. Technical Report 2000-36, Computer Science Department, Stanford University, December 2000. Available at http://dbpubs.stanford.edu/pub/2000-36)

[15] Zhang, Z., He, B., C.-C. Chang, K. Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. In Proceedings of the 31st Very Large Data Bases Conference, 2005.