ELSEVIER

Computer Standards & Interfaces 24 (2002) 291–309

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

Supporting unified interface to wrapper generator in Integrated Information Retrieval

Yue-Shan Chang [a], Min-Huang Ho [b,*], Wen-Chen Sun [c], Shyan-Ming Yuan [b]

[a] *Department of Electronic Engineering, Minghsin Institute of Technology, Taiwan*
[b] *Department of Computer and Information Science, National Chiao Tung University, 1001 Ta Hsueh Road, Hsin-Chu 300, Taiwan*
[c] *Embedded System Lab, Institute for Information Industry, Taiwan*

Received 6 October 2001; received in revised form 5 December 2001; accepted 9 February 2002

Abstract

Given the ever-increasing scale and diversity of information and applications on the Internet, improving the technology of information retrieval is an urgent research objective. Retrieved information is either semi-structured or unstructured in format and its sources are extremely heterogeneous. In consequence, the task of efficiently gathering and extracting information from documents can be both difficult and tedious. Given this variety of sources and formats, many choose to use mediator/wrapper architecture (Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. Ullman, A Query Translation Scheme for Rapid Implementation of Wrappers, International Conference on Deductive and Object-Oriented Databases, Singapore, 1995), but its use demands a fast means of generating efficient wrappers. In this paper, we present a design for an automatic eXtensible Markup Language (XML)-based framework with which to generate wrappers rapidly. Wrappers created with this framework support a unified interface for a meta-search information retrieval system based on the Internet Search Service using the Common Object Request Broker Architecture (CORBA) standard. Greatly advantaged by the compatibility of CORBA and XML, a user can quickly and easily develop information-gathering applications, such as a meta-search engine or any other information source retrieval method. The two main things our design provides are a method of wrapper generation that is fast, simple, and efficient, and a wrapper generator that is CORBA and XML-compliant and that supports a unified interface. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: XML; Information retrieval; Wrapper generation; CORBA

1. Introduction

1.1. Background and motivation

The scale and diversity of documents and information on the Internet nowadays is immense. Generally, search engines are the means of retrieving desired information, but no single search engine is up to the scale of the task [12,13]. For this reason, for extensive searches, a meta-search engine is the method usually preferred [1,2]. However, the format of most retrieved documents from the Internet is either semi-structured or unstructured. Given their great number and variety, a mechanism that can perform the tasks of gathering and classifying such heterogeneously sourced documents automatically is urgently needed.

To efficiently handle documents in such quantity and variety, many use mediator/wrapper architecture

* Corresponding author. Tel.: +886-6-572-2201; fax: +886-3-559-1402.
E-mail address: minhuang@ms4.hinet.net (M.-H. Ho).

0920-5489/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0920-5489(02)00016-8

with a meta-search system [20]. In this architecture, the mediator plays a role of mediating single common or standard format queries submitted from the user to specific wrappers for specific information sources. These wrappers have the task of wrapping queries into appropriate formats for those sources and of translating the results from them into specific structures for the user. That is, a mediator accepts a user query and dispatches it to a specific wrapper for a specific information source. The wrapper then translates the query into an appropriate query string or command for the specified Internet or Intranet information source. Upon receiving the result from the source, the wrapper extracts the desired information and returns it to the mediator. Each information source connected to the information retrieval system has its own wrapper. The source codes of the wrapper are tightly coupled with the format or structure of the source. That is, if the format of the information source is changed, then the source codes of the wrapper must also be changed. Many information sources constantly change content and format and new information sources are continually added. To overcome these problems, the information retriever requires an automatic framework for wrapper generation.

Many recent attempts to solve information retrieval problems with mediator/wrapper architecture have been made [1–9,11]. Most of them have focused on analyzing and translating heterogeneously sourced documents retrieved by wrappers. Many have also proposed automatic or semi-automatic frameworks to generate distinct wrappers for handling this [3–9,11]. The effort has mainly gone into designing wrappers to translate the returns into a specific representation for queries from the mediator. In fact, for the retrieval application developer faced with multiple information sources, it is important for the available retrieval applications to have a uniform programming interface. It is for that fact that we propose our Integrated Information Retrieval methodology with a unified interface, as shown in Ref. [18]. The flexible architecture here has a unified programming interface and an information retrieval application for querying a variety of sources. We use an Integrated Information Retrieval (IIR) service based on Common Object Service Specification (COSS) of Common Object Request Broker Architecture (CORBA). The metadata of the sources is defined by Document Type Definition (DTD) of eXtensible Markup Language (XML). With this system, an information retrieval developer can easily design applications or agents to collect desired information via a high-level uniform programming interface.

The proposed architecture is ideal for the information retrieval task. However, because of the multiple sources, a supportive framework is necessary. In addition, the framework must ensure that information retrieval application developers can generate wrappers that are simple and fast and that are both XML and CORBA-compliant. Being XML-compliant enables data exchange between different information sources, and being CORBA-compliant enables communication between heterogeneous systems. With practice, employing this framework in an SQL-like high-level query scheme, the user or the client program (e.g., an information retrieval application) can perform the extraction from a variety of sources.

1.2. Related works

As mentioned earlier, a wrapper is tightly associated with an information source. An information retrieval application has to prepare many different wrappers for each information source. In addition, the content and structure of the documents of information sources change constantly. The information retriever needs a fast and simple methodology for generating wrappers. Among the available wrapper-generating methods are TSIMMIS [7,8,20], XWRAP [4], W4F [11], JEDI [9], as well as those in Refs. [3,5,6,15,16]. The first, TSIMMIS, is a logical template-based approach for wrappers for specified information sources. The authors of that method provided a fast acting toolkit for generating wrappers from a set of high-level descriptions for information extraction. That is, the wrapper implementer must specify a set of rules written in a distinct high-level declarative language.

The next, XWRAP, is an XML-enabled wrapper construction framework. It provides a user-friendly interface environment for generating appropriate wrappers in a semi-automatic way. It allows users to enter a URL, upon which the system automatically analyzes and generates rules for retrieving remote documents. These are then checked for syntax and requisite parse trees for semi-structured documents are generated. With the implementer's help, the system creates information extraction rules. Finally, the rules and the

source-specific meta-data provided by the wrapper developer are used to generate an executable wrapper program for the given information source. The files provided by the wrapper implementer are all described using the XML template-based extraction specification language of XWRAP. The main use of XWRAP is for web documents (especially in HTML format).

The W4F method is a complete wrapper toolkit. First, it uses a set of retrieval rules to define the interface between the wrapper and the information source in the Retrieval Layer. Second, it uses an HTML Extraction Language (HEL) to express the desired data field in the Extraction Layer. Any retrieved information is stored in Nested String Lists (NSL) format. Finally, an XML template file in Mapping Layer uses querying operators to map the information to XML.

JEDI is a meta-data system for mediator/wrapper architecture. It supports an attributed-grammar-based extraction language to express the desired data fields of a document. It also provides a fault-tolerant parser to extract data from an information source by a specified grammar. Any wrapper it creates extracts data from the information source and maps it to an XML file.

1.3. Our approach

Most of the above-mentioned wrapper development methods have difficulty with designing query and extraction rules because a good knowledge of web documents and of the syntax of rules is required. Wrapper implementers find designing rules such difficult and tedious. In many systems on the Internet, the returned information is designed for user, not for a program. In addition, a wrapper is an important software component between the information retrieval system and the information source. A well-defined wrapper with a uniform communication interface improves the performance of a heterogeneous information retrieval system. However, writing this kind of wrapper increases the workload of the wrapper programmer. Our solution to that problem is an automatic generating framework for an XML-based wrapper with a CORBA-based unified interface. With this framework, the XML data model is used to express the meta-data of information sources, and the output file of the results is also in XML format. CORBA is an open system model that supports communication between the software components within distributed environments and is used to define the uniform interface for the meta-search system proposed in Refs. [1,18]. With this framework, an information retrieval application can use the CORBA standard to communicate with a variety of wrappers and acquire the results based on a standardized object model. In addition, because XML is now a popular standard for representing and exchanging data, the wrapper programmer has no need to learn a new extraction language to generate wrappers.

The rest of the paper is organized as follows. Section 2 presents a brief overview of the unified interface meta-search system. Section 3 introduces the architecture of the framework. In Section 4, we explain how the system operates. An example of a generated wrapper is demonstrated in Section 5, followed by discussion in Section 6 and conclusions in Section 7.

2. Overview of wrapper

2.1. Wrapper architecture

A wrapper is a software component that embraces an information source. Its main objective is to be the interface between the client program and information source. Because of the heterogeneous information sources, it is best to support a wrapper with a uniform interface. A typical uniform interface wrapper has three tasks. First, it receives user requests from a mediator and then translates them into a query string format (typically into URL form for web information sources) or into a query command (typical in the RDBMS system) acceptable to the information source. Second, it retrieves Internet or the Intranet documents from which it extracts desired information according to extraction rules provided by the user. Last, it stores the desired information in a specific form and provides an interface to allow a user or client to retrieve data in a high-level and structured way.

The typical architecture of a wrapper supporting for Ref. [18] is shown in Fig. 1. It has seven components: (1) *Query String Translator*, (2) *Parameter Encapsulator*, (3) *Document Parser*, (4) *Information Extractor*, (5) *Result Packer*, (6) *Network Transmission Interface*, and (7) *Server Skeleton Interface*. Each component is a stand-alone Java class, and can be

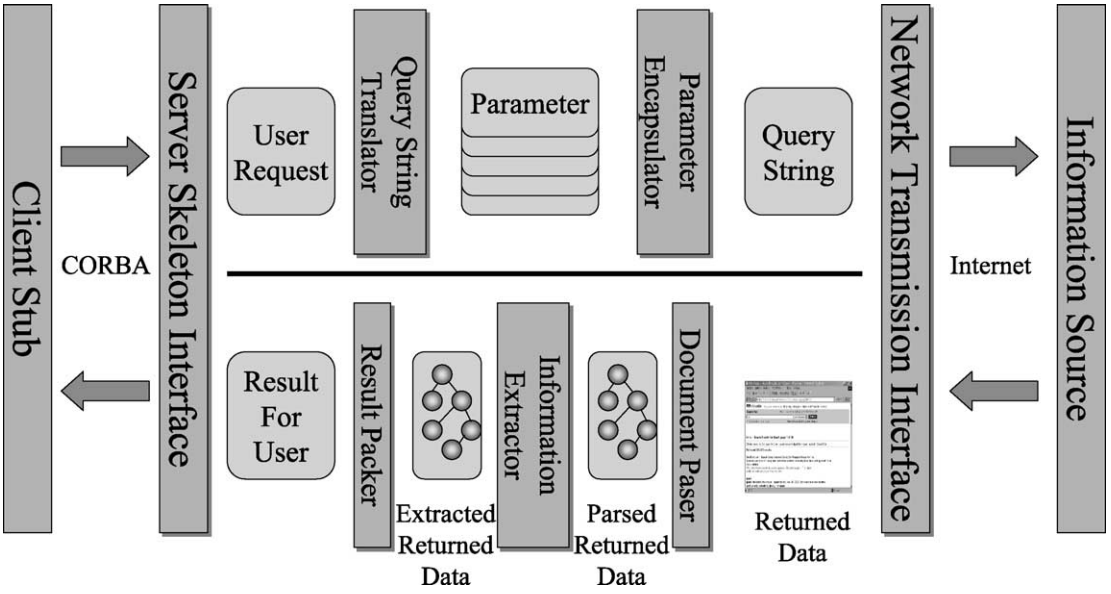


Fig. 1. Basic architecture of XML-based wrapper.

developed and replaced independently. All the data passing through these components are created as XML DOM objects. Consequently, the wrapper uses standard DOM API to develop the application. Because XML is a structured and meaningful data format, each component easily understands the content of the received data and treats it appropriately in an explicit and precise way. Our automatic framework for wrapper generation uses the advantages of XML.

We now discuss the architecture of the XML-based wrapper shown in Fig. 1 in more detail. On the right of the figure is *Network Transmission Interface*, which is the gateway between the wrapper and the information source. Its task is to provide an interface such that the wrapper can send the appropriate query string to the individual information source and then retrieve the results through the interface. On the left is *Server Skeleton Interface*, which is the interface mapped to the stub of the client program based on CORBA standard. It follows that the interface defined by IDL is such that the wrapper receives through the Server Skeleton the SQL-like request (e.g., *SELECT URL FORM ALTAVISTA WHERE KEYWORD=wrapper*) from the Client Stub of the client application. The *Query String Translator* has the task of parsing the user request, extracting such parameters as *URL*, *ALTAVISTA*, *KEYWORD=wrapper* and packing them into a DOM object. It then passes the object to *Parameter Encapsulator*, the main task of which is to encapsulate the parameters into an appropriate query string and send that to a specified search engine or information source. For example, the base query string for AltaVista (http://www.altavista.com) is in the form of http://www.altavista.com/cgi/query?q=DT. . ., and all the parameters (e.g., KEYWORD=wrapper) are appended to that string in the appropriate way. The query string with parameters is sent by *Network Transmission Interface* to the specified search engine (e.g., AltaVista). Next, *Network Transmission Interface* retrieves the results and passes the data to *Document Parser.*

Document Parser then parses the content into an XML parse tree. *Information Extractor* extracts the desired fields according to user-defined extraction rules. Obviously, four major fields (URL, TITLE, DESCRIPTION, and RELATED PAGE) are given in the returned AltaVista document, and *Information Extractor* consequently extracts from it any URL that contains the keyword ‘wrapper’. Finally, *Results Packer* packs into standard format all the information needed by the user into the Client Stub that is generated by CORBA IDL compiler.

2.2. Wrapper's interface

To support the uniform interface of the integrating information retrieval environment proposed in Ref. [18], the generated wrappers have to be reconciled with Integrated Information Retrieval (IIR) interfaces. There are five IIR interfaces: *InformationRetriever*, *Wrapper*, *MetaData*, *Collector*, and *Iterator*. The first of these is explained as follows:

```
Interface InformationRetriever
{
 MetaData Get_meta(in QuerySourceType qsType);
 Wrapper prepare(in ParameterList pl,    in QuerySourceName qsName,
                                         in QueryLanguageType qlType);
}
```

The interface of *InformationRetriever* has two methods, *Get_meta()* and *prepare()*. The purpose of *Get_meta()* is to obtain a object reference of *MetaData* object. The purpose of *prepare()* is to prepare a query request with appropriate parameters for a specified information source and then to obtain an object reference of Wrapper object to start the query process. The *Wrapper* interface is explained as follows:

```
Interface Wrapper
{
 Collector Query() raises(QueryProcessingError, QueryInvalid);
}
```

The interface of *Wrapper* allows a mediator to start the retrieval process by invoking the *Query()* method of the *Wrapper* interface. *InformationRetriever* then dispatches an appropriate wrapper to handle a specified information source according to the information obtained from *Get_meta()*. The interface of *MetaData* is explained as follows:

```
Interface MetaData
{
   Boolean QL_Available(in QuerySourceType qsType);
   Boolean Registry(in QuerySourceMeta metadata);
   Boolean Unregistry(in QuerySourceName qsName);
   Boolean Replace(in QuerySourceName qsName, in QuerySourceName metadata);
   QuerySourceMeta get(in QueryLanguageType ql_type);
}
```

The interface of *MetaData* provides data retrieval robustness while retrieving any information source. By the appropriate assignment of each field of *MetaData*, the client can obtain the format of a query request and the schema of the result. For the detail of *MetaData*, refer to Ref. [18].

The last two interfaces, *Collector* and *Iterator*, are responsible for collecting information from sources. They are explained on the opposite page:

```
Interface Collector
{
  MetaData GetMeta(in QueryLanguageType qsType);
    Readonly attribute long Result_size;
    Result retrieve_element_at(in long where);
    Iterator create_iterator();
}

Interface Iterator
{
  Result next();
    Boolean reset();
    Boolean more();
}
```

As may now be seen, a client program has two methods of retrieving desired data. The first is random retrieval, which uses *Result_size()* and *retrieve_element_at()*. The second is sequential retrieval, using *create_iterator()*. This last creates an *Iterator* object, which provides a simple interface for data retrieval. For detailed information about the unified interface integrating information environment, refer to Ref. [18].

3. System architecture of an automatic wrapper generator

3.1. System overview

To generate a wrapper for a specified search engine or information source correctly, the wrapper implementer must provide adequate information to the wrapper generator in the following forms: *Query Template* file, *Pattern* file, and *Scheme* file. The role of each in our wrapper generation framework is shown in Fig. 2. Within the framework, wrapper generation has three phases: (1) *Query Translation*, (2) *Documents Retrieval*, and (3) *Results Translation*.

In Fig. 2, we consider the developed information retrieval application. The application communicates to the wrapper according to the interface defined by IDL files. In the *Query Translation* phase, the wrapper translates the user's high-level SQL-like queries from the application via the CORBA standard into an acceptable format for the specified search engine or information source, for both of which, the wrapper generator must also know the query command patterns. The wrapper implementer in the QUERYTEMPLATE file must provide the query format.

In the *Document Retrieval* phase, using the QUERY TEMPLATE file, the wrapper attempts document retrieval from a specified information source by the appropriate query string or command. If the information provided by user is correct and the network connection is good enough, the desired documents is quickly retrieved into local data storage. Most returned documents from information sources are either semi-structured or unstructured. One of the responsibilities of the wrapper generation in this phase is to parse documents, extract desired information, and then store it into XML DOM objects. The wrapper implementer must provide in advance a PATTERN file that regulates the handling of the specified data fields for the document parsing process. In addition, for proper management of the data in the specified fields, detailed information about the data type and variable name of the user-interested fields has to be provided in the SCHEMA part of the PATTERN file. The QUERYTEMPLATE, PATTERN, and SCHEMA files are all written in XML syntax.

Following the *Document Retrieval* phase, the user-interested information is stored in the data structure of a wrapper in the form of XML DOM objects. In the *Result Translation* phase, obtaining the information from these objects via the CORBA standard is fast

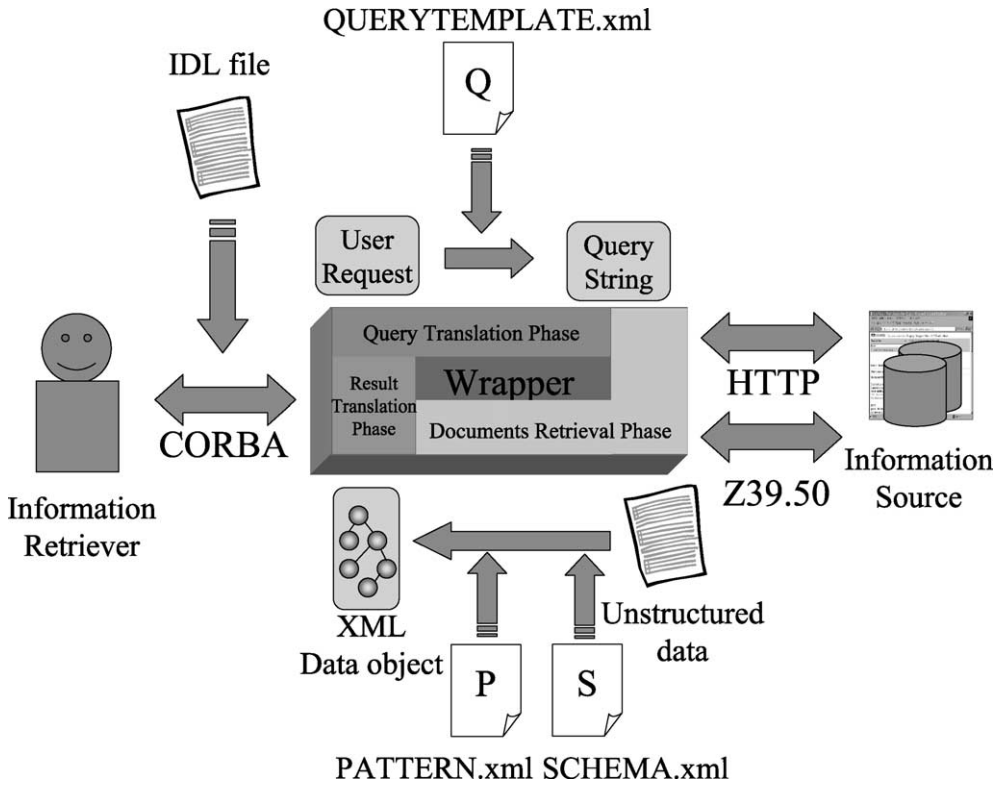


Fig. 2. An overview of the wrapper generator.

and simple for the client program. One of the responsibilities of the wrapper in this phase is to prepare the results extracted from DOM objects in appropriate format and send them back to the client program via CORBA.

The interface between the wrapper and the client program is also an important part of the framework. A concise and standard interface is needed here, so we adopt Interface Definition Language (IDL) of the CORBA standard to define it. The CORBA standard is language independent, so that the wrapper generated within the framework can be communicated to the client program whatever the language or operating system is. That is, a wrapper generated within the framework is an XML-based and CORBA-enabled component over the network. In this way, wrappers fully support the integrated heterogeneous information retrieval system we propose in Ref. [18].

3.2. Workflow of wrapper generation

As discussed earlier, most information sources constantly change content or even structure. Wrapper codes are tightly coupled with the structure of a specified information source. If the information source changes the document structure, the wrapper implementer must also change the wrapper codes. Such constant modification is both time-consuming and tedious. Consequently, an information retrieval application developer welcomes an automatic wrapper generation system that decreases the workload. Unlike what is proposed in other works, the XML-based wrapper generation framework we present is automatic and consequently answers the need. Not only is it XML-compliant and CORBA-enabled, but the generating procedure is simple and fast. The workflow for the framework is shown in Fig. 3.

There are many editing tools for XML and IDL files. The wrapper implementer must select the appro-

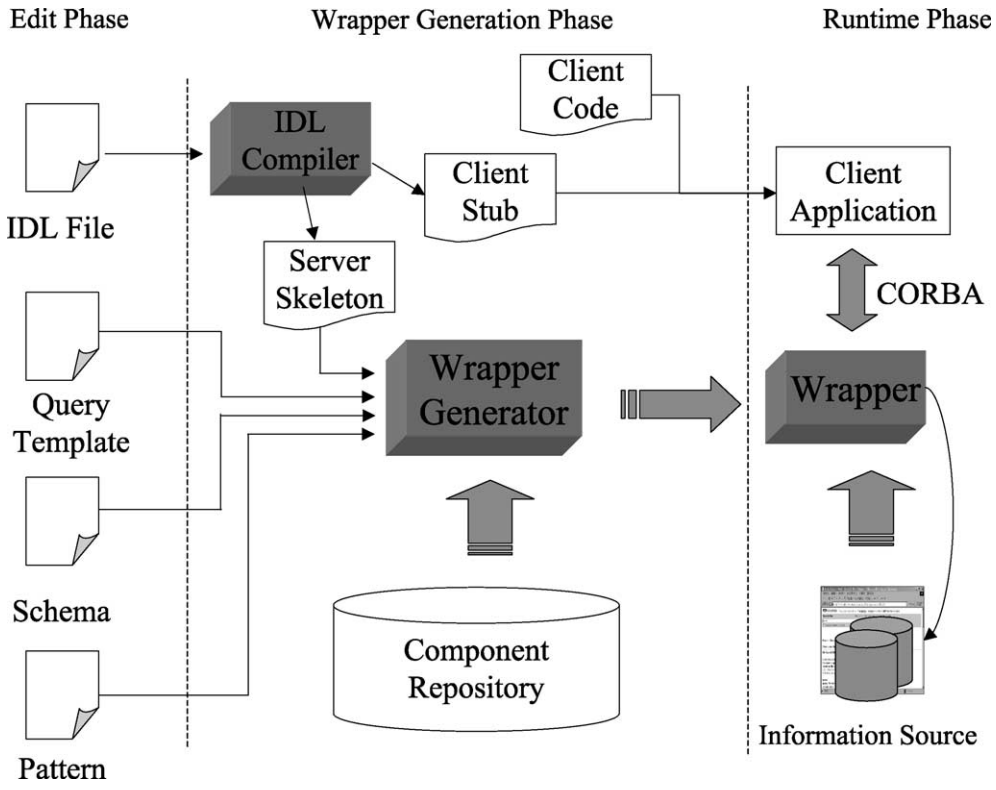


Fig. 3. The workflow of the proposed wrapper generation.

priate tools to define and prepare the QUERYTEMPLATE, PATTERN, SCHEMA, and IDL files shown in Fig. 3. Consider the *Wrapper Generation* phase shown in this figure. First, an IDL compiler is employed to compile a user-defined IDL file and produce a client stub and server skeleton for the interface between the client application and the CORBA-based generated wrapper. The information retriever uses the client stub to develop a client program that communicates to the wrapper in an appropriate way. The server skeleton together with the QUERYTEMPLATE, PATTERN, and SCHEMA files are required in the wrapper generation procedure. Employing these user-defined files, the generator selects the appropriate components stored in the *Component Repository* to produce the desired wrapper. At *Runtime Phase*, the new wrapper can be invoked by to retrieve the request in the client program via the CORBA interface defined in the IDL file formation.

3.3. Architecture of wrapper generation

The architecture of our wrapper generator is shown in Fig. 4. Three XML-formed files, QUERYTEMPLATE, SCHEMA, and PATTERN, are required. An XML parser parses them and stores the desired information into DOM objects defined by W3C. That is, following *XML Parsing Phase*, three DOM objects are produced. Then, *Query Rules Analysis Phase* creates the Query Transfer Rules and the Query String according to the DOM objects coming from the QUERYTEMPLATE file. The Data Extract Rules are generated at *Data Extract Rule Analysis Phase* according to the DOM objects generated from the SCHEMA file. The Pattern Sample is produced by *Pattern Analysis Phase* according to the DOM object generated from PATTERN file. The Code Generator shown in Fig. 4 then chooses suitable components from the Component Repository in accordance with all these information

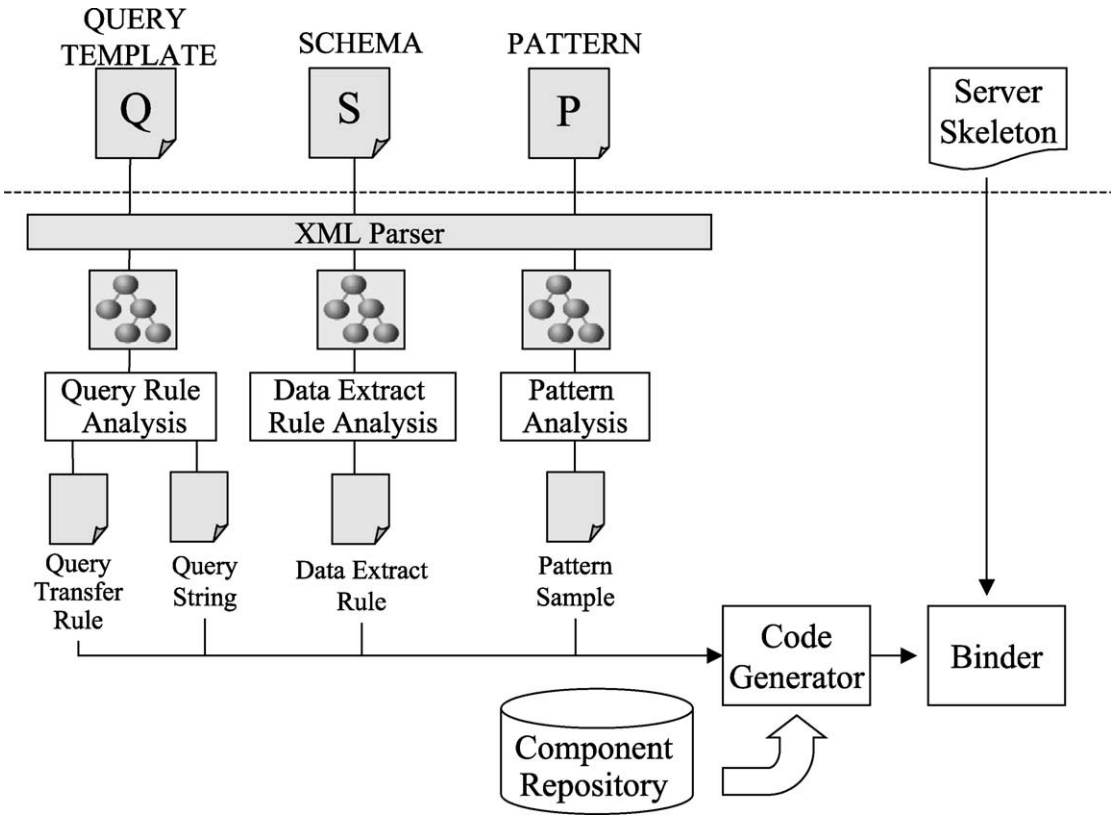


Fig. 4. Architecture of automatic wrapper generator.

(Query Transfer Rules, Query String, Data Extract Rules, and Pattern Sample). Finally, Binder binds Code Generator-chosen components and Server Skeleton code fragments produced by the IDL compiler into the desired new wrapper.

4. System implementation

4.1. Implementation basics

The proposed wrapper generator is implemented entirely in Java Language. We use JDK 1.3 to develop all the codes, and any new wrapper is consequently Java coded. The XML parser is a Xerces Java Parser 1.3.0, an Open Source Software published by APACHE XML Project [19]. The components in the Component Repository are also Java coded. We use XML tags to construct the Component Repository, which allows the system to use the tags to extract appropriate components.

4.2. DTD files for the system

The wrapper generator collects information about the specifications of a new wrapper by analyzing the three XML files prepared by the wrapper implementer. To facilitate usage of the XML tags, certain DTD files for the XML files are provided within the framework. They are (1) QUERYTEMPLATE.DTD, (2) SCHEMA.DTD, and (3) PATTERN.DTD. The QYERYTEMPLATE.DTD file is shown in Fig. 5.

The purpose of QUERYTEMPLATE.XML is to define the format of a querying string or command for a specified information source. Consequently, QUERYTEMPLATE.DTD is to define all the requirements about the definition of a querying string or commands for any information source. In addition, SCHE-

```
<?xml version="1.0" encoding="BIG5"?>
<!ELEMENT QUERYTEMPLATE (HEAD,DECLARE,QueryString, SendString)>
<!ELEMENT HEAD (PROJECTNAME,AUTHOR)>
<!ELEMENT PROJECTNAME (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT DECLARE (PARAMETER+)>
<!ELEMENT PARAMETER (PARA_name, PARA_description, PARA_temp?)>
<!ELEMENT PARA_name (#PCDATA)>
<!ELEMENT PARA_description (#PCDATA)>
<!ELEMENT PARA_temp (SRC,DEST)+>
<!ELEMENT SRC (#PCDATA|WORD)*>
<!ELEMENT DEST (#PCDATA|WORD)*>
<!ELEMENT WORD EMPTY>
<!ATTLIST WORD name CDATA #REQUIRED >
<!ELEMENT QueryString (#PCDATA|PARA_list|CONDITION)*>
<!ELEMENT PARA_list EMPTY>
<!ELEMENT CONDITION EMPTY>
<!ATTLIST CONDITION parameter CDATA #REQUIRED >
<!ELEMENT SendString (#PCDATA|INSERT_PARA)*>
<!ELEMENT INSERT_PARA EMPTY>
<!ATTLIST INSERT_PARA name CDATA #REQUIRED >
<!ATTLIST INSERT_PARA times CDATA "1">
```

Fig. 5. The source code of "QUERYTEMPLATE.DTD".

MA.XML and PATTERN.XML guide a new wrapper in parsing and extracting user-interested information from retrieved documents. This framework also must provide the new wrapper with DTD files as a guideline for parsing the XML files. In fact, these two files are always combined into one file called PATTERN.DTD. The content of PATTERN.DTD is shown in Fig. 6.

According to the specifications defined in XML files, the system generates codes for a new wrapper by integrating components chosen from the Component Repository. As before, the generated codes are all Java codes that can be compiled by Java Compiler (JDK 1.3) into Java class.

4.3. Component Repository

In our design, the wrapper is composed of several independent components. The Java codes of the components are written in advance and stored in the Component Repository. For simplicity of management, each component has its own meta-data XML file, in which the component name, author, descriptions, and tag are defined. In this way, while at the generation stage, the essential components in a wrapper are combined according to the attributes. That is, the components are written in Java code and managed by the XML meta-data files. With practice, the wrapper developer

```
<?xml version="1.0" encoding="BIG5" ?>
<!ELEMENT PATTERNS (HEAD,SCHEMA,RECORDSAMPLES)>
<!ELEMENT HEAD (PROJECTNAME,AUTHOR)>
<!ELEMENT PROJECTNAME (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT SCHEMA (VARIABLE+)>
<!ELEMENT VARIABLE (VAR_name, VAR_type, VAR_valid, VAR_descript)>
<!ELEMENT VAR_name (#PCDATA)>
<!ELEMENT VAR_type (#PCDATA)>
<!ELEMENT VAR_valid (#PCDATA)>
<!ELEMENT VAR_descript (#PCDATA)>
<!ELEMENT RECORDSAMPLES (#PCDATA)>
```

Fig. 6. The source code of "PATTERN.DTD".

can not only create components independently, but can also reuse them with ease.

5. Example demonstration

Fig. 7 features a real example of a generated wrapper for retrieving information from an AltaVista search engine. For such a search engine, the basic fields of interest to a user are the Title, Description, and URL in the returned pages for some keywords searched. A wrapper generated in this framework accepts the SQL-like query string via the CORBA standard from the mediator of an Integrated Information Retrieval system, and then feeds an appropriate query string to the search engine. On obtaining the results, the wrapper extracts the desired information according to the PATTERN.XML and SCHEMA.XML files from the wrapper implementer.

The first step in generating a new wrapper for AltaVista is to analyze the syntax and semantics of the query string, and then write the QUERYTEMPLATE. XML file in XML format according to QUERYTEMPLATE.DTD syntax. That is, the new wrapper must translate the higher-level query string in the upper left corner of Fig. 7 into a form that matches that of the string in the upper right corner of the figure. In our example wrapper, the only two words that may possibly be changed in the searching case are keyword and date. These are shown in bold face. We analyze the format of the AltaVista query strings and commands and write the QUERYTEMPLATE.XML file, the content of which is shown in Fig. 8.

For the second step, the information extraction stage, the wrapper generator verifies the correctness of PATTERN.XML according to PATTERN.DTD, after which the generated wrapper extracts the desired information according to the content defined in PATTERN.XML. Consequently, the implementer has to define the schema and the pattern matching string in PATTERN.XML. This is important for guiding the wrapper to extract the desired information from the

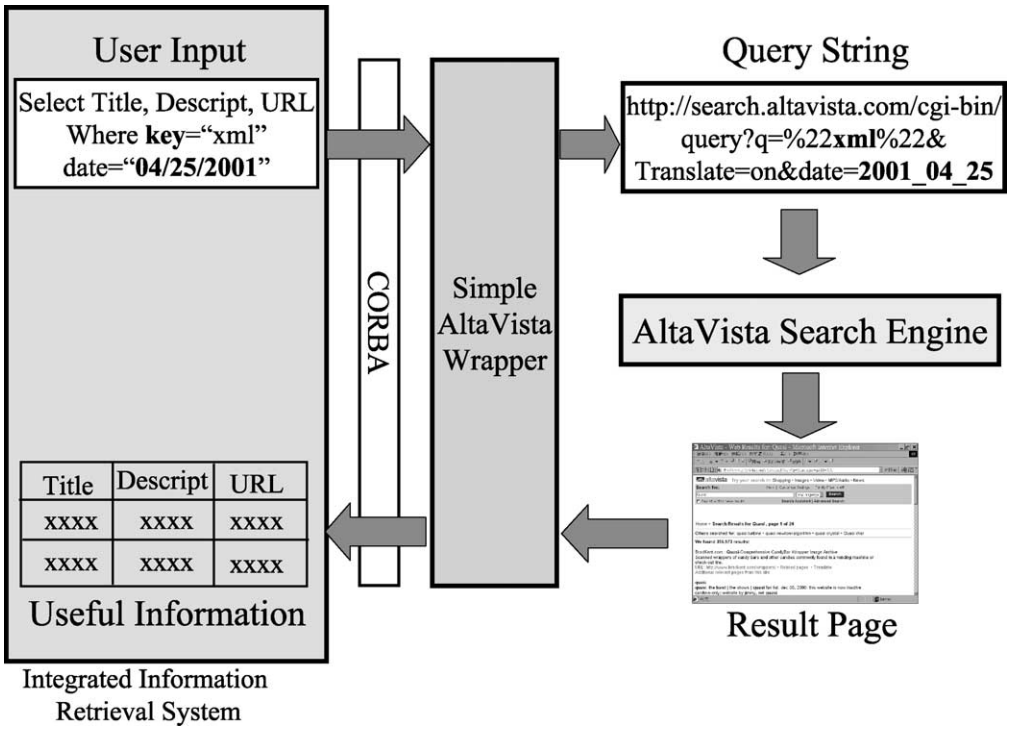


Fig. 7. The role of wrapper for AltaVista.

```
<?xml version="1.0" encoding="BIG5"?>
<!DOCTYPE QueryTemplate SYSTEM "QUERYTEMPLATE.DTD">
<QueryTemplate>
   <HEAD>
        <PROJECTNAME>ALTAVISTA</PROJECTNAME>
        <AUTHOR>James Sun</AUTHOR>
   </HEAD>
   <DECLARE>
        <PARAMETER>
        <PARA_name>keyword</PARA_name>
        <PARA_description>keyword for searching</PARA_description>
        <PARA_temp>
            <SRC>key=<WORD name="k" /></SRC>
            <DEST>q=%22<WORD name="k"/>%22</DEST>
        </PARA_temp>
        </PARAMETER>

        <PARAMETER>
        <PARA_name>date</PARA_name>
        <PARA_description>generate date of web page</PARA_description>
        <PARA_temp>
            <SRC> date=<WORD name="m"/> / <WORD name="d"/> /
                  <WORD name="y"/></SRC>
            <DEST> <WORD name="y"/>_<WORD name="m"/> _
                  <WORD name="d"/></DEST>
            <SRC> date=<WORD name="d"/>days </SRC>
            <DEST> after <WORD name="d"/> </DEST>
        </PARA_temp>
        </PARAMETER>
   </DECLARE>
   <QueryString>
        Select < Var_list/> Where <CONDITION parameter=
             "      keyword"/>, <CONDITION parameter="date" />;
   </QueryString>

   <SendString>
        http://search.altavista.com/cgi-bin/query?
             <INSERT_PARA name="keyword" times="*"/>
             &amp;amp;kl=XX&amp;amp;pg=q&amp;amp;Translate=on
             <INSERT_PARA name="date" times="1"/>
   </SendString>
</QueryTemplate>
```

Fig. 8. "QUERYTEMPLATE.XML" for AltaVista wrapper.

returned pages correctly. The architecture of PATTERN.XML, as we defined it for this example, is depicted in Fig. 9.

This architecture has three basic parts: (1) HEAD, (2) SCHEMA, and (3) RECORDSAMPLE. First, HEAD denotes the attributes of the file, which is an important information for document management. Second, SCHEMA describes all the fields to be processed. The VARIABLE item of SCHEMA defines the variable's name, data type, description and the like, and is the data schema for the structure of the specified information source. Third, RECORDSAMPLES is a matching string that is applied to the specified information source and has only one element, ⟨![CDATA [ ]]⟩. A pattern matching procedure is applied to the content with ⟨![CDATA[ ]]⟩ as the matching string. It checks to see if there are matched fields to be extracted. In the pattern, all the interested fields have to be embraced by the matching string defined by ⟨![CDATA[ ]]⟩ (in this example, ^&^. . .. . .^&/^). That is, all the characters between string ^&^ and ^&/^ are treated as the fieldname. Then the system compares the

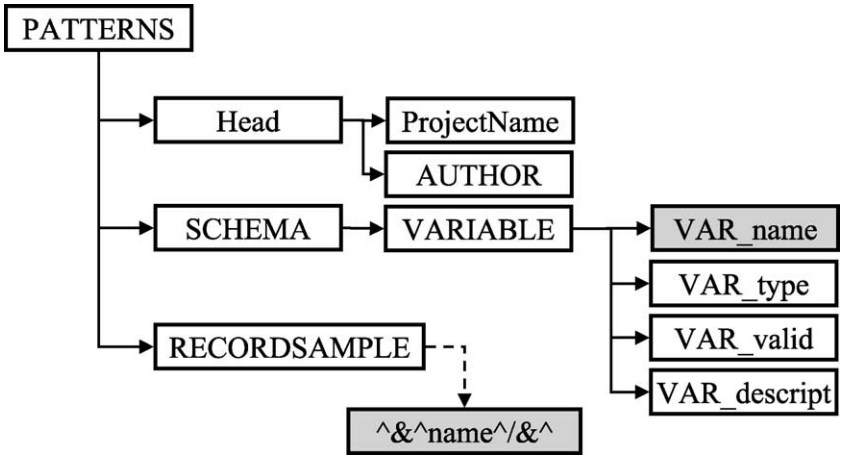


Fig. 9. Architecture of "PATTERN.XML" file in the example.

pattern file with the returned document to find the value of the field, and assigns the appropriate fieldname to it. The detail of the codes for PATTERN.XML is shown in full in Fig. 10.

We now consider an example of a real returned web page from AltaVista and observe what the real steps in wrapper implementation are. Sample fragments of the returned page are shown in Fig. 11.

The arranged and simplified HTML codes corresponding to Fig. 11 are shown in Fig. 12. Note that for ease of describing the steps for preparing PATTERN.XML, certain HTML codes are omitted from the figure.

The example web page shown in Fig. 12 has three fields of interest: (1) Title, (2) Description, and (3) URL. The first task of the wrapper implementer is to fetch the HTML file shown in Fig. 11. The matching string defined in the <RECORDSAMPLES> part of PATTERN.XML (in the example, ^&^……^&/^) must be inserted in the appropriate position and the appropriate fieldnames for the fields of interest chosen to replace the target string. In this case, the fragments of the finished pattern file are shown in Fig. 13. In contrast with Fig. 12, the fields of interest (Title, Description, URL) are properly stored in the mapped variables in the records. In addition, we use ^&^misc1^/&^ to mark fragments of no concern. To simplify matters for future procedure, we discard the value of misc1. Note also that the file shown in Figs. 12 and 13 is the simplified result. The real pattern string is shown in the down half of Fig. 10.

The workflow shown in Fig. 14 is of further help in explaining the pattern matching procedure of the new wrapper. The procedure is simple and direct. The system has only to discard any unwanted tokens (the white blocks in the figure) until the matching string is found. Once the first ^&^ is found, it extracts the characters between ^&^ and ^&/^, and assigns them to the appropriate field variable according to the context defined in PATTERN.XML. With this simple pattern matching method, the wrapper implementer has no need to analyze and understand the HTML codes and structure of documents retrieved from the information sources. The wrapper implementer needs only to retrieve the HTML web page from the specified information source (in this case, AltaVista), to find out the interested fields of the file, and then to mark and to define the fieldname of the interested field by embracing the matching string defined in PATTERN.XML. This is a simple, quick, and direct method of generating a new wrapper for a new or modified information source.

The example described above demonstrates the simplicity and ease with which the wrapper generator in this framework constructs an XML and CORBA-enabled wrapper. The wrapper implementer only has to provide the wrapper generator with the QUERYTEMPLATE, SCHEMA, and PATTERN files. These are then combined into two files (QUERYTEMPLATE.XML and PATTERN.XML), which are written in XML syntax validated by the respective DTD file. It is a simple matter for a programmer to analyze

```
<?xml version="1.0" encoding="BIG5"?>
<!DOCTYPE PATTERNS SYSTEM "PATTERN.DTD">
<PATTERNS>
   <HEAD>
        <PROJECTNAME>ALTAVISTA</PROJECTNAME>
        <AUTHOR>James Sun</AUTHOR>
   </HEAD>
   <SCHEMA>
        <VARIABLE>
            <VAR_name>Title</VAR_name>
            <VAR_type>text</VAR_type>
            <VAR_valid>yes</VAR_valid>
            <VAR_descript>Page title of result link</VAR_descript>
        </VARIABLE>
        <VARIABLE>
            <VAR_name>Descript</VAR_name>
            <VAR_type>text</VAR_type>
            <VAR_valid>yes</VAR_valid>
            <VAR_descript>Description of result page</VAR_descript>
        </VARIABLE>
        <VARIABLE>
            <VAR_name>URL</VAR_name>
            <VAR_type>preserve</VAR_type>
            <VAR_valid>yes</VAR_valid>
            <VAR_descript>URL of result link</VAR_descript>
        </VARIABLE>
        <VARIABLE>
            <VAR_name>misc1</VAR_name>
            <VAR_type>preserve</VAR_type>
            <VAR_valid>no</VAR_valid>
            <VAR_descript> Misc field</VAR_descript>
        </VARIABLE>
        <VARIABLE>
            <VAR_name>misc2</VAR_name>
            <VAR_type>preserve</VAR_type>
            <VAR_valid>no</VAR_valid>
            <VAR_descript>Misc field</VAR_descript>
        </VARIABLE>
        <VARIABLE>
            <VAR_name>misc3</VAR_name>
            <VAR_type>preserve</VAR_type>
            <VAR_valid>no</VAR_valid>
            <VAR_descript>Misc field</VAR_descript>
        </VARIABLE>
   </SCHEMA>
   <RECORDSAMPLES> <![CDATA[
<DL>
 <DT><B class=txt2>^&^misc1^/&^</B> <B><A
 ^&^misc2^/&^>^&^Title^/&^</A></B>
 <DD>^&^Descript^/&^<BR><SPAN class=ft>URL:
 ^&^URL^/&^ </SPAN><BR>^&^misc3^/&^
</DD></DL>
      ]]>
   </RECORDSAMPLES>
</PATTERNS>
```

Fig. 10. The detail source code of ”PATTERN.XML” for AltaVista wrapper.

1. Quasi Comprehensive Candy Bar *Wrapper* Image Archive
Thorough collection of candy wrappers images organized by country.
URL: www.bradkent.com/wrappers/
Translate More pages from this site Related pages

Fig. 11. The fragments of the returned document in AltaVista.

```
<DL>
  <DT><B class=txt2>1.</B>
  … omitted by the authors …
  <B> Quasi comprehensive Candy Bar Wrapper image Archive</B></A>
  <DD> Thorough collection of candy wrappers images organized by country <BR>
  <SPAN class=ft> URL:www.bradkent.com/wrappers/ </SPAN>
  <BR><A href=http://jump.altavista.com/go?url=http%3A%2F&amp;>Translate</A>
  ……..
  omitted by the authors
  ……..
  </DD>
</DL>
```

Fig. 12. Corresponding HTML Code to Fig. 11.

the syntax of a query string for an information source and then encode it in XML format. Preparing PATTERN. XML is equally trouble-free, simply a question of download-and-string-replacement work. An additional advantage is the availability of user-friendly interface tools to assist in the preparation of these files.

6. Comparisons, discussions and future work

This section begins with comparing the merits of our design with the previous works, and then discusses the challenges of generating wrappers for retrieving information from heterogeneous sources. Also, we mention topics in mind for future research.

6.1. Comparing automatic generation with hand-written wrapper

In both quantity and variety, the growth rate of forms of information on the Internet is enormous. Consequently, the only efficient option must be a retrieval application that collects information automatically. To handle such variety, an application must prepare many wrappers. The traditional application developer spends much time in writing codes for wrapper generation and in connecting wrappers to applications. Automatic wrapper-generating techniques, however, dispense with most of that work. A comparison of traditional and automatic wrapper-generating techniques is shown in Fig. 15.

We compare the performance of an experienced programmer with that of our generator in generating specific wrappers for two the widely used search engines, Yahoo and AltaVista. Tables 1 and 2 show the comparison for Yahoo and AltaVista respectively. For our design, we use Microsoft Windows 2000 on an Intel Pentium II machine run at 233 MHz.

6.2. Comparing our approach with others

As mentioned above, several previous approaches, each with its own characteristics, have focused on automatic or semi-automatic wrapper generation. We summarize those of XWRAP [4], W4F, [11], TSIM-

```
<DL>
  <DT><B class=txt2>1.</B>
  <B> ^&^Title^/&^</B>
  <DD> ^&^Descript^/&^<BR>
  <SPAN class=ft> ^&^URL^/&^ </SPAN>
  <BR>^&^misc1^/&^
  </DD>
</DL>
```

Fig. 13. The fragments of the completed pattern file.

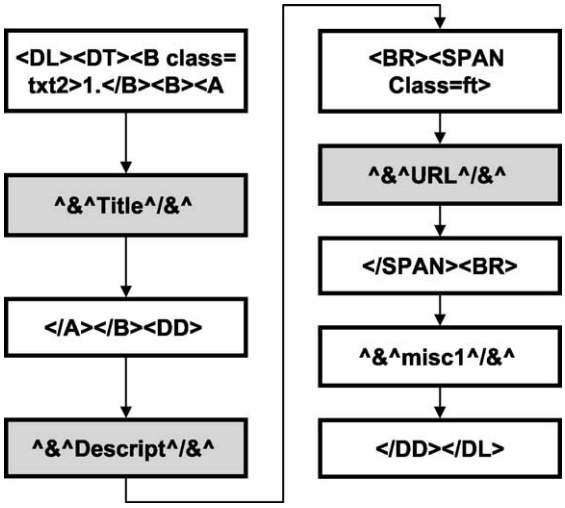


Fig. 14. The flow of pattern matching procedure.

MIS [7] and those of our own approach in Table 3. As can be seen there, we use XML to define the meta-data and provide the unified interface support for heterogeneous information sources. This makes wrapper generation for us a simple and fast undertaking, while the wrappers themselves can connect to a uniform interface information retrieval application that is CORBA-compliant. This constitutes the major merit of our design.

6.3. Versatile query string formats of search engines

The first challenge to generating a wrapper is the versatility of the query string format. There are many ways to design an interactive user interface for search engines or information sources with which to respond to the user requests. Each method has its own specific composition of query arguments. In addition, certain client-program-solutions (applet, ActiveX) do not use URL of WWW for server/client communication. Although most encapsulation rules can be discovered by closely analyzing HTML codes or commands, much attention must be paid to creating the encapsulation of search request for information retriever by a formal and automatic way. In our approach, we simply let the wrapper implementer discover and insert the rules into a QUERYTEMPLATE.XML file. This is a simple and fast method, whether for most information sources (especially for search engines) or for users searching for only a few, regular documents. For sources with complicated query string format, the wrapper implementer needs an assistant tool to help analyze the encapsulating rules for the query string. For sources

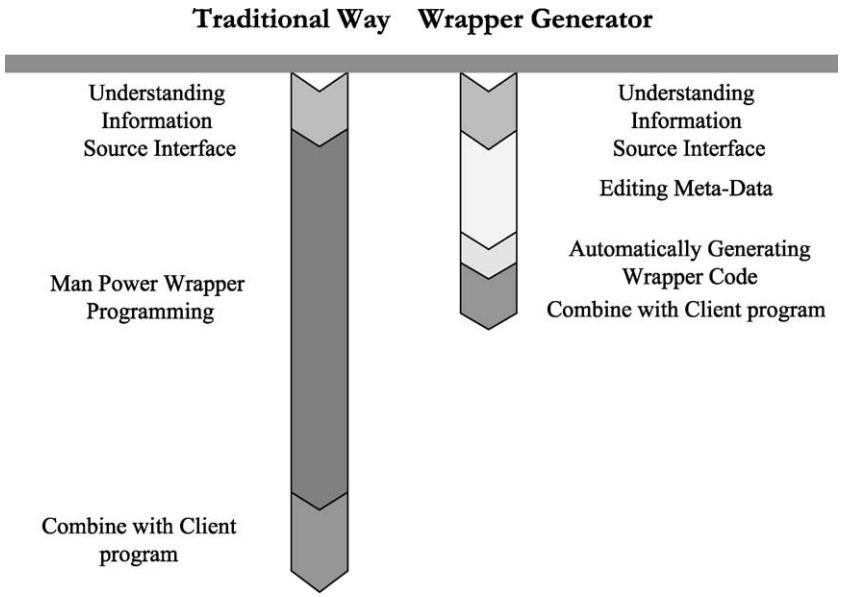


Fig. 15. The comparison between the traditional way and our approach.

Table 1
Comparison between manual writing and our approach for Yahoo

	Manual writing	Our approach
Meta-data file	No need	105 lines
Wrapper codes	153 lines	234 lines
Coding/generating time	About 2 days	3.8 s
Execute time	4.2 s	5.7 s

that constantly change their query string format, the need is for a tool that automatically adjusts the encapsulating rules for the query string. We have both such tools in view for future research.

6.4. Versatile protocols of information sources

The Internet has many information sources that do not provide a WWW interface for users, but which instead follow certain protocols (e.g., Z39.50) for server/client communication. Consequently, the only way for an implementer to generate a wrapper for these protocols is to know all their specifications. However, with these kinds of information sources, knowing the protocols is difficult if they do not follow any standard. However, an advantage of these sources is that if the protocol is a well-known standard, it is fixed. Fortunately, most well-known protocol standards have their own API that can be invoked by the wrapper.

6.5. Integrated wrapper development environment

All the file formats in our proposed framework are written in XML syntax. The implementer has no need to learn a new language to generate a wrapper. Writing XML files by hand is tedious. Although there are many word editors for editing XML files, it is also important to have an integrated development environment for wrapper generation. A wrapper implementer must have the following components: (1) an automatic query string analyzer for writing appropriate QUERYTEMPLATE.XML; (2) an XML file editor; (3) a web page retriever for sample information source files and to prepare the PATTERN.XML files needed for wrapper generation; (4) an automatic generator for producing the appropriate wrappers. The integrated development environment is also a possible topic of future research by the authors.

Table 2
Comparison between manual writing and our approach for AltaVista

	Manual writing	Our approach
Meta-data file	No need	116 lines
Wrapper codes	184 lines	278 lines
Coding/generating time	About 2 days	4.8 s
Execute time	5.1 s	6.4 s

Table 3
Comparisons among some famous approaches

	XWRAP	W4F	TSIMMIS	Our approach
Developer assistant	Yes	Yes	Yes	No
Unified interface supporting	No	No	No	Yes
Meta-data	Private structure	HEL	MSL	XML
Internal data structure	–	NSL	OEM	DOM
Component Repository	None	None	Library	Yes
Training data	Yes	No	1 sample	No
Providing GUI	Yes	Yes	None	None

7. Conclusions

In this paper, we have presented a framework for the automatic generation of wrappers that supports a unified interface meta-search system based on an XML data model and CORBA standard. With this framework, a wrapper implementer's only task is to design the wrapper specifications for a specified information source in an XML-syntax file, which a wrapper generator then uses to construct a new wrapper.

The usual method for extracting information fields of interest in the framework is by pattern matching. Using it, a wrapper implementer quickly and easily prepares wrapper specifications for a specified information source. This differs from previous approaches in that a wrapper generated in our framework focuses on extracting fields of interest from returned documents and not on analyzing their content. An implementer is not required to understand the whole structure of a specified information source. Consequently, time and cost in generating new wrapper are greatly reduced.

All the data structures, including imported files and the representation of objects are XML-compliant. Since XML is a widely popular standard nowadays, most developers are familiar with it and many software applications process data with it perfectly well. Combining the advantages of both XML and Java, the Component Repository used in the framework is flexible and extensible. The codes of the Component Repository are easy to be extended and managed. Finally, wrappers generated in this framework are all CORBA-enabled. That is, for information retrieval application, they are already language independent, in which merit adds to their communicative capabilities and spares the wrapper implementer much effort.

Acknowledgements

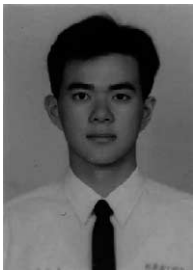
We are grateful for the many excellent comments and suggestions made by the anonymous referees. This work was supported in part by the National Science Council of the Republic of China under Grant No. NSC90-2213-E-159-005 and the Ministry of Education's Program of Excellence Research under Grant 89-E-FA04-1-4.

References

[1] Y.S. Chang, S.M. Yuan, W. Lo, A new multi-search engine for querying data through internet search service on CORBA, Int. J. Comput. Networks 34 (3) (2000) 467–480.
[2] Y.S. Chang, S.M. Yuan, A Framework for Integrating Information Retrieval on CORBA, in Proc. of 12th International Workshop on Database and Expert Systems Applications (DEXA 2001), Munich, Germany, Sept. 3–7 (2001) 180–185.
[3] M.E. Vidal, L. Raschid, J.R. Gruser, A meta-wrapper for scaling up to multiple autonomous distributed information sources, Proceedings of the Third International Conference of Cooperative Information Systems, New York City, USA (1998) 148–157.
[4] L. Liu, C. Pu, W. Han, XWRAP: an XML-enabled wrapper construction system for web information sources, 16th International Conference on Data Engineering (ICDE'2000), San Diego, CA, USA, IEEE (2000) 611–621.
[5] B. Chidlovskii, J. Ragetli, M. de Rijke, Automatic wrapper generation for Web Search Engines, Lecture Notes in Computer Science (LNCS) of Springer-Verlag, Shanghai, China (WAIM 2000) 1846 (6) (2000) 399–410.
[6] N. Ashish, C.A. Knoblock, Semi-automatic wrapper generation for Internet information sources, Proceedings of the Second IFCIS International Conference, Kiawah Island, South Carolina, USA (COOPIS'97) (1997) 160–169.
[7] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom, The TSIMMIS approach to mediation: data models and languages, J. Intell. Inf. Syst. 8 (2) (1997) 117–132.
[8] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, Integrating and accessing heterogeneous information sources in TSIMMIS, Proc. of the AAAI Symp. On Information Gathering, Stanford, California (1995) 61–64.
[9] G. Huck, P. Frankhouser, K. Aberer, E. Neuhold, Jedi: extracting and synthesizing information form the Web, Proceedings. 3rd IFCIS International Conference on Cooperative Information Systems, New York City, USA (1998) 32–41.
[10] A. Sahuguet, F. Azavant, Building Intelligent Web Applications Using Lightweight Wrappers, Data and Knowledge Engineering 36 (3) (2001) 283–316.
[11] M. Chau, D. Zeng, H. Chen, Personalized Spiders for Web Search and Analysis, JCDL'01, USA, Virginia (2001) 79–87.
[12] L. Introna, H. Nissenbaum, Defining the web: the politics of search engines, IEEE Comput. 33 (1) (2000) 54–62.
[13] D. Mattox, L. Seligman, K. Smith, Rapper: A Wrapper Generator with Linguistic Knowledge, WIDM 99 ACM, Kansas City, USA (1999) 6–11.
[14] L. Liu, W. Han, D. Buttler, C. Pu, W. Tang, An XML-Based Wrapper for Web Information Extraction, SIGMOD'99 ACM, Philadelphia, PA (1999) 540–543.
[15] Y.S. Chang, M.H. Ho, S.M. Yuan, A unified interface for integrating information retrieval, Comput. Stand. Interfaces 23 (4) (2001) 325–340.
[16] Apache XML Project, http://xml.apache.org/index.html.
[17] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. Ullman, A query translation scheme for rapid implementation of wrappers, International Conference on Deductive and Object-Oriented Databases, Singapore (1995) 161–186.

Yue-Shan Chang was born on August 4, 1965 in Tainan, Taiwan, Republic of China. He received the BS degree in Electronic Technology from National Taiwan Institute of Technology in 1990, the MS degree in Electrical Engineering from the National Cheng Kung University in 1992, and the PhD degree in Computer and Information Science from National Chiao Tung University in 2001. Dr. Chang became an Associate Professor at the Department of Electronics Engineering of Minghsin Institute of Technology in August 2001. His current research interests include Distributed Systems, Object Oriented Programming, Information Retrieval and Integration, and Internet Technologies.

Min-Huang Ho was born on February 1, 1969 in Kaohsiung, Taiwan, Republic of China. He received the BS and MS degrees in Industrial Education from National Taiwan Normal University in 1993 and 1995, respectively. Currently, he is a candidate of PhD in Computer and Information Science at National Chiao Tung University. His research interests are in Distributed Systems, Internet Technologies, and Mobile Agent Technologies.



Wen-Chen Sun was born on August 12, 1976 in Taipei, Taiwan, Republic of China. He graduated from Yuan-Ze University in 1999 and received MS degree in Computer and Information Science from National Chiao Tung University in 2001. He is now working as vice engineer at Institute for Information Industry. His research interests are in Distributed System and Mobile Computing Technologies.

Shyan-Ming Yuan was born on July 11, 1959 in Mauli, Taiwan, Republic of China. He received the BSEE degree from National Taiwan University in 1981, the MS degree in Computer Science from University of Maryland Baltimore County in 1985, and the PhD degree in Computer Science from University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he had been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include Distributed Objects, Internet Technologies, and Software System Integration. Dr. Yuan is a member of ACM and IEEE.