XWRAPComposer: A Multi-Page Data Extraction Service for Bio-Computing Applications

Ling Liu, Jianjun Zhang, Wei Han, Calton Pu, James Caverlee, Sungkeun Park
College of Computing, Georgia Institute of Technology

Terence Critchlow, Matthew Coleman, David Buttler
Lawrence Livermore Nationral Laboratory, California, USA

Abstract

This paper presents a service-oriented framework for the development of wrapper code generators, including the methodology of designing an effective wrapper program construction facility and a concrete implementation, called XWRAPComposer Three unique features distinguish XWRAPComposer from existing wrapper development approaches. First, XWRAPComposer is designed to enable multi-stage and multi-page data extraction. Second, XWRAPComposer is the only wrapper generation system that promotes the distinction of information extraction logic from query-answer control logic, allowing higher level of robustness against changes in the service provider's web site design or infrastructure. Third, XWRAPComposer provides a user-friendly plug-and-play interface, allowing seamless incorporation of external services and continuous changing service interfaces and data format. [1]

1 Introduction

With the wide deployment of Web service technology, the Internet and the World Wide Web (Web) have become one of the most popular means for disseminating scientific data from a variety of disciplines. Vast and growing amount of life sciences data reside today in specialized Bioinformatics data sources, many of them are accessible online with specialized query processing capabilities. The Molecular Biology Database Collection, for instance, currently holds over 500 data sources [1], not even including many tools that analyze the information contained therein. Bioinformatics data sources over the Internet have a wide range of query processing capabilities. Typically, many Web-based sources allow only limited types of selection queries. To compound the problem, data from one source often must be combined with data from other sources to provide scientists with the information they need.

The extraordinary growth of service oriented computing has been fueled by the enhanced ability to make a growing amount of information available through the Web. This brings good news and bad news. The good news is that the Web services provide the standard invocation interface for remote service calls and the bulk of useful and valuable information is designed and published in a human browsing format (HTML or XML). The bad news is that these "human-oriented" Web pages returned by Web services are difficult for programs to capture and extract information of interests automatically and to fuse and integrate data from multiple autonomous and yet heterogeneous data producer services. Also different web service providers use different and evolving custom data formats.

A popular approach to handle this problem is to write data wrappers to encapsulate the access to Web sources and to automate the information extraction tasks on behalf of human. A wrapper is a software program specialized to a single data source or single Web service (e.g., a web site), which converts the source documents and queries from the source data model to another, usually a more structured, data model [8]. Several projects have implemented hand-coded wrappers for a variety of sources [13]. However, manually writing such a wrapper and making it robust is costly due to the irregularity, heterogeneity, and frequent updates of the Web site and the data presentation formats they use. Hand-coding wrappers can become a major pain in situations where the data integration applications are more interested in integrating new data sources or frequently changing Web sources. We observe that, with a good design methodology, only a relatively small part of the wrapper code deals with the source-specific details, and the rest of the code is either common among wrappers or can be expressed at a higher level, more structured fashion. There are a number of challenging issues in automation of the wrapper code generation process.

- First, most Web pages are HTML or XML documents, which are semi-structured text files annotated with various HTML presentation tags. Due to the frequent changes in presentation style of the HTML documents, the lack

[1]This work is a joint project between Georgia Institute of Technology and University of California Lawrence Livermore National Laboratory. For the first six authors, this work is partially supported by the Department of Energy under a SciDAC SDM grant, the National Science Foundation under a CNS Grant, an ITR grant, a Research Infrastructure grant, and the industry sponsors under an IBM SUR grant, an IBM faculty award, and an HP equipment grant. For the last three authors, this work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48. UCRL-CONF-209837.

of semantic description of their information content, and the difficulty in making all applications in one domain to use the same XML schema, it is hard to identify the content of interest using common pattern recognition technology such as string regular expression specification used in LEX and YACC.

- Second, wrappers for Web sources should be more robust and adaptive in the presence of changes in both presentation style and information content of the Web pages. It is expected that the wrappers generated by the wrapper generation systems will have lower maintenance overhead than handcrafted wrappers for unexpected changes.
- Third, wrappers often serve as interface programs and pass the Web data extracted to application-specific information broker agents or information integration mediators for more sophisticated data analysis and data manipulation. Thus it is desirable to provide a wrapper interface language that is simple, self-describing, and yet powerful enough for extracting and capturing information from most of the Web pages.

In scientific computing domains such as bioinformatics and bioengineering, information extraction over multiple different pages imposes additional challenges for wrapper code generation systems due to the varying correlation of the pages involved. The correlation can be either horizontal when grouping data from homogeneous documents (such as multiple result pages from a single search) or vertical when joining data from heterogeneous but related documents (a series of pages containing information about a specific topic). Furthermore, the correlation can be extended into a graph of workflows as we will describe in Figure 2. Therefore there is an increasing demand for automated wrapper code generation systems to incorporate a multi-page information extraction service. A multi-page wrapper not only enriches the capability of wrappers to extract information of interests but also increases the sophistication of wrapper code generation.

Surprisingly, almost all existing wrappers generated by application code generators [13, 8] are single-page wrappers in the sense that the wrapper program responds to a keyword query by analyzing only the page immediately returned. Most wrappers cannot follow the links within this page to continue the information extraction from other linked pages, unless separate queries are issued to locate other linked pages.

Bearing all these issues in mind, we develop a code generation framework for building a semi-automated wrapper code generation system that can generate wrappers capable of extracting information from multiple inter-linked Web documents, and we implement this framework with XWRAPComposer, a toolkit for semi-automatically generating Java wrapper programs that can collect and extract data from multiple inter-linked pages automatically. XWRAPComposer has three unique features with regard to supporting multi-page data extraction.

- First, we introduce interface, outface, and composer script for each wrapper program we generate. By encoding wrapper developers' knowledge in Interface Specification, Outerface Specification and Composer Script, XWRAPComposer integrates single-page wrapper programs into a composite wrapper capable of extracting information across multiple inter-linked pages from one service provider.
- Second, XWRAPComposer transforms the multi-page information extraction problem into an integration problem of multiple single-page data extraction results, and utilizes the composer script to interconnect a sequence of single-page data extraction results, offering flexible execution choices to address diverse needs of different users. It generates platform-independent Java code that can be executed locally on users' machine. It also provide a WSDL-plugin module to allow users to produce WSDL enabled wrappers as Web Services [12].
- Third but not the least, XWRAPComposer supports micro-workflow management, such as intermediate information flow or result auditing. We demonstrate this capability by integrating XWRAPComposer and its generated wrappers with some process modeling tools such as Ptolemy [2], allowing users to interactively manage different components of a wrapper and the interaction between them.

2 The Design Framework

A multi-page wrapper code generation is a complex process and it is not reasonable, either from a logical point of view or from an implementation point of view, to consider the construction process as occurring in one single step. For this reason, we partition the wrapper construction process into a series of subprocesses called *phases*, as shown in Figure 1. A phase is a logically cohesive operation that takes as input one representation of the source document and produces as output another representation. XWrapComposer wrapper generation goes through six phases to construct and release a Java wrapper. Tasks within a phase run concurrently using a synchronized queue; each runs its own thread. For example, we decide to run the task of fetching a remote document and the task of repairing the bad formatting of the fetched document using two concurrently synchronous threads in a single pass of the source document. The task of generating a syntactic-token parse tree from an HTML document requires as input the entire document; thus, it cannot be done in the same pass as the remote document fetching and the syntax reparation. Similar analysis applies to the other tasks such as code generation, testing, and packaging.

The interaction and information exchange between any two of the phases is performed through communication with the bookkeeping and the error handling routines. The *bookkeeping* routine of the wrapper generator collects information about all the data objects that appear in the retrieved source document, keeps track of the names used by the program, and records essential information about each. For example, a wrapper needs to know how many arguments a tag expects, whether an element represents a string or an integer. The data

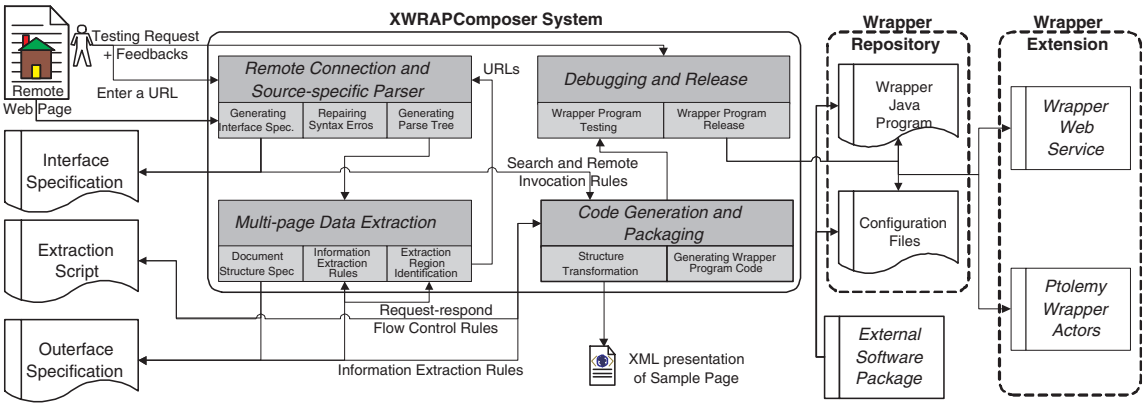

Figure 1. XWRAPComposer System Architecture

structure used to record this information is called a symbol table. The *error handler* is designed for the detection and reporting errors in the fetched source document. The error messages should allow a wrapper developer to determine exactly where the errors have occurred. Errors can be encountered at virtually all the phases of a wrapper. Whenever a phase of the wrapper discovers an error, it must report the error to the error handler, which issues an appropriate diagnostic message. Once the error has been noted, the wrapper must modify the input to the phase detecting the error, so that the latter can continue processing its input, looking for subsequent errors. Good error handling is difficult because certain errors can mask subsequent errors. Other errors, if not properly handled, can spawn an avalanche of spurious errors. Techniques for error recovery are beyond the scope of this paper.

Figure 1 presents an architecture sketch of the XWRAPComposer system. The system architecture of XWRAPComposer consists of four major components: (1) Remote Connection and Source-specific Parser; (2) Multi-page Data Extraction; (3) Code Generation and Packaging; and (4) Debugging and Release. Other components include GUI interface, bookkeeping and error handling. GUI interface allows wrapper developers to specify workflow of the multi-page data extraction, the request-respond flow control rules and cross-page data extraction rules interactively.

Remote Connection and Source-specific Parser is the first component, which prepares and sets up the environment for information extraction process by performing the following three tasks. First, it accepts an URL selected and entered by the XWrapComposer user, issues an HTTP request to the remote service provider identified by the given URL, and fetches the corresponding web document (or so called page object). During this process, the XWRAPComposer will learn the search interface and the remote service invocation procedure in the background and generate a set of rules that describe the list of interface functions and parameters as well as how they are used to fetch a remote document from a given web source. The list of interface functions include the declaration to the standard library routines for establishing the network connection, issuing an HTTP request to the remote web server through a `HTTP Get` or `HTTP Post` method, and fetching the corresponding web page. Other desirable functions include building the correct URL to access the given service and pass the correct parameters, and handling redirection, failures, or authorization if necessary. Second, it cleans up bad HTML tags and syntactical errors using XWRAPComposer plugin such as HTML TIDY [10, 11]. Third, it transforms the retrieved page object into a parse tree or so-called syntactic token tree. This page object will be used as a sample for XWRAPComposer to interact with the user to learn and derive the important information extraction rules, and the list of linked pages the user is interested in extracting information in conjunction with this page. In addition, all wrappers generated by XWrap use the streaming mode instead of the blocking mode (recall Section 2). Namely, the wrapper will read the web page one block[2] at a time. An interface specification will be created in this phase.

Multi-page Data Extraction is the second component, which is responsible for deriving information flow control logic and multi-page extraction logic, both are represented in form of rules. The former describes the flow control logic of the targeted service in responding to a service request and the latter describes how to extract information content of interest from the answer page and the linked pages of interest. XWRAPComposer performs the multi-page information extraction task in four steps: (1) specify the structure of the retrieved document (page object) in a declarative extraction rule language. (2) identify the interesting regions of the main page object and generating information extraction rules for this page; (3) identify the list of URLs referenced in the extracted regions in the main page; and (4) generating information extraction rules for each of the pages linked from the interesting regions of the main page object. We perform single page data extraction process using XWRAPElite [5] toolkit, a single page data extraction service developed by the XWRAP team at Georgia Tech. At the end of this phase, XWRAP-

[2] A block here refers to a line of 256 characters or a transfer unit defined implicitly by the HTTP protocol.

Composer produces two specifications: an outerface specification that describes the output format of the extraction result will be produced, and a composer script that describes both the information flow control patterns and the multi-page data extraction patterns.

Code Generation and Packaging is the third component, which generates the wrapper program code by applying three sets of rules about the target service produced in the first two steps: (1) the search and remote invocation rules, (2) the request-respond flow control rules, and the information extraction rules. A key technique in our implementation is the smart encoding of these three types of semantic knowledge in the form of active XML-template format (see Section 4 for detail). The code generator interprets the XML-template rules by linking each executable component with the corresponding rule sets. The code generator also produces the XML representation for the retrieved sample page object as a byproduct.

Debugging and Release is the fourth component and the final phase of the multi-page wrapping process. It allows the user to enter a set of alternative service requests to the same service provider to debug the wrapper program generated by running the XWRAPComposer's code debugging module. For each page object obtained, the debugging module will automatically go through the syntactic structure normalization to rule out syntactic errors, the flow control and information extraction steps to check if new or updated flow control rules or data extraction rules should be included. In addition, the debug-monitoring window will pop up to allow the user to browse the debug report. Whenever an update to any of the three sets of rules occurs, the debugging module will run the code generator to create a new version of the wrapper program. Once the user is satisfied with the test results, he or she may invoke the release to obtain the release version of the wrapper program, including assigning the version release number, packaging the wrapper program with application plug-ins and user manual into a compressed tar file.

XWRAPComposer wrapper generator takes the following three inputs: interface specification, outerface specification, and composer script, and compiles them into a Java wrapper program, which can be further extended into either a multi-page data extraction Web service (with WSDL specification) or a Ptolemy wrapper actor, which can be used for large scale data integration.

In the subsequent sections, we first provide a walkthrough example to illustrate the multi-page extraction process. Next, we focus our discussion primarily on multi-page data extraction component of the XWrapComposer, and give a brief description of the wrapping interface and remote invocation component as the necessary preprocessing step for information extraction, together with a short summary of code generation as the postprocessing for the multipage extraction.

3 Example WalkThrough

Before describing the detailed techniques used in designing multi-page data extraction services, we first present a walkthrough of XWRAPComposer using the motivating example introduced in Figure 2, where a biologist first uses a program called *Clusfavor* to cluster genes that have changed significantly in a micro-array analysis experiment. After extracting all gene ids from the Clusfavor result, he feeds them into the NCBI Blast service, which searches all related sequences over a variety of data sources. The returned sequences will be further examined to find promoter sequences. Let us focus on the NCBI BLAST service. Figure 2 shows the workflow of how a BLAST service request to NCBI will be served. It consists of four steps: BLAST response step presents the user with a request ID, BLAST delay step presents the user with the time delay for the result. BLAST Summary presents the user with an overview of all gene ids that match well with the given gene sequence id. Finally, BLAST Detail shows for each gene id listed in the summary page, the full sequence detail and the goal is to extract approximately 1000-5000 bases of the DNA sequence around the alignment to capture the promoter regulatory elements, the region of a gene where RNA polymerase can bind and begin transcription to create the proteins that can regulate cell function.

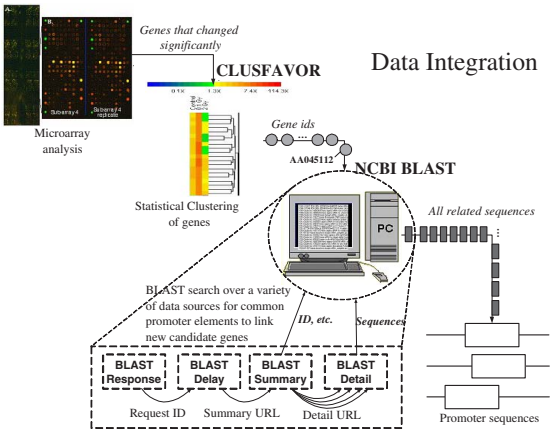

Figure 2. A Scientific Data Integration Example Scenario

Figure 3 illustrates a typical BLAST query using NCBI service [9]. A BLAST query involves four steps. The first step is to feed a gene sequence into the text entry of the query interface. Due to the time complexity of a BLAST search, the NCBI service provider typically returns a response page with a request ID and the first estimate of the waiting time for each BLAST search. The biologist may later ask NCBI for the BLAST results using the request ID (Step 2), the NCBI service will presents a delay page if the BLAST search is not completed and results are not yet ready to display (Step 3). Once the BLAST results are delivered, they are displayed in a BLAST summary page, which contains a summary of all genes matching the search query condition. Each of the matching genes will provide a link to the NCBI BLAST Detail page (Step 4). If the gene id used for the BLAST query is incorrect gene id or NCBI does not provide BLAST service

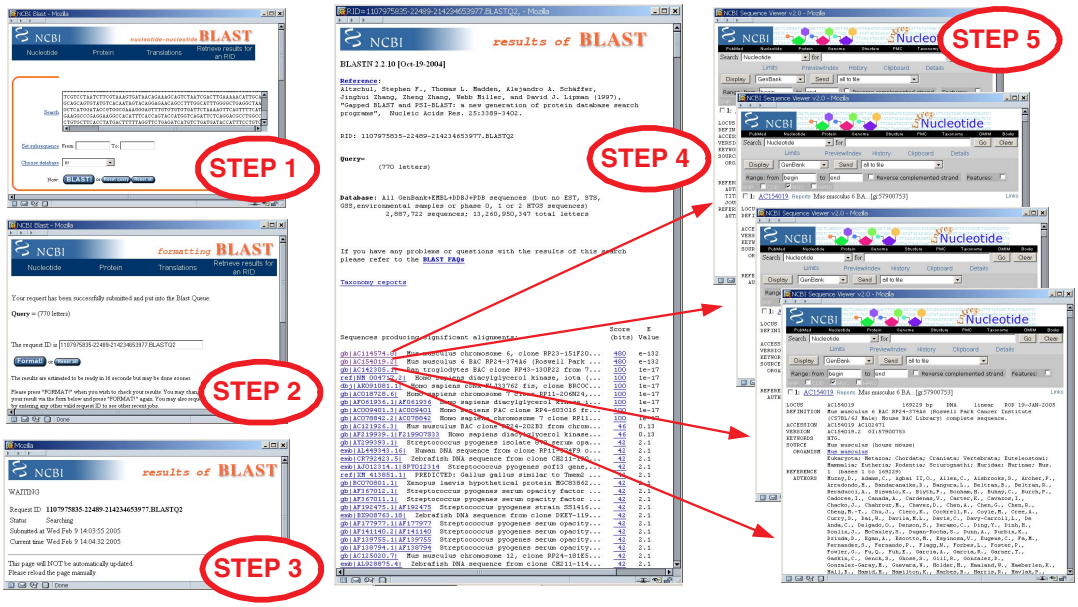

Figure 3. Multipage query with a NCBI web site

for the given gene id, an error page will be displayed. If the summary page does not include detailed information that the biologist is interested in, he has to visit each detail page (Step 5) through the URLs embedded in the summary page.

A critical challenge for providing system-level support for scientists to achieve such complex data integration tasks is the problem of locating, accessing, and fusing information from a rapidly growing, heterogeneous, and distributed collection of data sources available on the Web. This is a complex search problem for two reasons. First, as the example in Figure 2 shows, scientists today have much more complex data collection requirements than ordinary surfers on the Web. They often want to collect a set of data from a sequence of searches over a large selection of heterogeneous data sources, and the data selected from one search step often forms the filter condition for the next search step, turning a keyword-based query into a sophisticated search and information extraction workflow. Second, such complex workflows are manually performed daily by scientists or data collection lab researchers (computer science specialists). Automating such complex search and data collection workflows presents three major challenges.

- Different service providers use different request-respond flow control logics to present the answer pages to search queries.
- Cross-page data extraction has more complex extraction logic than the single page extraction system. In addition, different applications require different sets of data to be extracted by the cross-page data extraction engine. Typically, only portions of one page and the links that lead the extraction to the next page need to be extracted.
- Data items extracted from multiple inter-linked pages require being associated with semantically meaningful naming convention. Thus, mechanisms that can incorporate the knowledge of the domain scientists who issued such cross-page extraction job are critical.

There are several ways to design NCBI BLAST wrapper. First, we can develop two wrappers, one for NSBI BLAST summary and one for NCBI BLAST Detail. The NCBI BLAST summer wrapper can be integrated with the NCBI BLAST Detail wrapper by service composition. In this approach, we need to capture the request-respond flow control through a flow control logic in the composer script of NCBI Summary wrapper. The outerface specification of the NCBI summary wrapper consists of the general overview of the given gene id and the list of gene ids that are relevant to the given gene id. The NCBI BLAST Detail wrapper needs to extract approximately 1000-5000 bases of the DNA sequence around the alignment. The composite wrapper NCBI BLAST will be composed of the NCBI summary wrapper and a list of executions of the NCBI BLAST Detail wrapper. In the next section we describe the XWRAPComposer design using this example.

4 Multi-Page Data Extraction Service

We have developed a methodology and a framework for extraction of information from multiple pages connected via web page links. The main idea is to separate what to extract from how to extract, and distinguish information extraction logic from request-respond flow control logic. The control logic describes the different ways in which a service request (query) could be answered from a given service provider. The data extraction logic describes the cross-page extraction steps, including what information is important to extract at each page and how such information is used as a complex

filter in the next search and extraction step.

We use interface description to specify the necessary input objects for wrapping the target service and the outerface description to describe what should be extracted and presented as the final result by the wrapper program. We design and develop a XWRAPComposer Script language (a set of functional constructs) to describe the request-respond flow control logic and multi-page data extraction logic, and to implement the output alignment and tagging of data items extracted based on the outerface specification.

The compilation process of the XWRAPComposer includes generating code based on three sets of rules: (1) Remote connection and interface rules, (2) the request-respond flow control logic and multi-page extraction logic outlined in the composer script, (3) the correct output alignment and semantically meaningful tagging based on the outerface specification.

4.1 Interface and Outerface Specification

Interface specification describes the schema of the data that the wrapper takes as input. It defines the source location and the service request (query) interface for the wrapper to be generated. Outerface specification describes the schema of the result that the wrapper outputs. It defines the type and structure of objects extracted. The composer script consists of two sets of rule-based scripts. The request-respond flow control script describes the alternative ways that the target service will respond to a remote service request, including result not found, multiple results found or single result found, or server errors. The multi-page data extraction script which describes (1) the extraction rules for the main page, (2) the extraction rules for each of the interesting pages linked from the main page, and (3) the rules on how to glue single page data extraction components. XWRAPComposer's scripting language has domain-specific plugins to facilitate the incorporation of domain-dependent correlations between the fragments of information extracted and the domain-specific tagging scheme. Each wrapper generated by XWRAPComposer will be associated with an interface specification, an outerface description, and a composer script.

The design of the XWRAPComposer Interface and Outerface Specification serves two important objectives. First and for most, it will ease the use of XWRAP wrappers as external services to any data integration applications. Second, it will facilitate the XWRAPComposer wrapper code generation system to generate Java code. Therefore, some components of the specification may not be directly useful for the users of these wrappers. In the first release of the XWRAPComposer implementation, we describe the input and output schema of a multi-page (composite) wrapper in XML Schema and use the two XML schemas as the interface and outerface specification. Concretely, the interface specification describes the wrapper name and which data provider's service needs to be wrapped by giving the source URL and other related information. The outerface specification describes what data items should be extracted and produced by the wrapper and the semantically meaningful names to be used to tag those data items. Figure 4 shows a fragment of the interface and outerface description of an example NCBI BLAST summary wrapper.

```
<XCwrapper name="XC_BlastN_Summary" sourceURL=
  "http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?PAGE=Nucleotides">
  <interface><!-- input schema in XML Schema -->
    <xsd:element name="input" type="xsd:string">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="select_db" type="string"/>
          <xsd:element name="query_sequence" type="string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </interface>
  <outerface><!-- output schema in XML Schema -->
    <xsd:element name="resultDoc">
      <xsd:complexType>
        <xsd:element name="output">
          <xsd:complexType>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="homolog">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="geneid" type="string"/>
                    <xsd:element name="description" type="string"/>
                    <xsd:element name="length" type="int"/>
                    <xsd:element name="score" type="string"/>
                    <xsd:element name="expect" type="string"/>
                    <xsd:element name="identities" type="string"/>
                    <xsd:element name="strand" type="string"/>
                    <xsd:element name="link" type="string"/>
                    <xsd:element name="beginMatch" type="int"/>
                    <xsd:element name="endMatch" type="int"/>
                    <xsd:element name="alignment" type="string"/>
                    <xsd:element name="beginMatch" type="string"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:attribute name="docLocation" type="string"/>
        <xsd:attribute name="docType" type="string"/>
        <xsd:attribute name="createdBy" type="string"/>
        <xsd:attribute name="creationDate" type="string"/>
      </xsd:complexType>
    </xsd:element>
  </outerface>
</XCwrapper>
```

Figure 4. An Example Of Interface And Outerface Specification – NCBi Summary

4.2 Multi-page Data Extraction Script

XWRAPComposer multi-page data extraction service will generate a composer script for each wrapper it creates. Each composer script usually contains three types of root commands, document retrieval, data extraction and post processing. The document retrieval commands construct a file request or an HTTP request and fetch the document. The data extraction commands specify the detailed instructions on how to extract information from the fetched document. The post processing commands allow adding semantic filters to make the extracted results conform to the outerface specification.

Table 1 shows a list of commands that are currently supported in the first release of the XWRAPComposer toolkit [6].

Figure 5 gives an extraction script example for the NCBI Summary wrapper. Given a full sequence as the input, we first construct an NCBI Blast search URL based on the NCBI Blast interface description. The script fragment *Set variable* { *[text()]* indicates the sequence is in the input with the XPath, "*text()*". The first script command *FetchDocument* retrieves the NCBI Blast response page that contains a request ID. We extract the ID and construct the URL of the search results from the main page object. The control-flow command

Command	Category
ConstructHttpQuery	Document Retrieval
ReadFile	Document Retrieval
FetchDocument	Document Retrieval
ExtractLink	Data Extraction
ExtractContent	Data Extraction
GrabSubstring	Grab Function
GrabXWrapEliteData	Grab Function
GrabConsecutiveLines	Grab Function
GrabCommaDelimitedText	Grab Function
ContainSubstring	Boolean Comparison
While...Do...	Control Flow
If...Then...	Control Flow
ApplyStyleSheet	Post Processing
Sleep	Process Management

Table 1. Supported XWRAPComposer Extraction Root Commands

while...do... periodically invokes the second *FetchDocument* to retrieve the result page until the results are delivered. Finally we use *GrabXWRAPEliteData* to extract useful data from the main result page. We use the command *ExtractLink* to locate each of the linked pages of interest from the main page object and use the command *ExtractContent* to invoke the XWRAPElite single page data extraction service to extract useful data from each linked page. Due to the space restriction, we omit the concrete techniques used in XWRAPComposer for single page data extraction and refer readers to [3, 13] for further detail.

4.3 Execution Model of an XWRAPComposer Wrapper

A typical XWRAPComposer wrapper consists of the following five basic functional modules.

- The **Search Interface** module accepts the user input through the protocols defined by the user, such as the SOAP request in web service scenario. It constructs the service request (query command) and parameter list that will be forwarded to the wrapped target service. Consider the NCBI BLAST wrapper, its search interface accepts the gene sequence and the other parameters such as alignment precision from the input specification file or GUI interface. It composes the HTTP POST command, which will be used to execute the query.
- The **Remote Invocation** module accepts the service request (query command) and parameters generated by the search interface and converted them into the query acceptable by the wrapped target service. The query can be a HTTP POST command, a FTP GET command, or an RPC call. The remote invocation module interacts with the wrapped target service following the remote connection protocol defined by the wrapped target service and the communication procedure defined by the configuration file. The query result page will be forwarded to the parser for preprocessing before entering the multi-page data extraction module.
- The **Page Parser** translates the result page received from the remote invocation module into a token tree structure, filters out the uninteresting information such as advertisements from web pages, and converts the received document into a standard format such as HTML or XML. In addition to building a token based parse tree, the page parser should incorporate the domain-specific knowledge about the page encoded in the composer script to facilitate the data extraction process. For multi-page wrappers, the page parser will parse the main respond page based on its extraction rules and locate the list of linked pages of interest. For each of the linked pages of interest, the parser triggers the remote invocation module to fetch the actual page and parses the page based on its corresponding extraction rules.
- **Information Extraction** module processes each of the parsed documents passed from the parser and extracts the objects of interest defined by the outerface specification. It uses the domain specific knowledge about the pages of interest, encoded in the composer extraction script, to guide the concrete multi-page data extraction process. For each extracted data object, the XML tagging procedure is applied to assign tag name to the object based on the tagging rules encoded in the composer script.

```
/* Start constructing wrapper ncbisummary.  */
WrapperName "ncbisummary";

/* Contruct the URL for NCBi Blast search */
Generate blastSummaryPage ::  ConstructHttpQuery (input){
   Set inputSource {
      Set url {"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?QUERY=$$&..."};
      Set queryString { };
      Set method {"get"};
      Set variable { [text()] } ;
   }
}
Generate blastSummaryData ::  FetchDocument (blastSummaryPage) {}
Generate recordid ::  ExtractContent (blastSummaryData) {
   GrabSubstring {
      Set BeginMatch {"The request ID is <input name=\"RID\" size=\"50\"
type=\"text\" value=\"};
      Set EndMatch {"\" >" };
   }
}
Generate answerurl ::  ConstructHttpQuery (recordid){
   Set inputSource {
      Set url {"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?FORMAT_PAGE
_TARGET=Format_page_31680&RESULTS_PAGE_TARGET=Blast_Results_for_31680
&RID=$$&SHOW_OVERVIEW=on...&AUTO_FORMAT=Semiauto"};
      Set queryString { };
      Set method { "get" };
      /* The first recordid is the input id.*/
      Set variable { [text()] };
   }
}
Generate answerPage ::  FetchDocument(answerurl) {}
While {
   ContainSubstring(answerPage) {
      Set compSubstring {"This page will be automatically updated in"};
   }
} Do {
   Generate answerPage ::  FetchDocument(answerurl) {}
   /* Pause for 10 seconds.  */
   Sleep {
      set inverval {"10000"};
   }
}
Generate output ::  ExtractContent (answerPage) {
   GrabXWRAPEliteData {
   /* The following properties should be generated from a XWRAPELite tool.  */
   ...
   }
}
```

Figure 5. Extraction Script Example For NCBi Summary

- **Output Packaging and Delivery** module merges the output from the information extraction module and packages it into the final result format defined by the outerface specification. Then it delivers the data package to the user who initiates the execution of the wrapper program.

The first prototype of XWRAPComposer system is written in Java. Wrappers generated by XWRAPComposer are also coded in Java. In our first prototype implementation, the five components execute sequentially − a component starts execution only after the previous component finishes. The next extension of XWRAPComposer code generation system is to introduce parallel extraction among these five components. Parallel execution improves the performance, but it also incurs higher complexity in implementation.

4.4 WSDL-enabled Wrappers

XWRAPComposer is developed with two objectives in mind. First, we want to generate wrapper programs that can be used in command line or embedded in an application system as a wrapper procedure. Second, we want XWRAPComposer to be able to generate WSDL-enabled wrappers to allow each wrapper program to be used as a Web service. Our discussion so far has been focused on the first objective. In this section we briefly describe how to generate WSDL enabled wrappers.

In order to enable XWRAPComposer to generate WSDL-enabled wrapper services, we add two extensions to the XWRAPComposer wrapper generation system. First, we encapsulate an XWRAPComposer wrapper into a general Web service servlet. The servlet automatically extracts the input from a SOAP request, feeds it into the wrapper, and inserts the wrapping results in a SOAP envelope before sending back to the user. Second, we incorporate a WSDL generator to automatically generate Web service description by binding the wrapper's interface and outerface with the servlet configuration.

5 Related Work and Conclusion

The very nature of scientific research and discovery leads to the continuous creation of information that is new in content or representation or both. Despite the efforts to fit molecular biology information into standard formats and repositories such as the PDB (Protein Data Bank) and NCBI, the number of databases and their content have been growing, pushing the envelope of standardization efforts such as mmCIF [14]. Providing integrated and uniform access to these databases has been a serious research challenge. Several efforts have sought to alleviate the interoperability issue [4, 7], by translating queries from a uniform query language into the native query capabilities supported by the individual data sources. Typically, these previous efforts address the interoperability problem from a digital library point of view, i.e., they treat individual databases as well-known sources of existing information. While they provide a valuable service, due to the growing rate of scientific discovery, an increasing amount of new information (the kind of hot-off-the-bench information that scientists would be most interested in) falls outside the capability of these previous interoperability systems or services.

Wrappers have been developed either manually or with software assistance, and used as a component of agent-based systems, sophisticated query tools and general mediator-based information integration systems. XWRAPComposer is different from those systems in three aspects. First, we explicitly separate tasks of building wrappers that are specific to a Web service from the tasks that are repetitive for any service, thus the code can be generated as wrapper library component and reused automatically by the wrapper generator system. Second, we use inductive learning algorithms that derive information flow and data extraction patterns by reasoning about sample pages or sample specifications. More importantly, we design a declarative rule-based script language for multi-page information extraction, encouraging a clean separation of the information extraction semantics from the information flow control and execution logic of wrapper programs.

References

[1] DBCAT, the public catalog of databases. see http://www.infobiogen.fr/services/dbcat.

[2] Berkeley. Ptolemy group in eecs. *http://ptolemy.eecs.berkeley.edu/*, 2003.

[3] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. *Proceedings of IEEE ICDCS*, April 2001.

[4] T. Critchlow, K. Fidelis, M. Ganesh, R. Musick, and T. Slezak. Datafoundry: Information management for scientific data. *IEEE Transactions on Information Technology in Biomedicine, 4(1):52-57*, March 2000.

[5] DISL Group, Georgia Insitute of Technology. Xwrap elite project. *http://www.cc.gatech.edu/projects/disl/XWRAPElite*, 2000.

[6] Georgia Insitute of Technology. Xwrapcomposer. *http://www.cc.gatech.edu/projects/disl/XWRAPComposer*, 2003.

[7] L. Haas, P. Schwarz, P. Kodali, E. Kotlar, J. Rice, and W. Swope. Discoverylink: A system for integrated access to life sciences data sources. *IBM Systems Journal, 40(2):489-511*, 2001.

[8] L. Liu, C. Pu, and W. Han. XWrap: An Extensible Wrapper Construction System for Internet Information Sources. In *Technical Report, OGI/CSE, Feb.*, 1999.

[9] NCBI. National center for biotechnology information – blast databases. *http://www.ncbi.nlm.nih.gov/BLAST/*, 2003.

[10] D. Raggett. Clean up your web pages with HTML TIDY. *http://www.w3.org/People/Raggett/tidy/*, 1999.

[11] W3C. Reformulating HTML in XML. *http://www.w3.org/TR/WD-html-in-xml/*, 1999.

[12] W3C. Web services description language (wsdl) version 1.2 part 1: Core language. *http://www.w3c.org/TR/wsdl12/*, 2003.

[13] H. Wei. *Wrapper Application Generation for Semantic Web: An XWRAP Approach*. PhD thesis, Georgia Institute of Technology, 2003.

[14] J. Westbrook and P. Bourne. Star/mmcif: An extensive ontology for macromolecular structure and beyond. *Bioinformatics, 16(2):159-168*, 2000.