XWRAPComposer: A Multi-Page Data Extraction Service

Ling Liu, Jianjun Zhang, Wei Han, Calton Pu, James Caverlee,
Sungkeun Park

College of Computing, Georgia Institute of Technology

{lingliu, zhangjj, weihan, calton, caverlee, mungooni}@cc.gatech.edu

Terence Critchlow, David Buttler, Matthew Coleman

Lawrence Livermore Nationral Laboratory, California, USA

{critchlow1, buttler1, coleman16}@llnl.gov

Abstract

We present a service-oriented architecture and a set of techniques for developing wrapper code generators, including the methodology of designing an effective wrapper program construction facility and a concrete implementation, called XWRAPComposer. Our wrapper generation framework has two unique design goals. 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 a 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. We implement these design principles with the development of the XWRAPComposer toolkit, which can semi-automatically generate WSDL-enabled wrapper programs. We illustrate the problems and challenges of multi-page data extraction in the context of bioinformatics applications and evaluate the design and development of XWRAPComposer through our experiences of integrating various BLAST services.

KEY WORDS:

Web Services, Service Oriented Architecture, Data Extraction, Code Generator

Introduction

With the wide deployment of Web service technology, the Internet and the World Wide Web (Web) have become the most popular means for disseminating both business and scientific data from a variety of disciplines. For example, vast and growing amount of life sciences data reside in specialized Bioinformatics data sources, and many of them are accessible online with specialized query processing capabilities. Concretely, the Molecular Biology Database Collection currently holds over 500 data sources (DBCAT, 1999), 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.

Motivating Scenario   In the Bioinformatics and Bioengineering domain, many biologists currently use a variety of tools, such as DNA microarrays, to discover how DNA and the proteins they encode may allow an organism to respond to various stress conditions such as exposure to environmental mutagens (Quandt, Frech, Karas, Wingender, & Werner, 1995; Altschul et al., 1997; DBCAT, 1999). One way to accomplish this task is for genomics researchers to identify genes that react in the desired way, and then develop models to capture the common elements. This model will be used to identify previously unidentified genes that may also respond in similar fashion based on the common elements. Figure 1 illustrates a workflow that a genomics researcher has created to gather the data required for this analysis. This type of workflow significantly differs from traditional workflows, as it is iteratively generated to discover the correct process with a small set of data as the initial input. At each step the researcher selects and extracts the part of the output data that is useful for his genomic analysis in the next step, and determines which services should be used in the next step in his data collection process. Once the workflow is constructed, the genomic researcher will use the workflow as the data collection pattern to collect large quantities of data and perform large scale genomic analysis. Concretely, Figure 1 shows a pattern of a promoter model where the data collection is performed in eight steps using possibly eight or more Bioinformatics data sources through service oriented computing interfaces.

In Step (1), microarrays containing the genes of interest are produced and exposed to different levels of a specific mutagen in the wet-lab, usually in a time dependent manner.

In Step (2) gene expression changes are measured and clustered using some computational tools (e.g., *Clusfavor* (Peterson, 2002)), such that genes that changed significantly in a micro-array analysis experiment are identified and clustered. The representative genes from Clusfavor analysis will be used as the input for the next data collection step. Typically the researcher must choose from a wide variety of tools available for this task either manually based on his past experience or using a Web service selection facility. Each tool offers specific advantages in terms of their ability to analyze the microarray data, and each requires a different method of execution.

In Step (3), the full sequence from each of the representative genes chosen in the second step is retrieved from gene-banks.
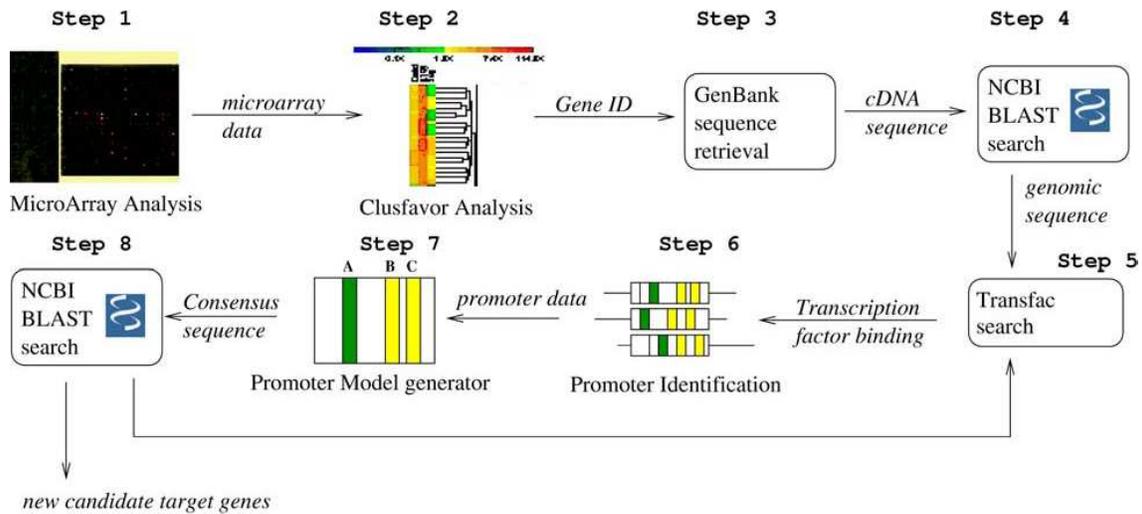
Figure 1.   An Example Workflow for Developing a Promoter Model

In Step (4), each gene sequence retrieved in Step (3) will be submitted to a gene matching service, such as NCBI Blast Web service, that will return homologs (other genes with similar sequences). The returned sequences will be further examined to find promoter sequences. Again, there are several services that provide gene similarity matching, many of which specialize in a particular species, such as ACEdb (Stein & Thierry-Mieg, 1999).

Once related sequences are discovered, approximately 1000-5000 bases of the DNA sequence around the alignment are extracted to capture the promoter regulatory elements – the region of a gene where RNA polymerase can bind and begin transcription to create the proteins that regulate cell function. In Step (5), these promoter sequences are identified and analyzed using specific tools, such as Mat-Inspector (Peterson, 2002), TRANSFAC, TRRD, or COMPEL (Quandt et al., 1995) to find the common transcription binding factors. To extract specific data, such as portions of a DNA sequence, returned by the sources, the data needs to be converted into a well-known format, such as XML, and post-processed in order to extract just the portions that are relevant for the next step.

In Step (6), regulatory profiles are then compared across each gene in the cluster to delineate common response elements that can be fed into the promoter model generator to create a promoter model in Step (7). Once the model is created, it can be used to search gene databases to find other candidate genes relevant to the study in Step (8), which starts a new iteration where these genes are fed back into this general workflow to refine and expand the promoter model until the genomic researcher is satisfied with the result. The collection of genes found in this iterative process will be presented as the final results of this complex data analysis task.

It is important to point out that each of these steps requires service selection, automated data extraction, service composition and integration. Choosing the appropriate source depends on the content, capabilities and load of the source, as well as the trustworthiness of the source. Some sites have much stricter standards on the quality of the

data that they admit, while others publish information as soon as it is available. Depending on the current needs of a particular researcher, different types of sites may be more appropriate to query. In addition to selecting a capable and trustworthy source, there are significant issues in extracting data from the sites. Most sites have custom query interfaces and return results through a series of HTML pages. For example, NCBI BLAST (Basic Local Alignment Search Tool) (Altschul et al., 1997) requires the user to take three or four steps in order to retrieve the matching sequence homologs. First, a gene sequence must be submitted through an HTML form. Users may then optionally select the format in which the data returned should be represented. Then, a series of delay pages are shown while the service calculates the final answer. Once the answer is computed, a page listing the related sequence ids and their alignment information are presented. The full homolog sequence is available by following a link from each alignment. Just to retrieve one set of similar sequences from this tool requires a significant amount of human effort in following each link, extracting the 1000-5000 bases of the DNA sequence around the alignment and integrating the data from each extraction to form the final result of one BLAST search.

Challenges of Data Extraction and Data Integration   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 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 services 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 (Liu, Pu, & Han, 1999). Several projects have implemented hand-coded wrappers for a variety of sources (Haas, Kossmann, Wimmers, & Yan, 1997; Bayardo, Jr. et al., 1997; Li et al., 1997; Knoblock et al., 1998). 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 of semantic description of their

information content, and the difficulty in making all applications in one domain 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 describe in Figure 1. 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 (DISL Group, Georgia Insitute of Technology, 2000; Sahuguet & Azavant, 1999; Baumgartner, Flesca, & Gottlob, 2001) 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, outerface, 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 (W3C, 2003).

• 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 (Berkeley, 2003), allowing users to interactively manage different components of a wrapper and the interaction between them.

In the following sections, we first give an overview of the XWRAPComposer system architecture, and then describe some important design and development efforts, using our motivating scenario described in this section as our application environment. Finally, we describe the status of the XWRAPComposer system development and discuss the future work in Section .
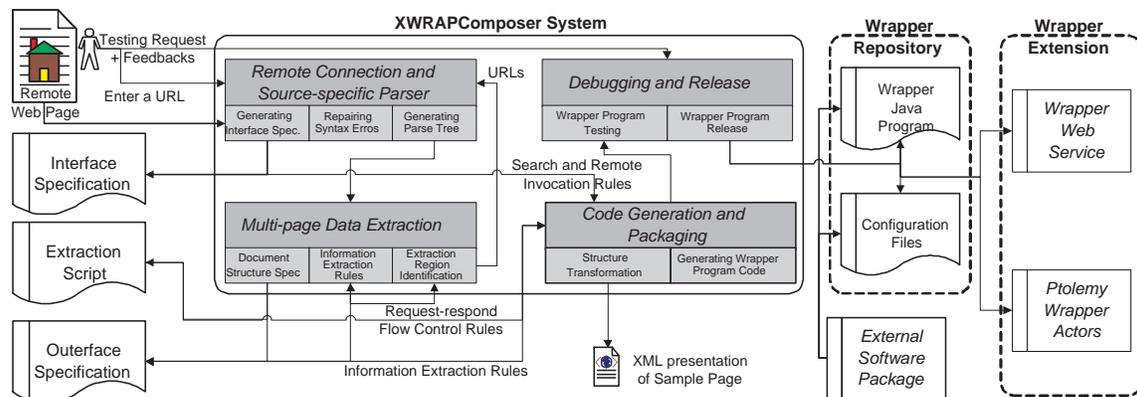
The Design Framework

Figure 2. XWRAPComposer System Architecture

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 2. 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 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 2 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. The 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 an XWRAPComposer plugin such as HTML TIDY (Raggett, 1999; W3C, 1999). 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). Namely, the wrapper will read the web page one block[1] 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 the XWRAPElite (DISL Group, Georgia Insitute of Technology, 2000) toolkit, a single page data extraction service developed by the XWRAP team at Georgia Tech. At the end of this phase, XWRAPComposer 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; and (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 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.

The XWRAPComposer wrapper generator takes the following three inputs: interface

---

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

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 next section, we focus our discussion primarily on multi-page data extraction component of the XWrapComposer, and provide a walkthrough example to illustrate the multi-page extraction process, including a brief description of the wrapping interface and remote invocation component as the necessary preprocessing step for information extraction, a short summary of code generation as the postprocessing for the multipage extraction.

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 Section .
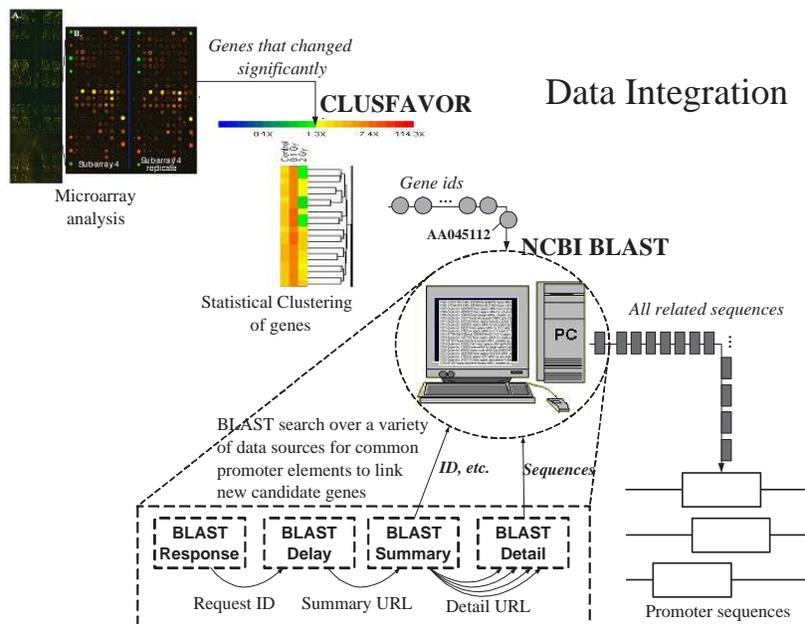
Figure 3.   A Scientific Data Integration Example Scenario

Recall the workflow presented in Figure 1, 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 3 shows the workflow of how a BLAST service request to NCBI will be served. It consists of four steps: (1) BLAST response step presents the user with a request ID. (2) BLAST delay step presents the user with the time delay for the result. (3) BLAST Summary presents the user with an overview of all gene IDs that
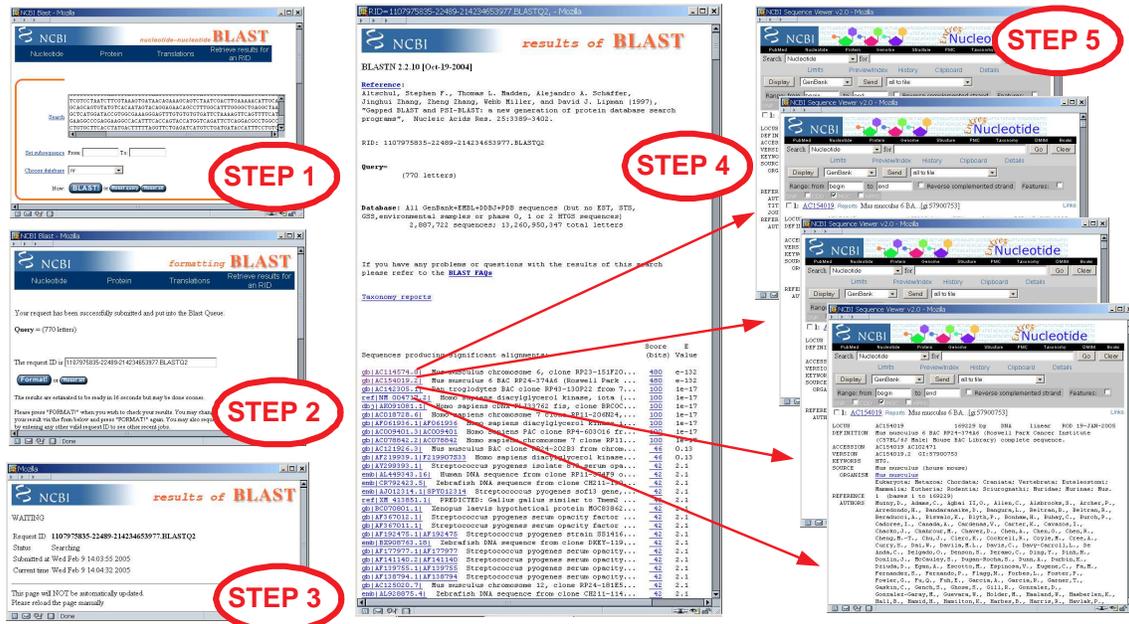
Figure 4. Multipage query with an NCBI web site

match well with the given gene sequence id. And finally, (4) 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 4 illustrates a typical BLAST query using the NCBI service (NCBI, 2003). A BLAST query involves five 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 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 3 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 an 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.

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.

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, 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 5 shows a fragment of the interface and outerface description of an example NCBI BLAST summary wrapper (LDRD Team, 2004).

Multi-page Data Extraction Script

The 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.

The general usage of commands is as follows:

```
<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 5. An Example Of Interface And Outerface Specification – NCBi Summary

```
Generate <object id> ::  <command name> (<input id>) {
  Set <property1 name> { <value> } [more value]
  Set <property2 name> { <value> }
  /* if the command is data extraction.  */
  [extraction code]
}
```

Where <object id> is the id of the output object from the command, <input id> is the id of the input object. Both input and output objects are XML nodes. For example, FetchDocument returns the content of a Web page, which is a text node in XML. Each

command specifies a set of built-in properties. <value> can be a string value, enclosed by a pair of quotes, such as *"this is a string value"*, or an XPath expression, enclosed by a pair of brackets, such as *[detailLink/text()]@<xpathroot>*. The value of "xpathroot" should be either <input id> or <object id> generated from previous commands.

If the command is used for data extraction, such as extractLink and extractContent Extraction code, the detail extraction logic needs to be specified. The main command type for the extraction script is grab functions. XWRAPComposer also provides miscellaneous commands for request-respond flow control, process management and Boolean comparison. In order to output XML data more flexibly, an XSL style sheet may be applied to any XML object using the ApplyStyleSheet command. Table 1 shows a list of commands that are currently supported in the first release of the XWRAPComposer toolkit (DISL Group, Georgia Insitute of Technology, 2003).

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

Figure 6 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 *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 (Buttler, Liu, & Pu,

2001; Wei, 2003) for further detail.

```
/* 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 6.   Extraction Script Example For NCBi Summary

Code Generation

XWRAPComposer generate its wrapper programs in two steps. First, it reads the user specified interface, outerface and composer script, and generates an XWRAPComposer wrapper, which contains the Java source code, an executable Java program, and a set of

configuration files. The configuration files include the input and output schemas obtained from interface and outerface specification of the wrapper, and the resource files used in the data extraction phase such as XSLT files. Concretely, the code generation phase consists of three main functions, as shown in Figure 2. The code generation process starts with reading the interface specification and generating the code for search interface construction, followed by generating the remote invocation method to establish the remote connection. Then, the code generator will generate the Java code that implements the request-respond flow control logic described in the composer script. For each possible request-respond state, the code for parsing the corresponding respond page will be generated. Furthermore, based on the extraction logic specified for each of the possible respond pages, we can generate the data extraction code fragment for each respond page and generate the glue code to compose the list of single page data extraction code into a multi-page data extraction routine. The third functional component is to generate debugging and release code to support an iterative process of testing, fixing bugs, re-packaging, and release. An XWRAPComposer user may feed a series of input pages to the debugging and release module to debug the wrapper program generated by XWRAPComposer. For each input page, the debugging module will automatically go through the search interface construction, remote connection establishment, document parsing, and multi-page data extraction to check if the expected output (specified in the outerface description) is returned. Once the user is satisfied with the test results, he or she may choose to release the generated wrapper program, which contains the Java source code, configuration files, the release version number, the required jar files (Java executables), and the user manual.

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 the 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 an HTTP POST command, an 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.

• The **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 a tag name to the object based on the tagging rules encoded in the composer script.

• The **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.
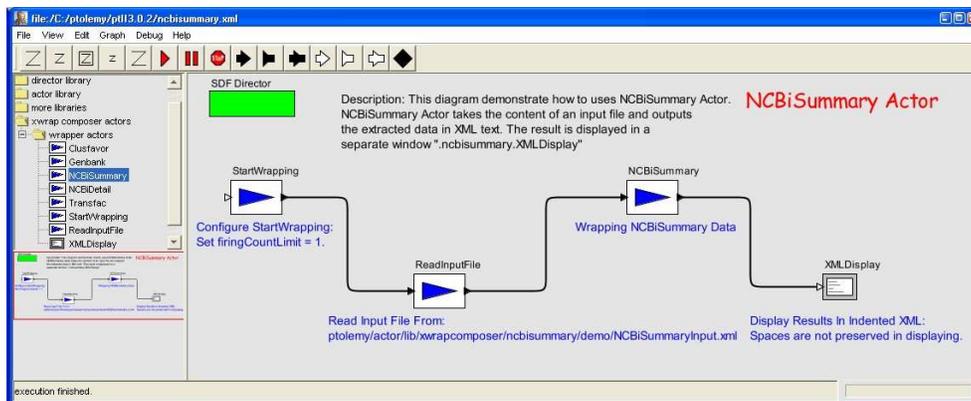
Figure 7. NCBI Blast Summary Wrapper

Figure 7 and Figure 8 demonstrate two XWRAPComposer wrappers and their mini-workflow structure. The GUI interface is developed using Ptolemy (Berkeley, 2003) (a process modeling tool). Each wrapper can be used as a Ptolemy actor (see the left menu on the screen shot) and is composed of four steps: **StartWrapping** initiates all the environment parameters, and triggers **ReadInputFile** to read a gene ID from a specified input file. The gene ID will then be sent to **NCBiSummary** Wrapper actor which performs the wrapping function upon receiving a BLAST service request with the given gene ID, and returns the set of ids of related genes as results. The last step is **XMLDisplay**, which pops
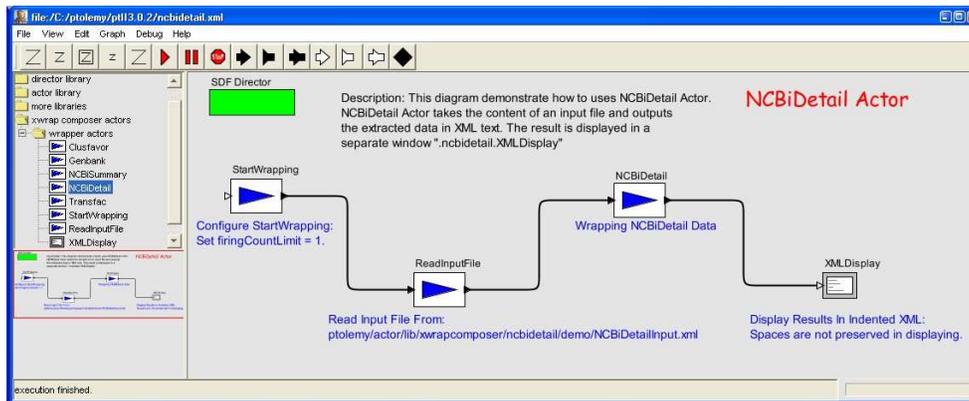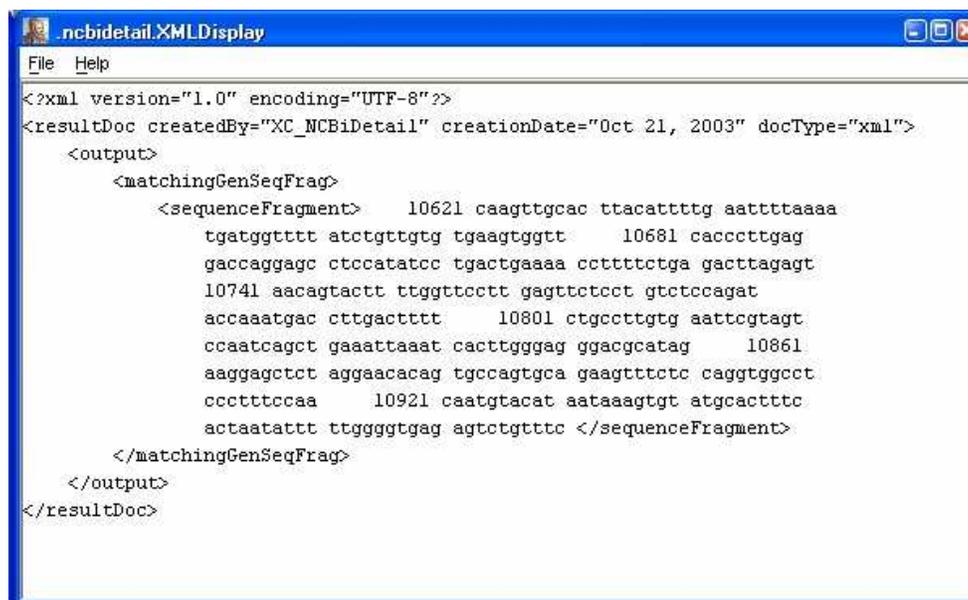
Figure 8.   NCBI Blast Detail Wrapper

up a window to present the wrapping results. Figure 9 and Figure 10 show the result of NCBI BLAST Summary wrapper and NCBI BLAST Detail wrapper respectively.

Figure 9.   Ptolemy Wrapper Actor Result Example – NCBi Blast Summary

Figure 10.   Ptolemy Wrapper Actor Result Example – NCBi Blast Detail

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. This approach provides end users with the flexibility of customizing their systems by using XWRAPComposer wrapper programs as building block. However, end users have to use Java programming languages for their system implementation because the generated XWRAPComposer wrapper programs are in Java. To free the end-user from the reliance on a chosen programming language like Java, we want XWRAPComposer to be able to generate WSDL-enabled wrappers to allow each wrapper program to be used as a Web service (W3C, 2002), which is our second objective. We chose Web services because it was proposed and has been successfully adopted by many systems for providing platform-independent and programming language-independent service access. End users can implement their client applications with full flexibility as long as their systems can access our server using SOAP protocol. 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. In this sense, XWRAPComposer wrappers are working as service providers to end users. When they interact with wrapped data sources, those XWRAPComposer wrappers act as the clients of those services. Second, to ease the implementation and deployment of XWRAPComposer

wrappers as Web services, we incorporate a WSDL generator to automatically generate Web service description by binding the wrapper's interface and outerface with the servlet configuration. Figure 11 shows the extensions added to the XWRAPComposer to produce wrappers as WSDL web services.
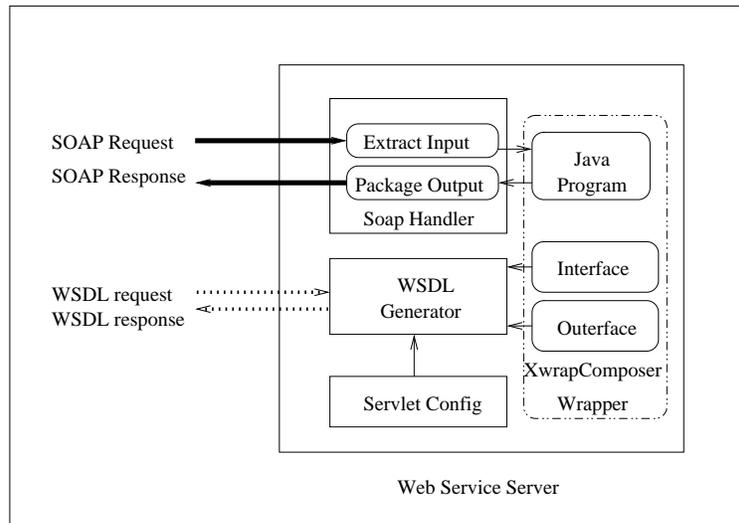

Figure 11. Web-service Enabled Wrappers

Figure 12. XWRAPComposer online wrapper repository

Wrapper Program Repository

As a part of the XWRAPComposer effort, we design and develop an online wrapper generation and registration system to assist the usage of XWRAPComposer wrappers and simplify the wrapper generation and management overhead. All wrappers generated by

XWRAPComposer can be registered directly into our online wrapper repository. A snapshot of this repository is shown in Figure 12.


```
- <Alignment>
  - <geneLink>
      href="
      <Target>http://www.hgmp.mrc.ac.uk/</Target>
      "
  </geneLink>
    <score>Score = 46.1 </score>
    <Evalue>Expect = 0.001 </Evalue>
  - <QueryString>
      <AlignmentName>Query:</AlignmentName>
      <m>117</m>
      <Sequence>gtcctcctcctccatctctttct</Sequence>
      <n>139</n>
  </QueryString>
    <Alignments> |||||||||||||||||||</Alignments>
  - <SequenceString>
      <AlignmentName>Sbjct:</AlignmentName>
      <m>22106</m>
      <Sequence>gtcctcctcctccatctctttct</Sequence>
      <n>22128</n>
  </SequenceString>
</Alignment>
```

Figure 13.   XWRAPComposer wrapper execution result: An example

Consider the first wrapper shown in Figure 12.   The target service provider is http://fugu.hgmp.mrc.ac.uk/blast/.   It provides a standard BLAST interface. After obtaining the XWRAPComposer wrapper source code and jar file, the user can upload this wrapper through an online registration interface, available at http://disl.cc.gatech.edu/ldrdscript/html/registerwrapper.htm. One can download the generated wrapper source code directly by clicking on wrapper code column of the corresponding target service provider. Using the XWRAPComposer library and the composer scripts that we released, this wrapper source code can be compiled on the user's local machine and executed as command line Java application. A user can also use our online wrapper execution interface to execute each registered wrapper either as a servlet or a web service. An example of online execution result is given in Figure 13. All XWRAPComposer wrappers for BLASTN services are presenting a uniform interface to the end users, which facilitates the large scale integration of multiple BLASTN services.

Related Work

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 (Westbrook & Bourne, 2000). Providing integrated and uniform access to these databases has been a serious research challenge. Several efforts (Critchlow, Fidelis, Ganesh, Musick, & Slezak,

2000; Davidson et al., 1999; Goble et al., 2001; Haas et al., 2001; McGinnis, 1998; Siepel et al., 2001) have sought to alleviate the interoperability issue, 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 (Wiederhold, 1992; Liu & Pu, 1997; Liu, Pu, & Lee, 1996). For instance, the most documented information mediator systems (e.g., Ariadne (Knoblock et al., 1998), CQ (Liu, Pu, & Tang, 1999; Liu et al., 1998), Internet Softbots (Kushmerick, Weld, & Doorenbos, 1997), TSIMMIS (Garcia-Molina et al., 1997; Hammer et al., 1997), Araneus (Atzeni, Mecca, & Merialdo, 1997)) all assume a pre-wrapped set of web sources. However, developing and maintaining wrappers by hand is labor intensive and error-prone, due to technical difficulties such as undocumented HTML/XML tags and subtle variations in the content (small to the human perception, but difficult for programs).

Tova Milo and Sagit Zohar (Milo & Zohar, 1998) use schema matching to simplify the wrapper generation, when both the source schema and the result schema are available. They observe that in many cases the schema of the data in the source system is very similar to the result schema. In such cases, much of the translation work can be done automatically based on the schema similarity. They define a middleware schema, and each data source to be used in their system needs a mapping of its data and schema to (or from) the middleware format. They develop an algorithm to match and translate the objects in the source with the objects in the result comparing the two instances of the middleware schema. Since most Web pages are still in schema-less HTML, schema matching does not work. However, if more XML information appears on the Web, this approach will speed up the wrapper generation, since XML documents contain schema information.

SoftBot (Kushrnerick, 1997) developed a wrapper generation system using inductive learning techniques. Several generic wrapper classes with adjustable parameters are pre-defined in the wrapper generation system. Each wrapper class can extract information from one document pattern. Wrapper developers highlight interesting sections in many sample documents, and then a machine learning algorithm will adjust those parameters to find a combination of wrapper classes to extract the highlighted sections correctly. If such a combination is not available, the algorithm will return the best combination with the fewest mistakes. The developers can either correct the best combination manually or add more wrapper classes fitting new patterns to find a complete correct combination.

NoDoSE also adopts the inductive learning technique. Using a GUI, the user hierarchically decomposes a plain text file, outlining its regions of interest and then describing their semantics. The task is expedited by a mining component that attempts to infer the grammar of the file from the information the user has identified so far.

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 a 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. Most 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.

Conclusion

Both enterprise systems and science and engineering integration applications require gathering information from multiple, heterogeneous information services. Although Web service technology such as WSDL, SOAP, and UDDI, has provided a standardized remote invocation interface, there exist other types of heterogeneity in terms of query capability, content structure, and content delivery logics due to inherent diversity of different services. A popular approach to handling such heterogeneity is to use wrappers to serve as mediators to facilitate the automation of collecting and extracting data from multiple diverse data providers.

We have described a service-oriented framework for development of wrapper code generators and a concrete implementation, called XWRAPComposer, to evaluate our framework in the context of bioinformatics applications. 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 request-respond flow 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.

The XWRAPComposer project continues along three dimensions. First, we are interested in extending XWRAPComposer code generation capability to allow wrappers to be generated for a wide variety of complex data sources. Second, we are interested in exploring data provenance techniques for large scale data integration. In particular, we are interested in extracting data provenance information from the vast amount of data contents provided in many scientific data service providers. We believe that the data provenance information is critical for facilitating the scientific data integration process, and improving scientific data integration quality. Third but not the least, we are working on providing user-friendly GUI to support for interactive specification of interface, outerface and composer script.

Acknowledgements

References

Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., et al. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research. 25 (1997) 3389-3402.*

Atzeni, P., Mecca, G., & Merialdo, P. (1997). Semi-structured and structured data in the web: going back and forth. In *Proceedings of ACM SIGMOD Workshop on Management of Semi-structured Data.* Tucson, Arizona.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual web information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB).* Rome, Italy.

Bayardo, Jr., R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., et al. (1997). Semantic integration of information in open and dynamic environments. In *Proceedings of ACM SIGMOD Conference.* Tucson, Arizona.

Berkeley. (2003). Ptolemy group in eecs. *http://ptolemy.eecs.berkeley.edu/.*

Buttler, D., Liu, L., & Pu, C. (2001). A fully automated object extraction system for the world wide web. In *Proceedings of the 2001 International Conference on Distrubuted Computing Systems ICDCS.* Phoenix, Arizona, USA.

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

Davidson, S., Buneman, O., Crabtree, J., Tannen, V., Overton, G., & Wong, L. (1999). Biokleisli: Integrating biomedical data and analysis packages. *Bioinformatics: Databases and Systems, S. Letovsky, Editor, Kluwer Academic Publishers, Norwell, MA:201-211.*

DBCAT. (1999). The public catalog of databases. *http://www.infobiogen.fr/services/dbcat.*

DISL Group, Georgia Insitute of Technology. (2000). XWRAP Elite Project. *http://www.cc.gatech.edu/projects/disl/XWRAPElite.*

DISL Group, Georgia Insitute of Technology. (2003). XWRAPComposer. *http://www.cc.gatech.edu/projects/disl/XWRAPComposer/.*

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., et al. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems, 8(2), 117-132.*

Goble, C. A., Stevens, R., Ng, G., Bechhofer, S., Paton, N., Baker, P. G., et al. (2001). Transparent access to multiple bioinformatics information sources. *IBM Systems Journal, 40(2):532-551.*

Haas, L., Kossmann, D., Wimmers, E., & Yan, J. (1997). Optimizing queries across diverse data sources. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB).* Athens, Greece.

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

Hammer, J., Brennig, M., Garcia-Molina, H., Nesterov, S., Vassalos, V., & Yerneni, R. (1997). Template-based wrappers in the tsimmis system. In *Proceedings of ACM SIGMOD Workshop on Management of Semi-structured Data.* Tucson, Arizona.

Knoblock, C. A., Minton, S., Ambite, J. L., Ashish, P. J. M. N., Muslea, I., Philpot, A. G., et al. (1998). Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Madison, WI, USA.

Kushmerick, N., Weld, D. S., & Doorenbos, R. (1997). Wrapper induction for information extraction. In *Proceedings of Int. Joint Conference on Artifical Intelligence (IJCAI).* Nagoya, Japan.

Kushrnerick, N. (1997). *Wrapper induction for information extraction.* Unpublished doctoral dissertation, University of Washington.

LDRD Team. (2004). LDRD Project. *http://www.cc.gatech.edu/projects/disl/LDRD.*

Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J., et al. (1997). Capability based mediation in tsimiss. In *Proceedings of ACM SIGMOD Conference.* Tucson, Arizona.

Liu, L., & Pu, C. (1997). An adaptive object-oriented approach to integration and access of heterogeneous information sources. *Distributed and Paralle Databases: An International Journal, 5*(2), 167-205.

Liu, L., Pu, C., & Han, W. (1999). XWrap: An Extensible Wrapper Construction System for Internet Information Sources. In *Technical report, ogi/cse, feb.*

Liu, L., Pu, C., & Lee, Y. (1996). An adaptive approach to query mediation across heterogeneous databases. In *Proceedings of the International Conference on Coopertive Information Systems.* Brussels, Belgium.

Liu, L., Pu, C., & Tang, W. (1999). Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering, 11*(4), 610-628. (Special Issue on Web Technology)

Liu, L., Pu, C., Tang, W., Biggs, J., Buttler, D., Han, W., et al. (1998). CQ: A Personalized Update Monitoring Toolkit. In *Proceedings of ACM SIGMOD Conference.* Seattle, USA.

McGinnis, S. (1998). (Genbank User Services, National Center for Biotechnology Information (NCBI), National Library of Medicine, US National Institute of Health). *Personal Communication, 1, Jan.*

Milo, T., & Zohar, S. (1998). Using schema matching to simplify heterogeneous data translation. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB).* New York City, USA.

NCBI. (2003). National center for biotechnology information – blast databases. *http://www.ncbi.nlm.nih.gov/BLAST/.*

Peterson, L. (2002). *CLUSFAVOR.* Baylor College of Medicine. See http://mbcr.bcm.tmc.edu/genepi/.

Quandt, K., Frech, K., Karas, H., Wingender, E., & Werner, T. (1995). MatInd and MatInspector: new fast and versatile tools for detection of consensus matches in nucleotide sequence data. *Nucleic Acids Research. 23 (1995) 4878-4884.*

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

Sahuguet, A., & Azavant, F. (1999). WysiWyg Web Wrapper Factory (W4F). In *Proceedings of World Wide Web (WWW) Conference.* Orlando.

Siepel, A. C., Tolopko, A. N., Farmer, A. D., Steadman, P. A., Schilkey, F. D., Perry, B., et al. (2001). An integration platform for heterogeneous bioinformatics software components. *IBM Systems Journal, 40(2):570-591.*

Stein, L. D., & Thierry-Mieg, J. (1999). Scriptable access to the Caenorhabditis elegans genome sequence and other ACEDB databases. *Genome Res. 8 (1999) 1308-1315.*

W3C. (1999). Reformulating HTML in XML. *http://www.w3.org/TR/WD-html-in-xml/.*

W3C. (2002). Web services. *http://www.w3c.org/2002/ws/.*

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

Wei, H. (2003). *Wrapper Application Generation for Semantic Web: An XWRAP Approach.* Unpublished doctoral dissertation, Georgia Institute of Technology.

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

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer, 25*(3), 38-49.

ABOUT THE AUTHOR

Ling Liu is an associate professor at the College of Computing, Georgia Institute of Technology. There, she directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining research issues and technical challenges in building large scale distributed computing systems that can grow without limits. Dr. Liu and the DiSL research group have been working on various aspects of distributed data intensive systems, ranging from decentralized overlay networks, exemplified by peer to peer computing, data grid computing, to mobile computing systems and location based services, sensor network computing, and enterprise computing systems. She has published over 150 international journal and conference articles. Her research group has produced a number of software systems that are either open sources or directly accessible online, among which the most popular ones are WebCQ and XWRAPElite. Most of Dr. Liu's current research projects are sponsored by NSF, DoE, DARPA, IBM, and HP. She is on the editorial board of several international journals, such as IEEE Transactions on Knowledge and Data Engineering, International Journal of Very large Database systems (VLDBJ), and International Journal of Web Services Research. She has chaired a number of conferences as a PC chair, a vice PC chair, or a general chair, including IEEE International Conference on Data Engineering (ICDE 2004, ICDE 2006, ICDE 2007), IEEE International Conference on Distributed Computing (ICDCS 2006), IEEE International Conference on Web Services (ICWS 2004)

Jianjun Zhang is currently a Ph.D. student in the College of Computing at Georgia Institute of Technology. His research interests include web services, wide-area networked computing systems and applications. Jianjun Zhang received his M.E. and B.S. from Department of Computer Science, Wuhan University, China in 1999 and 1996. His Ph.D. dissertation research is focused on efficient and reliable multicast techniques in large scale networked computing systems.

Wei Han received his Ph.D. in 2003 at College of Computing, Georgia Institute of Technology. His Ph.D. research was on automated wrapper code generation. Wei Han received his B.S. from Tsinghua University, China in 1997 and his M.S. of Computer Science and Engineering from Oregon Graduate Institute in 1999. Now, he is working for IBM Research, Almaden Research Center.

Calton Pu holds the position of Professor and John P. Imlay, Jr. Chair in Software at Georgia Institute of Technology. He has published extensively in operating systems, transaction processing, and Internet data management. His current research is mainly sponsored by NSF, DARPA, Department of Energy, HP and IBM. He received his Ph.D. in Computer Science from University of Washington in 1986. He is a member of ACM, a senior member of IEEE, and a fellow of AAAS.

James Caverlee is currently a Ph.D. student in the College of Computing at the Georgia Institute of Technology. His research interests are focused on web information management and retrieval, Web services, and Web based data intensive systems and applications. James graduated magna cum laude from Duke University in 1996 with a B.A. in Economics.

He received the M.S. degree in Engineering-Economic Systems & Operations Research in 2000, and the M.S. degree in Computer Science in 2001, both from Stanford University. His Ph.D. dissertation research is focused on Web Information Retrieval and Web data mining, including efficient and spam-resilient Web search algorithms.

Sungkeun Park was a master student of College of Computing at Georgia Institute of Technology and a member of Distributed Data Intensive Systems Lab (DiSL). Sungkeun has worked on reputation trust in peer-to-peer systems and Web service enabled wrapper code generation systems.

Terence Crithchlow is the team lead for the BioEncyclopedia effort within the Biodefense Knowledge Center (BKC) at the Lawrence Livermore National Lab (LLNL). In this capacity, Dr. Crithchlow is leading a large team of researchers and developers in an effort to integrate relevant biodefense information into a consistent environment for use by BKC analysts. Prior to working with the BKC, Dr. Critchlow led several research projects focusing on improving scientists' interactions with large data sets. He obtained a Ph.D. in Computer Science from the University of Utah in 1997 and has been a member of the Center for Applied Scientific Computing at LLNL ever since.

David Buttler received his Ph.D. in 2003 at the College of Computing, Georgia Institute of Technology. He has been working in the Data Science group at Center for Applied Scientific Computing, Lawrence Livermore National Laboratory Lab (LLNL) since his Ph.D. graduation. His research interests are in information management for distributed data intensive systems. In particular, he is interested in information discovery, update monitoring, and source selection. Dr. Buttler earned a B.Sc. in Computer Science from the University of Alberta in 1998, and a B.S. in Mathematics from Andrews University in 1995.

Matthew Coleman Dr. Matthew Coleman is currently a Project Leader in the Biomedical Division within Biosciences at Lawrence Livermore National Laboratory (LLNL), working on understanding ionizing radiation effects by comparing gene expression and proteomic changes in exposed cells. His technical training is in molecular biology where he received his B.A. from the University of Massachusetts in 1987, and his Ph.D. in biophysical studies of proteins from Boston University in 1997. Dr. Coleman then completed a post-doctoral fellow at LLNL developing and applying genomic technologies. He has authored over 50 publications in peer-reviewed journals, published abstracts and book chapters covering a diverse breadth of molecular and biochemical biology as well as bioinformatics. He is the key domain scientist of the DoE SciDAC project and the XWRAPComposer project.

Appendix
XWRAPComposer Extraction Script Command

The first release of the XWRAPComposer supports seven categories of command. They are listed as following.

(1) Document Retrieval Command

We support the following Document Retrieval Commands in the first release of the XWRAPComposer toolkit.

ConstructHTTPQuery   This command constructs an HTTP query that contains three components: URL, queryString and HTTP method. It has four properties: URL, queryString, httpMethod and vars. The first three properties are actually templates with placeholders, "$$". The last property is a list of strings to replace the placeholders.
Example:

```
Generate genbankSummaryPage ::  ConstructHttpQuery {
  Set inputSource {
    Set url { "http://www.amazon.com/book-search.cgi" }
    Set queryString { "keyword=$$&author=$$&start=1" }
    Set httpMethod { "post" }
    Set vars {
    /* the sub property names are only for reading.  */
    /* We replace the placeholders by the order of the vars.  */
      Set keyword { "java" }
      Set author { "john" }
      }
    }
  }
}
```

The result will be

```
<inputSource>
    <URL>http://www.amazon.com/book-search.cgi</URL>
    <queryString>keyword=java&author=john&start=1</queryString>
    <httpMethod>get</httpMethod>
<inputSource>
```

ConstructFileQuery   This command constructs a file request, which contains only one property, file name.

FetchDocument   This command takes a file request or an HTTP request as input and returns the content of the file or the Web page. It does not have any properties.

(2) Data Extraction

Currently we support two types of data extraction commands. They are used for extracting links or extracting content.

ExtractLink   This command indicates to extract an HTTP request from the input. It usually needs to extract URL, queryString and httpMethod. If queryString and httpMethod are not extracted, the default values will be used. The default httpMethod is "get" and queryString is "". The return object of ExtractLink can be the input of FetchDocument.

ExtractContent   This command indicates to extract content from the input, which contains all kinds of data. The extraction method needs to be specified with GrabFunctions and XML construction commands as well as other commands.

(3) Grab Function

The Grab function is designed to facilitate the text parsing and string analysis during the extraction process. We support the following four grab functions.

GrabSubstring   Assuming the input is a text node, this commands extracts a substring by two properties, beginMatch and endMatch. It will return the string between beginMatch and endMatch. If there are multiple result strings, we only choose the first one.

GrabXWrapEliteData   This command applies an XWRAPElite wrapper to the input. The input should be a text node that represents the content of a Web page. The properties are generated by the XWRAPElite toolkit. It allows modification for fine-tuning.

GrabCommaDelimitedText   This command extracts comma-delimited data into XML. It has the following properties.
- *LineDelimiters:* The delimiters to separate data into rows. The default is the system line separators.
- *Delimiters:* The delimiters to separate data in a row to a list of cells. The default is comma.
- *StopStrings:* String that will be ignored.
- *Filters:* It contains two sub properties, minColCount and maxColCount. We will filter out all the rows whose column numbers are not in the range of [`minColCount`, `maxColCount`] inclusively.
- *RowOutput:* It specifies how to output the tabular data for each row in XML.

GrabConsecutiveLines   This command is to extract consecutive text lines from the input. It has three properties, beginMatch, endMatch and matchingMethod. The default matching method is to match the first string of the beginning line and the ending line with beginMatch and endMatch properties. However, we might need some domain specific matching method in some cases. Then we can use external functions as shown in NCBiDetail example.

(4) Boolean comparisons

ContainSubstring   This command returns a Boolean value, which indicates if the input contains a substring. It has one property, compSubstring.
Example:

```
ContainSubstring (answerPage) {
    Set compSubstring { "This page will be automatically updated in"}
}
```

The example demonstrates a command that checks if the text value of answerPage contains a string of "This page will be automatically updated in".

(5) Control Flow

While $\cdots$ **Do** $\cdots$ This command checks the conditions in the while clause, while the conditions are true, repeat the do clause. The while clause contains Boolean commands and the do clause contains other extraction-related commands.

(6) Post Processing

ApplyStylesheet This command applies a style sheet to the input XML. It has a property, StyleSheetFile.

(7) Process Management

Sleep   This command pauses the process for certain time.
    Usage: Sleep "<number of milliseconds>"