

# An Example-Based Environment for Wrapper Generation\*

Paulo B. Golgher, Alberto H. F. Laender, Altigran S. da Silva<sup>†</sup>, and  
Berthier Ribeiro-Neto

Department of Computer Science  
Federal University of Minas Gerais  
31270-010 Belo Horizonte MG Brazil  
{golgher,laender,alti,berthier}@dcc.ufmg.br

**Abstract.** In the so-called Web information systems, the role of extracting data of interest from Web sites is played by software components generically known as wrappers. As a result, the existence of flexible tools for designing, developing and maintaining wrappers is crucial. In this paper, we present WByE (Wrapping By Example), a user-oriented set of tools for helping the user to build wrappers. WByE is based on information implicitly provided by the user by means of suitable and intuitive interfaces. It includes two components: the ASByE tool, used for generating specifications on how to fetch desired pages (be them static or dynamic), and the DEByE tool, used for the extraction of data implicitly present in the fetched pages.

## 1 Introduction

One of the most important features of the so-called *Web information systems* is the capability of incorporating data from many different Web sites. In such systems, the role of extracting data of interest from an specific Web site is played by software components generically known as *wrappers*. As pointed out in [16], the task performed by a wrapper roughly involves three steps: (1) the fetching of the pages from the Web site; (2) the identification and extraction of data (objects) implicitly present in the fetched pages; and (3) the storage of the extracted data in suitable a format (e.g., XML, relational tables, etc.) for further manipulation. The range of Web applications requiring wrappers is enormous and, therefore, the existence of flexible tools for designing, developing, and maintaining them is crucial.

Motivated by this, many works have been developed in the last few years that address the problem of wrapper generation. The pioneer works were carried out using very sophisticated mechanisms, but which required some expertise from the user in subjects such as automata, grammars, agents, internet protocols,

---

\*This work is supported by Project SIAM (grant MCT/FINEP/PRONEX 76.97.1016.00) and by individual research grants from CNPq and CAPES.

<sup>†</sup>On leave from the University of Amazonas, Brazil.

and computer programming in general. This is the case of many very successful projects such as *ARANEUS* [3], *TSIMMIS* [6], *LORE* [14], *ARIADNE* [13], *FLORID* [12], and *W3QS* [8] to name just a few.

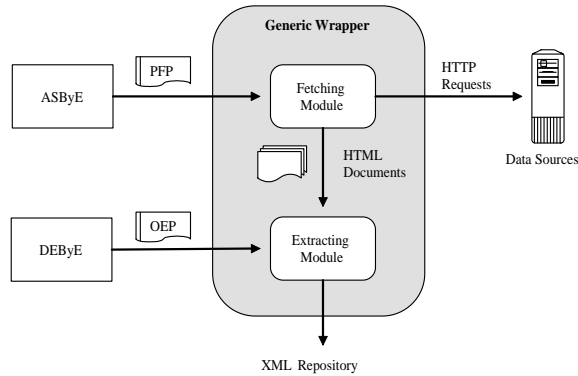
More recently, new approaches for wrapper generation have emerged that are oriented towards the less experienced Web user with few or none programming skills. This is the case for tools such as *NoDoSe* [2], *W4F* [16], and *XWRAP* [11]. Most of these approaches, however, address the problem of fetching Web pages in a simplistic manner or assume that the user already has collected the documents in a separate process.

In this paper, we present the *WByE* (*Wrapping By Example*) environment, a user-oriented set of tools for helping the user in the task of fetching and extracting data from Web sites. The ground for the development of such an environment is the idea that the user is able to interact with high-level graphical interfaces to provide examples of how steps (1) and (2) mentioned before should be performed for obtaining data of interest from a particular Web site. These examples are then used to automatically generate low-level specifications (including page fetching commands and data extraction patterns) that will be used to automatically collect and extract data from the Web site.

The *WByE* environment is composed of two integrated tools named *ASByE* (Agent Specification By Example) and *DEByE* (Data Extraction By Example). The first tool is used to generate a *page fetching plan* (PFP) that guides the behavior of an agent responsible for fetching a set of pages from the target Web site. The user interacts with the tool's interface to provide examples of how to reach the desired pages within a site, to fill any forms needed, and to navigate a set of related pages to form a collection. The generated fetching plan specifies how the agent will perform these tasks automatically when invoked. ASByE is specially suitable for building plans to fetch pages generated automatically as a result of filling an HTML form.

The second tool, DEByE, is used to specify how to extract data from the pages fetched and to logically organize them according to the user's perception of the implicit structure of the data in the pages. The DEByE's interface is based on a metaphor of nested tables which are built by the user by cutting and pasting pieces of data present on a sample page of the Web site he/she is interested in. From the examples provided through the assembled tables, the tool derives an *object extraction pattern* (OEP) which describes the textual context and the structure of the objects to be extracted.

Once generated by the corresponding tools, the PFP and the OEP are given as input to a general purpose wrapper, which includes a page fetching component and an extractor component. The wrapper then automatically performs the page fetching and the data extraction tasks, and stores the extracted data according to an XML-based format. If the wrapper needs to be modified due to changes in the site, the *WByE* environment allows the user to revise the PFP and/or the OEP accordingly. Fig. 1 illustrates the general framework provided by the *WByE* environment.



**Fig. 1.** Wrapper generation using WByE.

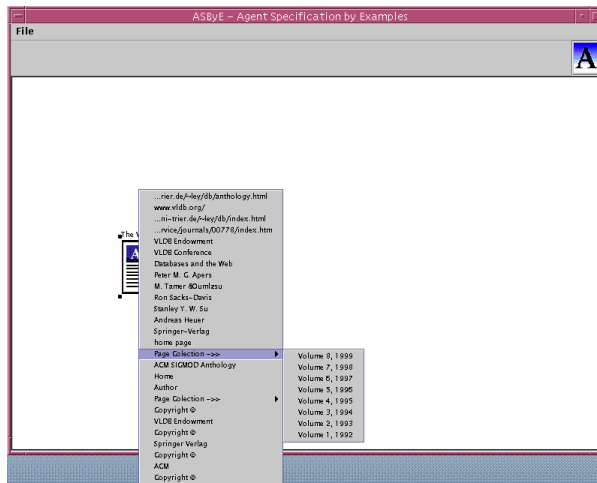
The DEByE tool and the DEByE approach for data extraction have been detailed described in [9, 15]. For this reason, in the present paper, we focus our discussion on the features and the functionality of the ASByE tool. Thus, the rest of the paper is organized as follows. Section 2 describes the ASByE tool. Section 3 presents an example of a wrapper constructed for a popular Web site using the WByE environment. Finally, Section 4 concludes the paper.

## 2 The ASByE Tool

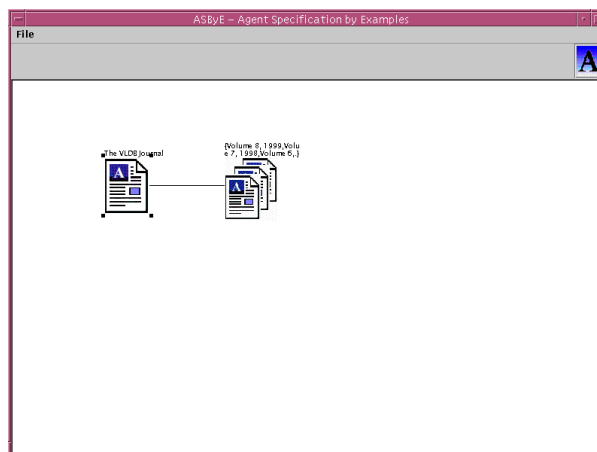
In this section, we give an overview of the ASByE tool. First, we present the visual paradigm used by its user interface. Next, we discuss the features related to the fetching of sets of static pages stored in a Web server. Finally, we show how the tool addresses the problem of collecting sets of dynamic pages, i.e., pages dynamically generated as a result of filling HTTP forms.

### 2.1 Visual Web Exploration

The user interface of the ASByE tool uses a graph-like structure to represent a portion of the Web. The nodes displayed in its work space represent pages and directed arcs represent hyperlinks. Similar metaphors were previously used by the *Hy+* system [7] and by the diagrams of the *ARANEUS* data model [3], which inspired the “look and feel” of the interface. The user navigates from node to node exploring the hyperlinks according to his/her interests. Fig. 2 shows a typical usage session with the interface, while Fig. 3 shows the types of arcs and nodes used by the interface to represent the different pages a user finds when browsing a Web site. The source nodes in the graph (i.e., the ones not pointed by any other node) are called *Web entry points* and are directly selected by the user using a dialog box where he/she enters the URL of the page from where he/she wants to start the exploration. The tool then fetches the page and builds



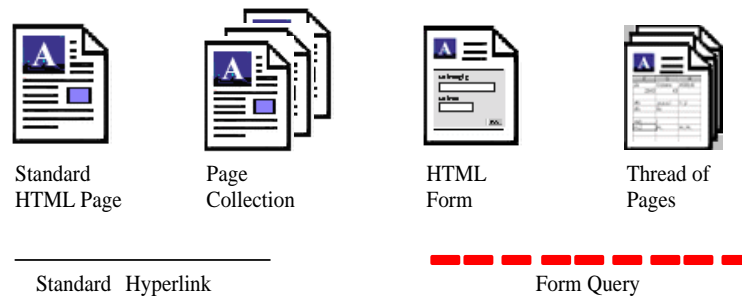
(a)



(b)

**Fig. 2.** Usage session for Web Exploration.

a node corresponding to it. From this point onwards, the user can select for each node an operation he/she wants to perform. The set of operations available depends on the type of the node reached. The most common and simple operation allows the user to select a hyperlink to explore, what is done by selecting one of the hyperlinks shown in a pop-up menu. When a hyperlink is selected, the corresponding page is fetched and a new node is created in the interface's work space to represent it.



**Fig. 3.** Arcs and nodes used to represent a portion of the Web.

Alternatively, the user can follow a hyperlink by selecting the “View in Browser” operation. This operation opens a window of a browser\* where the user can analyze the page contents and select the hyperlink of his/her interest. Due to specific instrumentation included in the page previously to its presentation, the ASByE tool is notified of the hyperlink selected by the user, what causes the fetching of the corresponding page and the creation of a new node to represent the newly select hyperlink. This feature allows the user to explore the Web in the traditional manner, and eases the task of navigating through pages with many hyperlinks, which would be difficult to represent in a pop-up menu.

## 2.2 Page Collections

In many cases, the user is interested in treating a set of logically related pages as a collection. As an example, suppose one is interested in collecting from the DB&LP Web site [4] the pages that contain information on all the volumes of the VLDB Journal. These pages can be reached from the VLDB Journal Page on that site (see Fig. 4). It will be tedious or even unfeasible for the user to specify that he/she wants to individually collect each page corresponding to each volume. To cope with this, the ASByE tool tries to infer, from each page represented by an icon, sets of related links in such a way that a single collecting specification can be generated for the whole collection of pages referred by these links. The tool uses several heuristics to determine which links will be grouped. These heuristics are based solely on the position and on the labels of the hyperlinks. As an additional advantage, when more similar links are added to the page, the corresponding pages will be automatically included in the collection.

In the interface, when the user selects a “Page Collection” entry in the hyperlinks menu (see Fig. 3), a new node is created with an icon that represents a collection of pages. We note that no page is fetched at this moment and that this node is considered terminal. Moreover, the only operation available is for the generation of the fetching plan for the page collection.

\* Currently, only Netscape is supported, but the tool can be easily extended to support the MS Explorer.

## The VLDB Journal

The VLDB Journal and the proceedings of the [VLDB Conference](#) are the main publications of the [VLDB Endowment](#).

Editors-in-Chief

[Peter M. G. Apers](#), [M. Tamer Özsu](#), [Ron Sacks-Davis](#), [Stanley Y. W. Su](#)

[VLDB Journal Page at Springer](#)

### Contents

- [Volume 8, 1999](#)
- [Volume 7, 1998](#)
- [Volume 6, 1997](#)
- [Volume 5, 1996](#)
- [Volume 4, 1995](#)
- [Volume 3, 1994](#)
- [Volume 2, 1993](#)
- [Volume 1, 1992](#)

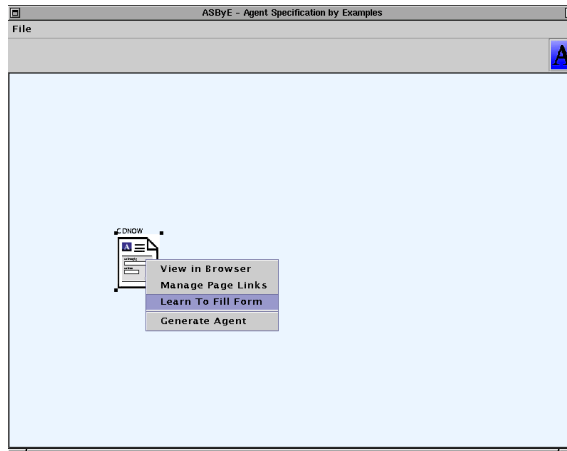
**Fig. 4.** A snapshot of the VLDB Journal page from the DB&LP site.

For identifying a set of links in a page as links to a page collection, the following heuristics are considered: (1) **Hyperlinks Proximity** - Sets of very near hyperlinks in a page are considered candidates to form a collection (e.g., several links separated by a single comma); (2) **Similarity in the URLs** - Similarities in the URLs referred by a group of hyperlinks count positively towards considering them as a collection (e.g., URLs formed by the same directory path); (3) **URL Hosting** - URLs belonging to different HTTP servers are usually not considered as part of a collection; (4) **Enumeration in the Hyperlink Labels** - A group of hyperlinks whose labels form an explicit enumeration is a strong candidate to be considered as a collection (Fig. 4 presents an example of such a case); (5) **Number of Hyperlinks in the Page** - The fewer is the number of hyperlinks the stronger is the support needed for the above heuristics to be taken in consideration when identifying page collections.

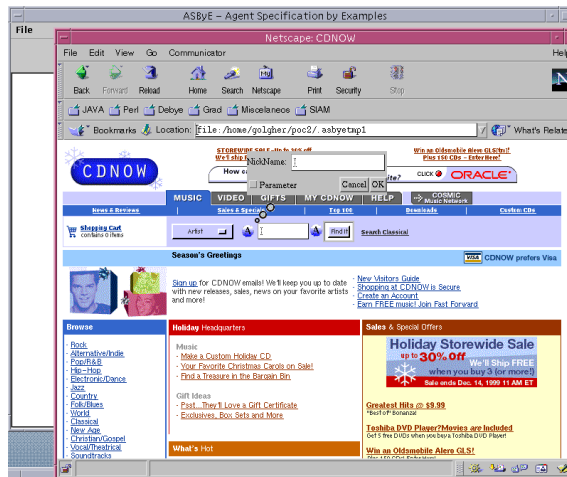
The heuristics adopted work well for pages where the semantic context is well defined, as is the case for the page shown in Fig. 4. However, in many other cases the heuristics may fail for pages whose subject is less focused, as it occurs in pages from the so-called “Web Portals”. For cases like this, the interface allows the user to group or ungroup links by his/herself to form customized page collections.

### 2.3 Dealing with Forms

A recent study [10] estimates that 80% of the Web contents are located on the so-called *hidden Web*, i.e., on Web pages which are dynamically generated. In most of the cases, these pages are generated by programs invoked through HTML forms. Thus, an important feature for any Web agent is the ability of automatically filling these forms to retrieve the generated pages. Moreover, in many cases the agent must take parameters as input to supply different values to form fields each time they run. The ASByE tool provides means for the user to include in a page fetching plan instructions to fill HTTP forms, with default



(a)



(b)

**Fig. 5.** Dealing with forms in the ASByE tool.

values or with values derived from parameters supplied as input at the moment the plan is executed. Here again, the interface asks the user to provide an example of how to fill the form and uses this information to generate instructions for filling it in the collecting specification.

Fig. 5(a) shows an example of navigation that leads to a page containing a form. For this page, the user selected the “Learn to Fill Form Operation”. This operation opens a window of a browser displaying a modified version of the form page. In this version, a small bullet is included near to each form field, as shown in Fig. 5(b) (notice the small round shaped A).

When the user clicks one of these bullets, a pop-up box is opened for the user to specify: (a) if the field should be a parameter or if a default value should be used all the time and (b) a nickname to be used when referring to the field. This nickname eases the task of later assigning values to the parameterized fields, since the user does not need to know the name coded in the HTML description of the page, which might not be representative of the meaning of that field. After filling the fields he/she is interested in, the user finally submits the form as he/she would normally do in an ordinary operation. The tool then uses this information to generate the correct instructions for filling and submitting the form within the page collecting specification.

In most cases, the result of submitting an HTML form is a linked thread of pages. Thus, after the sample form submission, the interface creates a new node using an icon that indicates an answer thread. Next, we discuss how we collect pages that compose an answer thread in our tool.

## 2.4 Collecting Threads of Answer Pages

Although there can be many ways to generate answer pages that result from filling HTML forms, almost all services available on the Web include, in each page, a hyperlink or a button to the next page in the answer. We call such a set of pages a *thread*.

To generate collecting specifications for a thread of answer pages, we again rely on a user action. By selecting the operation “Learn to Follow” from an answer thread node, the user is presented with the first page of the thread shown in the browser window. Actually, this page is a modified version of the first page of the thread, instrumented in such a way that every link selection and button pressing is trapped by the tool. Then, the user is asked to select the link or to press the button that leads to the next page in the thread. This action is trapped by the tool and its request (a GET or POST) is analyzed with the corresponding parameters. Based on this, a number of “hints” to control the behavior of the fetching agent is coded in the page fetching plan. These hints are used to determine how to fetch, from the first page of a thread, all subsequent pages until the last page has been reached.

Depending on the type of mechanism used to fetch the subsequent pages (link or button), one of two available heuristics is used and coded in the PFP. If the user selects a link during the “Learn To Follow” operation, the ASByE tool generates a pattern expression that extracts the link used to retrieve the subsequent results. The tool uses techniques adopted by the DEByE tool [15] for generating this pattern expression. While there is a match for this regular expression, the pages are collected until no more links that match the expression are found (Heuristic 1).

When a button or other mechanism causes a form to be posted for retrieving subsequent pages, the heuristic applied is a bit more complex. The heuristic is based on the fact that generally a variable is passed when the form is posted to inform the page of the thread to be fetched. For instance, the following set of

URLs below corresponds to the URLs used to fetch the second and third pages of a given query on the eBay site [5] (notice the skip variable):

```
http://search.ebay.com/search/search.dll?MfcISAPICommand=GetResult
&query=guitar&SortProperty=MetaEndSort&SortOrder=%5Ba%5D&skip=50
```

```
http://search.ebay.com/search/search.dll?MfcISAPICommand=GetResult
&query=guitar&SortProperty=MetaEndSort&SortOrder=%5Ba%5D&skip=100
```

Thus, the ASByE tool traps the form request generated by the user during the “Learn to Follow” operation and uses this information to infer the variables that change their values on the next request for a page of the thread. This is done by parsing the retrieved document and extracting the values of the corresponding form that was posted in the example given by the user. The numerical values that change from one posting to the other are considered the values that are responsible for determining the current page of the thread. Therefore, for fetching the remaining thread of pages, the tool generates information on how to enumerate the values of these variables (Heuristic 2).

In order to validate the two heuristics presented, we developed PFPs for 10 of the most popular sites on the Web, according to the 100Hot site [1]. Some of the most popular sites (such as the Quote.com site) do not provide any means for generating a thread of answer pages and were therefore discarded. Also, although the eBay and Monster sites were not the 9th and 10th of 100Hot list, they were added because there are not many sites that require the use of the Heuristic 2. The results are shown in Table 1. We observe that the results are very good, except in the case of the Lycos site. In this case, ASByE failed to generate a correct pattern for Heuristic 1 because the first page of the thread used to generate the pattern had a completely different layout than the other pages in the thread.

Site	Heuristic	Query	Pages Available	Pages Retrieved
Yahoo!	Heuristic 1	“dynamic page crawler”	10	10 (100%)
Microsoft	Heuristic 1	“windows”	9	9 (100%)
AOL	Heuristic 1	“page crawler”	41	41 (100%)
Lycos	Heuristic 1	“dynamic page crawler”	100	1 (1%)
Excite	Heuristic 2	“dynamic page crawler”	100	100 (100%)
Altavista	Heuristic 1	“dynamic page crawler”	20	20 (100%)
GO	Heuristic 1	“dynamic page crawler”	40	40 (100%)
Amazon	Heuristic 1	“database system”	5	5 (100%)
eBay	Heuristic 2	“guitar”	218	218 (100%)
Monster	Heuristic 2	“search engine”	8	8 (100%)

**Table 1.** Validation of the heuristics for thread collection.

### 3 An Example

In order to illustrate a typical interaction of a user with the WByE environment, in this section we present the development of a wrapper intended to fetch and extract data from the Amazon bookstore site. This wrapper could be used in many applications, such as shop comparison and mediators. The wrapper generation process includes two major steps: (1) the generation of the page fetching plan (PFP), with information on how to fetch the data, and (2) the generation of the object extraction pattern (OEP) which describe how to structure and extract the data of interest.

The first issue to address when generating the PFP is to locate the HTML page which allows the user to make queries to retrieve the data. Using one of the exploring features presented in the previous section, the user can easily reach the Amazon's advanced search page, which provides author and title based search, as illustrated in Fig. 6(a).

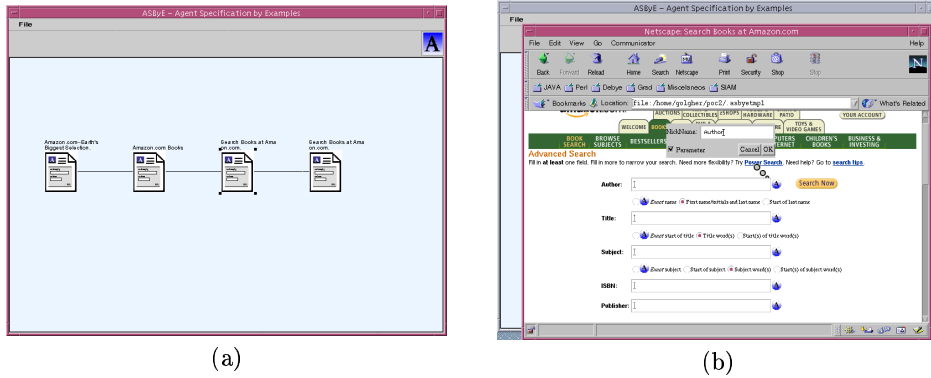


Fig. 6. Locating the Amazon's advanced search page and defining fields properties.

Then, the user determines which query fields should be parameterized and, optionally, gives a nickname for each field using the special bullets inserted in the page, as illustrated in Fig. 6(b). In this case, the author and title query fields were chosen, thus allowing the agent to fetch data related to different authors and/or titles each time it is executed. After defining these properties of each field, the user submits the form as he/she would do in a traditional navigation. The ASByE tool traps the information submitted, fetches the corresponding pages, and represents them by an icon in its interface, as shown in Fig. 7(a).

The result of submitting a query to the Amazon Bookstore site is a thread of one or more pages, connected by a "Next Results" link which retrieves the next 50 results related to the initial query. Thus, the user has to use the "Learn To Follow" operation in order to give an example of how to follow this thread of pages. This operation opens a browser window showing the first page of the thread. The user then gives the example by clicking in the "Next Results" link

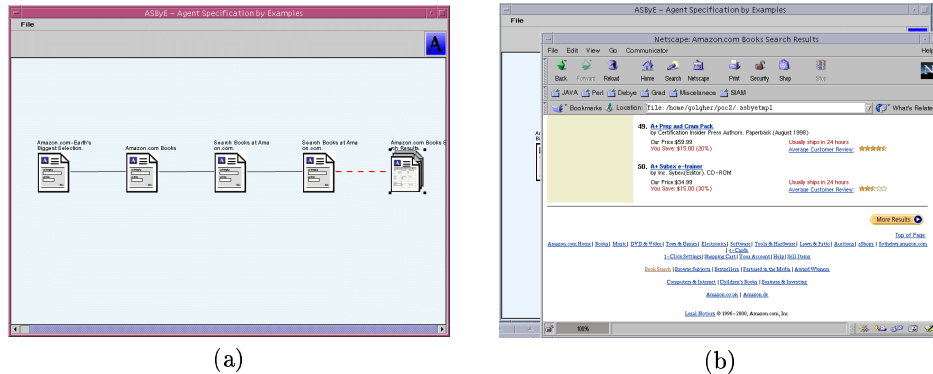


Fig. 7. Representation of the results (a) and learning to follow the Amazon's site.

of this page. Fig. 7(b) illustrates this step. Finally, the user generates the page fetching plan. An excerpt of PFP generated for this example are shown in Fig. 8.

```

<PFP source='Amazon'>
  <PLAN>
    <REQUEST method=1>
      http://www.amazon.com/exec/obidos/search-handle-form/
    </REQUEST>
    <REQUEST_FIELDS>
      <VALUE field='index' parameter='no'> books </VALUE>
      <VALUE field='query-0' parameter='yes'> internet </VALUE>
      ...
    </REQUEST_FIELDS>
    <FOLLOW_METHOD>
      <METHOD 1 </METHOD>
      <LINK_POS 200 </LINK_POS>
      <LINK_LEN 91 </LINK_LEN>
      ...
    </FOLLOW_METHOD>
  </PLAN>
</PFP>

```

Fig. 8. An excerpt of the generated PFP for the Amazon site.

Having generated the PFP, the next step is the generation of the OEP. Using the DEByE tool, the user loads a sample page from the set of answer pages returned by the Amazon site and gives examples of the data of interest. An OEP is then generated. Fig. 9 illustrates the excerpt of the OEP generated for the given example. The PFP and the OEP generated in the previous steps are then fed to the generic wrapper which fetches and extracts the data, storing them in an XML format [9] in a repository for further use.

## 4 Conclusion and Future Work

In this paper we discussed WByE, a software environment for fetching and extracting data from Web sites. Its main distinctive feature is to be fully based

```

<?xml version = "1.0"?>
<OBJECTS>
  <TUPLE type="Book">
    <ATOM type="Title">
      <PATTERN>
        <![CDATA[<b>[\s]*?a [^\<]+>[\s]*? ...
      </PATTERN>
    </ATOM>
    <ATOM type="Authors">
      <PATTERN>
        <![CDATA[<dd>[\s]*?([\x20-\x3B\x3D\x3F-\x7E\xA0-\xFF]*?) ...
      </PATTERN>
    </ATOM>
    ...
  </TUPLE>
</OBJECTS>

```

**Fig. 9.** An excerpt of the generated OEP for the Amazon site.

on information implicitly provided by the user by means of suitable and intuitive interfaces based on high level abstractions such as nested tables and directed graphs. WByE includes two components: the ASByE (Agent Specification By Example) tool, used for generating specifications on how to fetch desired pages (be them static or dynamic) from some Web site, and the DEByE (Data Extraction By Example) tool, used for generating specification on how to extract data implicitly present in the fetched pages.

Since the DEByE tool was already introduced in [9, 15], we focused our discussion on the ASByE tool. Thus, we described its main features, presented the results of an experiment we have done with some popular Web sites, and discussed an example of its usage within the WByE environment.

The ASByE tool has been designed having in mind a large class of Web sites that provide huge amounts of data buried into HTML pages, be them static or dynamically generated. Observing the features found in many of these sites, we implemented the ASByE tool to deal with common situations found in them, such as page collections, forms, and page threads. As a consequence, for sites that include features based on technologies such as Java scripts and Java or Flash applets, i.e., browser-processed generic code, the ASByE tool would fail in generating a proper collecting specification.

Compared to other systems such as W3QS, FLORID, ARANEUS, XWRAP, and W4F, ASByE includes some important features that are unique. First, it requires no code writing; second, it includes a navigation metaphor based on an intuitive visual paradigm; third, it supports automatic filling of HTML forms; and fourth, it allows handling page collections and threads of answer pages. These features make ASByE specially suitable for cases when the generated wrapper has to be maintained due to changes in the sites of interest.

Although we have discussed the use of the ASByE tool in the context of wrapper generation, its range of application is very much broader. For example, we are now investigating its use for addressing a very well known problem in general purpose Web crawlers used in search engines, which is the systematic collection of dynamically generated Web pages.

## References

1. 100 HOT. 100 Hot Web Site. <http://www.100hot.com/>.
2. ADELBERG, B. NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Seattle, Washington, 1998), pp. 283–294.
3. ATZENI, P., MECCA, G., AND MERIALDO, P. Semistructured und Structured Data in the Web: Going Back and Forth. *SIGMOD Record* 26, 4 (1997), 16–23.
4. DB&LP. DB&LP's Index to ACM TODS. <http://www.informatik.uni-trier.de/~ley/db/journals/tods/index.html>.
5. EBAY. eBay Web Site. <http://www.ebay.com/>.
6. HAMMER, J., GARCIA-MOLINA, H., NESTOROV, S., YERNENI, R., BREUNIG, M., AND VASSALOS, V. Template-Based Wrappers in the TSIMMIS Experience. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (Tucson, Arizona, 1997), pp. 532–535.
7. HASAN, M., MENDELZON, A., AND VISTA, D. Applying Database Visualization to the World Wide Web. *ACM SIGMOD Record* 25, 4 (1996), 40–44.
8. KONOPNICKI, D., AND SHMUELI, O. Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems (TODS)* 23, 4 (1998), 369–410.
9. LAENDER, A. H. F., RIBEIRO-NETO, B., DA SILVA, A. S., AND SILVA, E. S. Representing Web Data as Complex Objects. In *Proceedings of the First International Conference on Electronic Commerce and Web Technologies EC-Web 2000* (Greenwich, UK, 2000), S. Madria and G. Pernull, Eds., Lecture Notes in Computer Science.
10. LAWRENCE, S., AND GILES, C. Searching the World Wide Web. *Science* 280, 4 (1998), 98–100.
11. LIU, L., PU, C., AND HAN, W. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In *Proceeding of the 16th International Conference on Data Engineering* (San Diego, California, 2000), pp. 611–621.
12. LUDÄSCHER, B., HIMMERÖDER, R., LAUSEN, G., MAY, W., AND SCHELEPPHORST, C. Managing Semistructured Data with FLORID: a Deductive Object-Oriented Approach. *Information Systems* 23, 8 (1998), 589–614.
13. MUSLEA, I., MINTON, S., AND KNOBLOCK, C. A hierarchical approach to wrapper induction. In *Proceedings of the 3rd Conference on Autonomous Agents* (Seattle, Washington, 1999), pp. 190–199.
14. QUASS, D., WIDOM, J., GOLDMAN, R., HAAS, K., LUO, Q., MCHUGH, J., NESTOROV, S., RAJARAMAN, A., RIVERO, H., ABITEBOUL, S., ULLMAN, J. D., AND WIENER, J. L. LORE: A Lightweight Object REpository for Semistructured Data. In *Proceedings of the International ACM SIGMOD Conference on Management of Data* (Montreal, Canada, 1996), p. 549.
15. RIBEIRO-NETO, B., LAENDER, A. H. F., AND DA SILVA, A. S. Extracting Semi-Structured Data Through Examples. In *Proceedings of the Eighth ACM International Conference on Information and Knowledge Management - CIKM'99* (Kansas City, Missouri, 1999), pp. 94–101.
16. SAHUGUET, A., AND AZAVANT, F. Web Ecology: Recycling HTML pages as XML documents using W4F. In *Proceedings of the Second International Workshop on the Web and Databases* (Philadelphia, Pennsylvania, 1999), pp. 31–26.