行政院國家科學委員會專題研究計劃成果報告

Web 文件之資料自動萃取引擎之分析研究
The Study of Automatic Data Extraction Engines for Web Documents

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC90-2213-E-009-121

執行期限：民國 90 年 8 月 1 日至民國 91 年 7 月 31 日

主持人：吳毅成

計畫參與人員：蘇瑞元、洪憲忠、簡光廷、簡廉哲、蔡銘韓

執行單位：國立交通大學資訊工程系

中 華 民 國　九十一　年　十二 月 二十五 日

中文摘要

本計劃將研究有關網頁文件上的資料自動萃取問題。網頁文件的資料萃取涉及兩個問題，一是瀏覽順序，二是資料萃取。許多的網頁並沒有辦法直接以網址來取得。比如許多網站如 104 人力網需要登入才能瀏覽重要資料，有些網站如奇摩站的超連結是經由執行某些程式才會產生。因此在萃取網頁資料前，常常必須瀏覽至所需的網頁後才能做資料萃取。因此網頁間瀏覽的順序相當重要。這個計畫除了解決網頁資料萃取的問題之外，也必須同時解決網頁瀏覽的問題。為了讓網頁資料萃取更有彈性且更易設計，我們設計了一個以 XML 為基礎的新的描述語言，叫做資料萃取服務描述語言(DESDL)，用來描述資料萃取服務中的流覽與資料萃取。在 DESDL 中，一個 script 程式包函一組服務，每一個服務從指定的網頁上萃取資料並處理這些資料。舉例來說，儲存這些資料到資料庫或利用這些資料來瀏覽下一頁。DESDL 使得控制瀏覽順序與資料萃取變的更加容易。簡單的說，DESDL 具有下列的特色。

(1) 使用 XPath 當作萃取網頁內部資料的查詢敘述格式。
(2) 能以特定的順序瀏覽網頁。
(3) 可以填寫表單並啟動下一個服務來萃取提交表單後的下一頁。
(4) 支援外掛程式，這裡稱為 DESDLet 來處理萃取的資料。舉例來說，儲存到資料庫或瀏覽下一頁。
(5) 與目前的瀏覽器的規格一致。
(6) 模擬按一下的動作並啟動下一個服務來萃取下一頁。

在計畫中，我們實作了 DESDL 系統，並實際將之應用在一個比價系統上以證明其功能。

Abstract：

In this paper, we design a new XML-based description language, named Data Extraction Service Description Language (DESDL), for data extraction services. In DESDL, one script includes a set of services each of which extracts data from the designated (web) pages and then processes these data, e.g., stores these data into local databases or browse next pages. DESDL facilitates to control the sequence of pages for data extraction. In brief, DESDL supports the following features:

(1) Use XPath as the format of query expressions to extract data inside pages.
(2) Navigate pages in a sequence.
(3) Fill forms to invoke next services.
(4) Support plug-in code, named DESDLet, to process the extracted data, e.g., store into databases or browse next pages.
(5) Conform to the current browser.
(6) Simulate the clicking action to invoke next services.

In this paper, we also implement the system for DESDL and demonstrate the system by using it to help implement a price-comparison site.

Keyword: data extraction, navigation, DESDL, plug-in, DESDLet

1 Introduction

With the rapid development of World Wide Web (WWW) recently, it becomes more and more important for users to obtain information over Internet. In order to collect useful information over Internet, users first search them from some portal sites, such as Yahoo! [14][24], and then browse by clicking through the related pages. Usually, these users simply want to extract some significant segments inside web pages, instead of retrieving the whole web pages.

Furthermore, users may want to process the browsed web pages (in HTML or XML [7][10][18][19][20]) automatically, such as extracting some products' names and prices into database for later use. Thus, it becomes quite important to support some data extraction tools to help users to automate the process. This is especially important for the following applications. Some price comparison web sites [5] need to automate the process of data extraction by storing product information into database. Some companies need to automate the process of data extraction to retrieve business news regularly.

In order to automate the process of data extraction, many researchers proposed data extraction languages, as in [2][3][4][8][9][11][15][16][17][22][23]. These languages are classified into the following two classes: *query-based* languages and *service-based* languages.

In the query-based languages, users write query expressions to extract data among a set of pages. In general, these languages follow the select-from-where (SFW) structure, as SQL. Most of data extraction languages are query-based, such as XML-QL [4], XQL [15], XQuery [17][21], W3QL [8][9], and WebOQL [2][3] etc. Due to the SFW structure, the page processing sequence is non-deterministic in many query-based languages. For example, when extracting the product prices in page A and the product information in page B, we only need to specify how to match the product information with the same product name in both pages A and B inside the "where" clause. Apparently, we do not have to indicate which page to retrieve first.

includes several services each of which extracts data from one designated web page and then processes the

In the service-based languages, such as WIDL [11] (Web Interface Definition Language), one script extracted data, e.g., stores these data into local databases or makes use of these data to browse next pages. In the service-based languages, since one page needs one service to process, it is inherently required to specify the sequence of pages.

Now, consider the issue of the page sequence. In the languages without specifying the page sequence, we will leave it to the system to choose the best sequence to do. Thus, this may reduce users' efforts of specifying the sequence and may make query expressions simpler or clearer. However, this may also make the extraction system harder to implement, because it has to decide which page to retrieve first. In addition, we also observe that it is often important and necessary to specify the sequence of extracting pages for the following problems:

- The cookie problem.

  Many web sites require that users login with accounts and passwords before allowing users to browse more pages. Usually, the web servers set cookies in the first homepage, so that the servers can recognize the users in next pages. Thus, specifying the page sequence is necessary.

- The problem of security mechanisms.

  Similarly, many web sites encrypt next browsed pages (e.g., using the SSL protocol [6][13]) after users login with accounts and passwords. Thus, specifying the page sequence is also necessary.

- The problem of filling forms and choosing the "GET" or "POST" action.

  In many web pages, users need to fill up the form and then use "GET" or "POST" action to access next pages. Again, this demonstrates the necessity of the page sequence.

- The problem of the referral header options.

  Although many web servers do not make use of this feature, specifying the page sequence is necessary in this case.

In fact, some query-based languages, such as W3QL [8][9], and WebOQL [2][3], also provide users with some features to specify the sequence of pages. However, due to the SFW structures, these languages still have some drawbacks. For example, before extracting data, the system (W3QS) for W3QL needs to collect all navigated pages, many of which may be wasteful. In WebOQL, we cannot fill forms to retrieve next pages. Besides, in an extreme case that we want to extract all the entries in a web site, say Yahoo! [24], we have to create a huge tree in WebOQL. The tree may be so large as to slow down or complicate the processing.

In order to facilitate the better control of the sequence of extracting pages, this paper proposes a new service-based and XML-based language, named a Data Extraction Service Description Language (DESDL). In brief, DESDL supports the following features.

(1) Use XPath [22] as the format of query expressions to extract data or locate elements inside pages.

    In the DESDL language, for each service, we use XPath as the format of query expressions to extract data or locate elements inside the corresponding page. This is because XPath is already a standard of W3C Consortium for extracting data inside pages. The XQuery language is also based on XPath. The WIDL and XML-QL languages use DOM-like expressions. The W3QL language uses regular expressions (in PERL code) to extract data. The WebOQL language uses its own expressions to match the data.

(2) Navigate pages in a sequence.

In the DESDL language, we can navigate pages in a sequence. (Note that the sequence is depth-first, normally.) For example, in DESDL, we can traverse the whole Yahoo! category tree [24] for data extraction. In both XML-QL and XQuery languages, the page sequence is not specified. The WIDL language only allows a service to invoke a *single* next page recursively. Both W3QL and WebOQL languages also supports navigation in some sequences.

(3) Fill forms to invoke next services.

In the DESDL, W3QL, and WIDL languages, we can specify how to fill forms to invoke next services. Others cannot.

(4) Support plug-in code, named DESDLet, to process the extracted data, e.g., store into databases or browse next pages.

In the DESDL languages, users can write DESDLets to help process the extracted data and integrate with other programs. For example, save into database, remove duplicated pages or unnecessary pages, notify users, issue warning messages, etc. In the WIDL language, the role is changed. Other programs can invoke the services (in WIDL) to extract data for the use of these programs. So, in WIDL, page traversal is not so important. In the W3QL, users can use Perl code to help match data. None of other languages supports similar interfaces to other programs.

(5) Conform to the current browsers.

When implementing the data extraction system, it is very important to conform to the current browsers and leverage the current browser objects, such as the Webbrowser control in Microsoft Windows [12]. The reasons are as follows. Since HTML documents are not well formed as XML documents are, the parsing trees may often be ambiguous. Therefore, if our system implements our own parsers, it is most likely that the parsing trees are different from those in browsers. Thus, the users may have difficulty to locate elements due to different parsing tree, and it is hard for our system to support visualization tools in the future. Another reason is described in the following item. None of the other languages supports this feature.

(6) Simulate the clicking action to invoke next services.

In the DESDL language, we can first locate the elements using XPath expressions and then invoke next services by simulating the action of clicking on these elements. The major advantage of this feature is that many web pages invoke next pages via Javascript function calls. Since we leverage the current browsers (as described in the previous item (5)), we can easily do this. Otherwise, there is almost no general way to invoke next services in such cases. None of the other languages supports this feature.

	XML-QL	XQuery	W3QL	WebOQL	WIDL	DESDL
Targeted documents	XML	XML	HTML + XML	HTML + XML	HTML + XML	HTML + XML
Locating elements	DOM-like	XPath	PERL code	Its own expressions	DOM-like	XPath
Navigation	Not sequenced	Not sequenced	Depth-first	Depth-first and Breadth-first	Linear	Depth-first
Filling Form	No	No	Yes	No	Yes	Yes
Integrating programs	None	None	PERL code	None	IDL for other programs	Plug-ins
Conforming to browsers	No	No	No	No	No	Yes
Simulating clicking	No	No	No	No	No	Yes

From above, we summarize the comparisons among these languages in Table 1.

In this paper, we also implement the system for DESDL and demonstrate it by using it to implement a price-comparison site where we extract product information from tens of electronic-commerce sites in Taiwan. From our empirical experiences, one programmer only needs about one working day to write a DESDL script to extract product information from one electronic-commerce (or e-commerce) web site. This greatly reduces the overhead of maintaining such a web site.

In this paper, Section 2 describes the specification of our DESDL language. Section 3 illustrates our DESDL system by a real application for the price comparison. Section 4 makes concluding remarks.

2 DESDL

In this Section, we will describe the DESDL language. A DESDL script is enclosed by the element `<DESDL>`. This element contains two types of elements: The element `<INIT>` designates the URL of the initial web page (in the attribute `URL`) and the initial service (in the attribute `SERVICE`) associated with the page. The element `<SERVICE>` is used to specify how to extract data from the associated pages and process these data.

```
. . .
  <td><b><a href="http://...">
          DESDL: A Data Extraction Service
          Description Language</a></b>
      <b>I-Chen Wu, Jui-Yuan Su, . . . </b>
      <i>Proceedings of . . .</i>
  </td>
  <td><b><a href="...">
          WebOQL: Restructuring Documents,
          Databases, and the Web </a></b>
      <b> Gustavo Arocena and Alberto Mendelzon.
      </b>
      . . .
  </td>
. . .
```

Figure 1: A sample HTML file.

```
<DESDL>
  <INIT SERVICE="GetPaperAttr" URL="..."/>
  <SERVICE NAME="GetPaperAttr"/>
    <VAR NAME="Title" PATH="//b[0]//text()"/>
    <VAR NAME="Author" PATH="//b[1]/text()"/>
  </SERVICE>
</DESDL>
```

Figure 2: A DESDL script to extract data from the HTML file in Figure 1.

For example, Figure 2 shows a DESDL script to extract data from a sample HTML document in Figure 1. The DESDL system initially loads a web page at the `URL` specified in the element `<INIT>`, and then uses the service `GetPaperAttr` (specified in the element `<INIT>` too) to process the page. In the service `GetPaperAttr`, the elements `<VAR>` define the names and values of the variables.

Query Expressions

In the DESDL language, for each service, we use XPath as the format of query expressions to extract data or locate elements inside the associated page. The attribute `PATH` inside the element `<VAR>` contains an XPath expression that locates the data with the variable value. For example, in the script in Figure 2, the XPath expression of the variable `Title`, `//b[0]//text()`, means to locate the first element with tag b and then extract the text inside the element. By extracting data from the document in Figure 1, we obtain the contents of the variables, `Title` and `Author`, as "`DESDL: A Data Extraction Service Description Language`" and "`I-Chen Wu, Jui-Yuan Su, . . .`", respectively.

Navigation

In DESDL, when extracting data from a page in a service, we can also invoke next services by using the element `<INVOKE>`, which includes two important attributes: `SERVICE`, the name of next service, and `URL`, the URL of the next page to request.

Furthermore, in DESDL, we can invoke multiple services from one service. For example, after extracting a set of URLs from the associated page, we can continue to access multiple pages at these URLs. In order to access multiple pages, the element `<SERVICE>` may contain the element `<FOREACH>`. In the element `<FOREACH>`, the attribute `FROM` indicates an array variable evaluated from an XPath expression earlier. In the DESDL system, each array element is processed by the script inside `<FOREACH>` once, as follows.

```
<DESDL>
  <INIT SERVICE="GetPaperAttr" URL="..."/>
  <SERVICE NAME="GetPaperAttr">
    <VAR NAME="SearchBase"
         PATH="//table/tr/td"/>
    <FOREACH FROM="$SearchBase">
      <VAR NAME="Title" PATH="b[0]//text()"/>
      <VAR NAME="Link" PATH="b[0]/a/@href"/>
      <VAR NAME="Author" PATH="b[1]/text()"/>
      . . .
      <INVOKE SERVICE="GetAbstract"
              URL="$Link"/>
    </FOREACH>
  </SERVICE>
  <SERVICE NAME="GetAbstract">
    ...
</DESDL>
```

Figure 3: An example of using multi-way navigation.

For example, in Figure 3, the variable `SearchBase` contains an array of all rows (each of which corresponding to a paper in Figure 1) in tables, and, therefore, the DESDL system will process all elements inside `<FOREACH>` for each row. Then, for each table row (or product), extract the title, the link (to the page with product information) and the authors into the variables `Title`, `Link`, and `Author`, respectively; and invoke a new service `GetAbstract` with the page at the URL `$Link`, which means the value of the variable `Link`. For a portal site with a tree of category, we can easily use this way to navigate the whole category tree. We will describe an example of navigation through the whole category tree in Section 3. Our navigation is in the depth-first ordering.

Filling Forms and Clicking

In the DESDL language, we can specify how to fill forms or click to invoke next services.

```
. . .
<Script language=Javascript>
   function Directto(){. . .
      window.open("http://www.csie.nctu ...")
      . . . }
</Script>
<FORM NAME="Form1" ACTION="results.cfm?...
   METHOD=POST onSubmit="Directto()">
   <b>Search DL</b>
   <INPUT TYPE="Text" NAME="query" VALUE="">
   <INPUT TYPE="Submit" NAME="Go">
</FORM>
. . .
```

Figure 4: A sample HTML file with forms.

Consider a sample HTML file with forms in Figure 4. In both WIDL and W3QL languages, we can fill forms to invoke next services. However, none of other languages can process the clicking action that makes a Javascript function call as shown in Figure 4. In DESDL, we can fill forms while simulating the clicking action as shown in Figure 5.

```
<DESDL>
  <INIT URL="http://www.csie.nctu.edu.tw/..."
        SERVICE="ExtractPaperInfo"/>
  <SERVICE NAME="ExtractPaperInfo">
    <VAR NAME="FormBase" PATH="...//FORM"/>
    <FOREACH FROM="$FormBase">
       <FORMPARAM NAME="query" VALUE="Web
                Query" PATH="INPUT[0]"/>
       <INVOKE SERVICE="GetPaperAttr"
              PATH="INPUT[1]" />
    </FOREACH>
  </SERVICE>
   . . .
</DESDL>
```

Figure 5: A DESDL script to fill up form in Figure 4.

In Figure 5, we fill up the form parameters by using `<FORMPARAM>`. In this form parameter element, named `query`, we fill the value "`Web Query`" into the text field at the XPath `INPUT[0]` based on each array data of `FROM`. Then, we invoke the next service `GetPaperAttr` with the page returned by clicking the partial XPath `INPUT[1]`, that is the `INPUT` with name `GO`.

DESDLets

In our research, we find that many users may want to modify or slightly modify the behavior of the `invoke` operations. For example, save into database, remove duplicated pages or unnecessary pages, notify users, issue warning messages, etc. Therefore, we support a mechanism, named DESDLets, that allows users to change the behavior at `<INVOKE>`. In DESDL, the element `<INVOKE>` contains an optional attribute `DESDLET`, that specifies the DESDLet routine to process the invocation.

For example, we add one line (bolded) into the script in Figure 3, as shown in Figure 6. In this line, the attribute `DESDLET` indicates the program `RemoveRepeated.dll` as the invocation routine, which removes the access to duplicated pages.

The DESDL system passes three parameters into DESDLet: the set of variables that have been specified (e.g., `Title` and `Link` in Figure 6), the service name (e.g., `GetAbstract`), and the URL or path to be invoked (e.g., the content of `URL`). Then, DESDLets process these data. For example, DESDLets can store the extracted values into databases, or check to avoid duplication in navigation if the URL has been invoked earlier. Finally, DESDLets can decide whether to invoke the service specified in the `<INVOKE>` element to load next page and call the next service.

```
<DESDL>
  <INIT SERVICE="GetPaperAttr" URL="..."/>
  <SERVICE NAME="GetPaperAttr">
    <VAR NAME="SearchBase"
         PATH="//table/tr/td"/>
    <FOREACH FROM="$Searchbase">
      <VAR NAME="Title" PATH="b[0]//text()"/>
      <VAR NAME="Link" PATH="b[0]/a/@href"/>
      . . .
      <INVOKE SERVICE="GetAbstract"
              DESDLET="RemoveRepeated.dll"
              URL="$Link"/>
    </FOREACH>
  </SERVICE>
   . . .
</DESDL>
```

Figure 6: An example of using DESDLet.

Currently, there are some DESDLets implemented in the system. The first is to check duplicated URLs. The hash table is implemented in the DESDLets to store the URL that the system has been navigated. DESDLets check the URLs in the hash table to ensure that the same pages will not be accessed more than once. If the URLs are not accessed before, DESDLets insert the URLs into the hash table. The second is to store data into database. The data are processed and then save into database. The third is to log records. Specific records are logged for analysis and tracking.

In the next section, we illustrate our DESDL system by a real application for price comparison.

3 Application: Price Comparison

Based on DESDL, we implemented a price comparison system that extracts price information of products from some e-commerce web sites. The categories of e-commerce web sites include those for selling communication materials, computer hardware and software products, book stores, movies, music disks, daily necessities, etc. In this system, one script of DESDL is written for extracting data in each site. The DESDL system parses the script, traverses related pages, extracts related information (such as prices) of products from these web sites, and saves them into database.

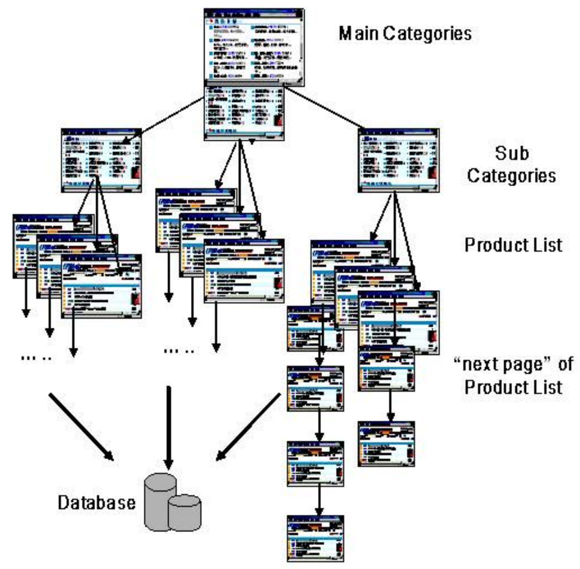

Figure 7: The page hierarchy of an e-commerce web site.

In the rest of this Section, we will use the bidding web site [5], www.ubid.com.tw (abbr. UBID below), as an example to explain how the DESDL scripts are used to extract data. Figure 7 shows the page hierarchy of this web site.

The web page model of UBID is described as follows. The homepage of UBID contains links to several main categories. Each main category contains links to several subcategories. Each subcategory links to product list page. If there are a lot of products in one subcategory, the product list page will contain a link to next page of product list. The formats of product list page and "next page" of product list page are the same.

```
<DESDL NAME="UBID">
  <INIT SERVICE="MainCategory" ... />
  <SERVICE NAME="MainCategory">
    <FOREACH ...>
       ...
       <INVOKE SERVICE="SubCategory" ... />
    </FOREACH>
  </SERVICE>
  <SERVICE NAME="SubCategory">
    <FOREACH ...>
       ...
       <INVOKE SERVICE="ProductList" ... />
    </FOREACH>
  </SERVICE>
  <SERVICE NAME="ProductList">
       ...
  </SERVICE>
</DESDL>
```

Figure 8: A DESDL script for the web site UBID.

There are three services defined in this example, `MainCategory`, `SubCategory` and `ProductList`, as shown in Figure 8. The `MainCategory` service loads the initial main category page at the URL of the `<INIT>` element, extracts the URLs of the subcategory pages, and then invokes the next pages via these URLs with the `SubCategory` service. Then, the `SubCategory` services load the subcategory pages, extract the URLs of the product list pages, and then invokes the next pages via these URLs with the `ProductList` service.

```
<SERVICE NAME="MainCategory">
  <VAR NAME=HyperLink PATH=".…./a"/>
  <FOREACH FROM=$HyperLink>
     <INVOKE SERVICE="SubCategory"
            PATH="@href"/>
  </FOREACH>
</SERVICE>
```

Figure 9: The MainCategory service.

The `MainCategory` service is shown in greater detail in Figure 9. First, it extracts all hyperlinks to the subcategory into the variable `HyperLink`, and then uses `FOREACH` to access each subcategory page.

```
<SERVICE NAME="SubCategory">
  <VAR NAME=HyperLink PATH=".…./a"/>
  <FOREACH FROM=$HyperLink>
     <INVOKE SERVICE="ProductList"
              PATH="@href"/>
  </FOREACH>
</SERVICE>
```

Figure 10: The SubCategory service.

Next, similarly, the `SubCategory` service, as shown in Figure 10, loads the pages that are invoked by `MainCategory` service, extracts the URLs of the product list pages, and then invokes the next pages via these URLs with the `ProductList` service.

```
<SERVICE NAME="ProductList"/>
  <VAR NAME="Products" PATH=".…."/>
  <FOREACH FROM=$Products>
     <VAR NAME="Title" PATH="td[0]/text()"/>
     <VAR NAME="Info" PATH="td[1]/text()"/>
     <VAR NAME="Price" PATH="td[2]/text()"/>
     <INVOKE DESDLET="save2db.dll"/>
    <!--Save into DB>
  </FOREACH>
  <VAR NAME="HyperLink" PATH="//a['next']"/>
  <FOREACH FROM=$HyperLink>
     <INVOKE SERVICE="ProductList"
            PATH="@href"/>
  </FOREACH>
</SERVICE>
```

Figure 11: The ProductList service.

Finally, the `ProductList` service, as shown in Figure 11, loads the pages, extracts the URL of the next product list page, and then may invoke the pages via these URLs with the `ProductList` service, if the next page exists. More importantly, the `ProductList` service also extracts the data of each product in the page and saves into the database.

4 Discussions and Conclusion

In this paper, we design an XML-based description language, named DESDL, for data extraction services. In DESDL, the users can describe a set of services each of which extracts data from the designated web pages and then saves these data into local databases or navigates pages. The features of DESDL are summarized as follows.

(1) Use XPath as the format of query expressions to extract data or locate elements inside pages.

(2) Navigate pages in a sequence.

(3) Fill forms to invoke next services.

(4) Support plug-in code, named DESDLet, to process the extracted data, e.g., store into databases or browse next pages.

(5) Conform to the current browsers.

(6) Simulate the clicking action to invoke next services.

We have already implemented the system for DESDL and demonstrated it by implementing a price-comparison site where we extract product information from tens of e-commerce sites in Taiwan. From our experiences, one programmer only needs one working day to write a DESDL script to extract product information from one E-commerce web site. This greatly reduces the overhead of maintaining such a web site. We deeply believe the DESDL system can be used in many practical Internet applications in the future.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China for the support under grant NSC 90-2213-E-009-121.

計劃成果自評

本計劃對網頁資料的自動萃取所需的瀏覽與資料萃取問題，提出一套較有彈性

的解決方法，利用一個以 XML 為基礎的資料萃取服務描述語言來描述瀏覽過程與萃取的資料位置。DESDL 使得控制瀏覽順序與資料萃取變的更加容易。DESDL 具有下列的特色。

(1) 使用 XPath 當作萃取網頁內部資料的查詢敘述格式。
(2) 能以特定的順序瀏覽網頁。
(3) 可以填寫表單並啟動下一個服務來萃取提交表單後的下一頁。
(4) 支援外掛程式，這裡稱為 DESDLet 來處理萃取的資料。舉例來說，儲存到資料庫或瀏覽下一頁。
(5) 與目前的瀏覽器的規格一致。
(6) 模擬按一下的動作並啟動下一個服務來萃取下一頁。

此外，我們實作了 DESDL 系統，並實際將之應用在一個比價系統上以證明其功能。

本計畫成果已發表於 ICS 2002。

References:

[1] Association for Computing Machinery. “ACM Portal to Computing Literature”, ACM, New York, 2002. http://portal.acm.org/portal.cfm.

[2] Gustavo Arocena and Alberto Mendelzon. “WebOQL: Restructuring Documents, Databases, and the Web”, In *Proceedings of ICDE*, 1998, Orlando, Florida.

[3] G. Arocena. “WebOQL: Exploiting Document Structure in Web Queries”, Master's Thesis, University of Toronto, 1997.

[4] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. “XML-QL: A Query Language for XML”, In *Proceedings 8th International World Wide Web Conference (WWW8)*, 1999. Computer Networks 31(1116) : 1155-1169.

[5] Ebay Inc., "ebay: The World’s Online Marketplace", 2002. http://www.ubid.com.tw.

[6] Alan Freier, Philip Karlton, Paul Kocher. “The SSL Protocol Version 3.0”, Internet Draft, Mar. 1996. http://wp.netscape.com/eng/ssl3/ssl-toc.html

[7] Steve Holzner. “XML Complete”, McGraw-Hill, 1998.

[8] D. Konopnicki, O. Shmueli. “W3QL: A Query System for the World Wide Web”, in *Proceedings of the 21th International Conference on Very Large Databases*, Zurich, 1995.

[9] David Konopnicki , Oded Shmueli. “Information gathering in the World-Wide Web: the W3QL query language and the W3QS system”, *ACM Transactions on Database Systems (TODS)*, Volume 23 Issue 4, Dec. 1998.

[10] Sean Mcgrath. “XML by example: building E-commerce applications”, Prentice Hall PTR, 1998.

[11] Phillip Merrick, Charles Allen. “Web Interface Definition Language”, W3C NOTE, Sep. 1997. http://www.w3.org/TR/NOTE-widl.

[12] Microsoft Corporation. “WebBrowser Control”, Programming and Reusing the Browser, MSDN Library, 2002. http://msdn.microsoft.com/library/default.asp?url=/workshop/browser/webbrowser/browser_control_node_entry.asp.

[13] Netscape Communications Corporation. “Secure Sockets Layer”, 2000. http://wp.netscape.com/security/techbriefs/ssl.html.

[14] Openfind Information Technology, Inc. "Openfind Enterprise Portal Technology Provider", 2002. http://www.openfind.com.tw.

[15] Jonathan Robie, Joe Lapp, David Schach. “XQL：XML Query Language”, Workshop on XML Query Languages, Dec. 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[16] W3C Consortium, “XML Query”, Apr. 2000. http://www.w3c.org/XML/Query.

[17] W3C Consortium. “XQuery 1.0: An XML Query Language”, W3C Working Draft, 16 Aug. 2002. http://www.w3.org/TR/xquery/.

[18] W3C Consortium. “Extensible Markup Language (XML) 1.0 (Second Edition)”, W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[19] W3C Consortium. “Hyper Text Markup Language”, Jan. 1998. http://www.w3c.org/Markup/.

[20] W3C Consortium, “HTML 4.01 Specification” W3C Recommendation, Dec. 1999. http://www.w3.org/TR/html4/.

[21] W3C Consortium. “XQuery 1.0 and XPath 2.0 Data Model”, W3C Working Draft, Aug. 2002. http://www.w3.org/TR/query-datamodel/.

[22] W3C Consortium. “XML Path Language (XPath) 2.0”, W3C Working Draft, Aug. 2002. http://www.w3.org/TR/xpath20/.

[23] W3C Consortium. "XML Pointer Language (XPointer) Version 1.0", W3C Working Draft, Aug. 2002. http://www.w3.org/TR/xptr/.

[24] Yahoo! Inc, "Yahoo Search Engine", 2002.
http://www.yahoo.com.