ELIXIR: A library for writing wrappers in Java

Edward Wild

Undergraduate Honors Thesis, Fall 2001

Supervising Professor: Raymond J. Mooney

Introduction

The use of information from the Internet is becoming more prevalent in applications. LIBRA (Mooney and Roy, 1999), WhizBang!Labs (http://www.whizbanglabs.com), and mySimon (http://www.mysimon.com) are examples of applications and companies that use information from the Internet. LIBRA uses book data from Amazon.com to recommend books to users, and mySimon uses information taken from various online stores to help users find the lowest price for an item. Available tools to help extract information from the Internet include the WysiWig Web Wrapper Factory (Sahuguet and Azavant, 1998), WebL (Kistler and Marais, 1998), and SPHINX (Miller and Bharat, 1998). This paper presents ELIXIR, a set of utilities that helps users to write programs that extract information from the Internet.

Wrappers are programs that retrieve documents from the Internet and extract information from them. They are a way to translate information on a web page designed to be seen into a format that can be used by a computer program. Typically it is easy for a programmer to describe how to find information on a page by using pattern matching or regular expressions. The difficulty with writing wrappers in programming languages such as Java lies in the fact that, in the finished wrappers, the description of where the information is located is often hidden inside a routine that is concerned with the string operations necessary to find and extract that information. This makes large wrappers difficult to write, understand, and maintain.

A solution to the problem mentioned in the above paragraph would be to find a way to separate the description of where information is from the operations needed to extract that information through the use of a library. Such a separation would allow the programmer writing or maintaining the wrapper to focus on the location of information, rather than the steps needed to extract information. For example, the programmer only needs to encode descriptions such as "the string between 'Capital:' and '
'," rather than algorithms like "find the string 'Capital:' and extract everything from the end of that string until the string '
'."

To separate the description of where information is from the algorithm used to access it the library must clearly indicate how the description is used to form an algorithm. For example, does the description "the string between 'Capital:' and '
"' mean to find the first occurrence of 'Capital:' and extract everything until the final occurrence of '
'? Or does the description mean to find the first occurrence of '
' and then extract everything from the closest preceding occurrence of 'Capital:' until that point?

The library should also give the user enough power to describe the location of the information to be extracted. If there is information that the library cannot extract, it should allow the user to define the algorithm to extract that information in such a way that the separation between the description and the algorithm is maintained.

The Extensible Library for Information Extraction on Internet Resources (ELIXIR) is a library for writing

Figure 1: CIA World Factbook country listing

wrappers that can extract text from a web page and follow links on that page. It contains a set of objects that the user combines to describe the location of information on a web site. Those objects automatically perform the operations necessary to extract information.

Description

Text on the Internet can be described by its position on a web page, and that web page's location on the Internet. It is useful to describe both locations in terms of a hierarchy rooted at some page. For example, consider the CIA World Factbook. The capital of the United States can be described as the string found by downloading the country listing from http://www.cia.gov/cia/publications/factbook/menugeo.html (see figure 1), following the link with anchor text "United States," and extracting the text that appears after the string "Capital:" and before the next new line. This method generalizes to find the capital of every country listed in the CIA World Factbook by following each link to a country from http://www.cia.gov/cia/publications/factbook/menugeo.html instead of only following the link for the United States. One can also find additional information about each country by specifying the appropriate patterns.

ELIXIR allows users to construct a tree model of the data they wish to extract. Each node in the tree has a function that maps a string to a new string. At each node, an input string is received from the parent, the

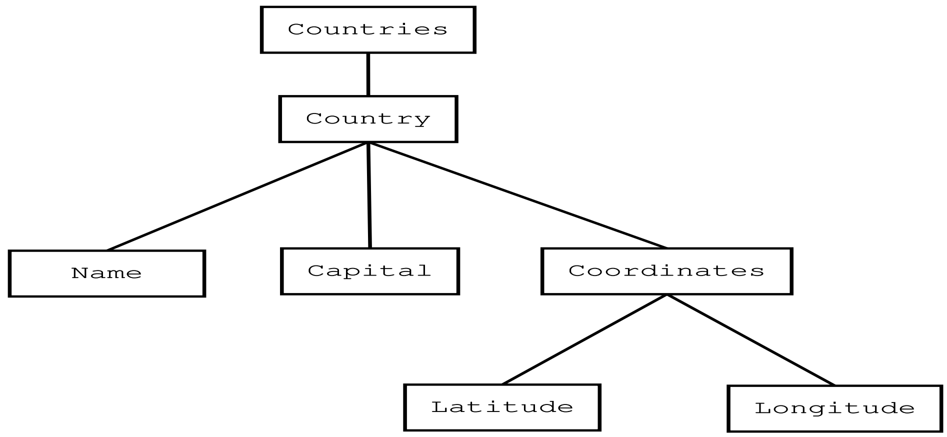


Figure 2: Rule tree for World Factbook

```
<table width="100%" cellspacing="6">
<tr>
<td valign="top" width="20%" align="right"><font face="arial" size="2"><b>Capital:</b></font></td>
<td valign="top"><font face="arial" size="2">
Washington, DC
</font></td>
</tr>
</table>
```

Figure 3: HTML source from the CIA World Factbook containting the capital of the United States (from http://www.cia.gov/cia/publications/factbook/geos/us.html)

function is applied to that string, and the output is passed along to the children. At the leaf nodes, the result of the function is stored as the value of the node. The tree thus represents the structure of the data and the instructions needed to extract the data. Consider a task that requires that a database be built that has records for each country's name, capital, and geographic coordinates (see figure 2). The root node of a tree for this task downloads the country listing at http://www.cia.gov/cia/publications/factbook/menugeo.html. The root node has a child node which takes the country listing and extracts the URL to each country. The page referenced by this URL is downloaded and passed along to the current node's children. There is a child that extracts the country's name, a child that extracts the country's capital, and one that extracts the countries geographic coordinates.

The function associated with each node is represented as a stack of functions called item extractors. Each item extractor maps a string to a (string, integer) pair where the string is used as input to the next item extractor on the stack and the integer is the position in the input string following where the string matched. The output of the function for an input is obtained by iterating through the stack of item extractors. The input for each item extractor is the output from the previous item extractor. The output of the function is the value of the last, or bottom, item extractor. A function is said to match when its output is not the empty

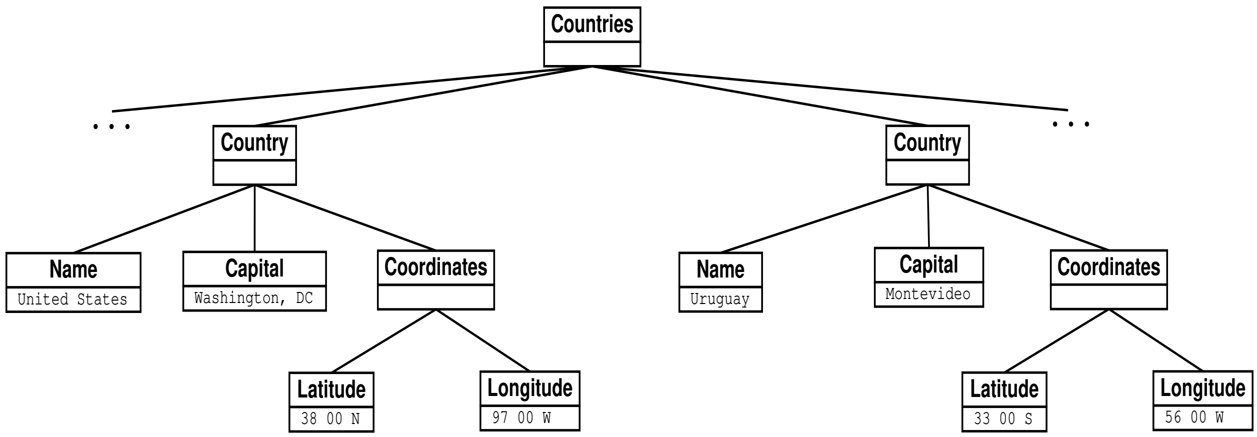


Figure 4: Attribute tree for World Factbook

string. Take, for example, a function that takes as input a page with information on a country and outputs the capital of that country (see figure 3). The function may be defined using two item extractors. The item extractor on the top of the stack extracts the string between the strings “Capital:” and “</table>.” This value is the input to the item extractor on the bottom of the stack, which extracts the string between “” and “” and returns the capital of the country. When this function is applied to the country information for the United States, the top item extractor returns “</td> <td valign="top"> Washington, DC </td> </tr>” and the bottom one applied to this string returns “Washington, DC.”

Each node in the tree is called a rule. Associated with each rule is a name, a stack of item extractors, a cardinality, and a list of children. In our example, one rule is named “country.” This rule has item extractors that extract the part of the page containing the desired information, a cardinality that specifies that as many countries as possible should be extracted, and child rules that extract the capital, name, and geographic coordinates of the country. The output of a rule is determined by trying to match the stack of item extractors to an input string until the item extractor at the top of the stack fails to match, or the rule has matched as many times as its cardinality will allow. The cardinality of a rule can be any integer greater than zero, or a special value that indicates that the top item extractor should match as many times as possible. The output of a rule is passed on to its children each time that the item extractors match.

Since a rule may match more than once, it is necessary to distinguish between the tree of rules and the tree defined by the output of the rules. Consider the rules to extract information about countries from the CIA World Factbook. Rather than specify a separate rule for each country, the user can construct a rule that works for all the countries, and then indicate that the rule should be applied several times. Thus the tree of rules will contain only the rule to specify how to get information about a country, but the result of that rule will be a tree of attributes that contains information on many countries. An attribute has a name, a value, and a list of children. Its name is the name of the rule that generated it. Its value is the output of the rule that generated it, and its children correspond to the children of the rule that generated it. Thus each time a rule matches a new attribute is created for that rule. and each attribute will have its own set of children. If a rule does not match, no attribute will be generated for it. A portion of the attribute tree for the CIA World Factbook is shown in figure 4.

In order to create a method for conditional matching, a new kind of rule is necessary. This new kind of rule will have no extractors of its own, and will always have cardinality of one. It will try to match each of its children on a given input until one child matches. That child will take its parent's place in the resulting attribute hierarchy. Using this method, the user can specify a way to conditionally choose a branch based on the matching of item extractors.

Implementation

Figure 5 is a wrapper for the CIA World Factbook country listing written using ELIXIR. This section describes the implementation of ELIXIR and uses the CIA World Factbook wrapper (`WorldFactbookExample`) to show how ELIXIR can be used to write wrappers.

The basic string matching facilities are provided by classes that implement the `ItemExtractor` interface. The `ItemExtractor` interface consists of two methods, `extract()` and `item()`, which allow the `ItemExtractor` to match strings. The `extract()` method takes a string as input and returns an integer that corresponds to the position in the input string where the next attempt to match the string should begin. This method must also guarantee that subsequent calls to `item()` will return the extracted string itself. `ItemExtractor` implementations exist to perform basic string matching (`ForwardItemExtractor` and `ReverseItemExtractor`), perform regular expression matching (`RegexItemExtractor`), and retrieve web pages from URLs (`LinkFollower`).

`WorldFactbookExample` uses `ForwarItemExtractors` and `LinkFollowers`. The `LinkFollower` in line 19 will follow links relative to its argument. For example, for input `"geos/us.html"` the `LinkFollower` will return the page at http://www.cia.gov/cia/publications/factbook/geos/us.html. The `ForwardItemExtractor` in lines 28 and 29 finds the first occurence of the string `"<b>Capital:</b>"` in its input, then the next occurence of the string `"</table>"` and returns the string in between.

`ItemExtractor` provides a convenient way to extend the functionality of ELIXIR. `TextPage` provides a way to parse a string of text into tokens using an `ItemExtractor`. This allows `ItemExtractor` to be a very

```
package examples;

import wrapper.*;
import textutils.*;
import java.io.*;

public class WorldFactbookExample {

    public static void main(String[] args) {
        Wrapper w = new Wrapper() {
            protected void defineRules() {
                RuleBuilder rb = new RuleBuilder("countries",
                                                new ItemExtractor[] {new LinkFollower()});
                rb.addRuleToRule("countries",
                                 "country",
                                 new ItemExtractor[] {
                                     new ForwardItemExtractor("<li><a href=\"",
                                                              "\">"),
                                     new LinkFollower("http://www.cia.gov/cia/publications/factbook/")},
                                 Cardinality.MANY);
                rb.addRuleToRule("country", "name",
                                 new ItemExtractor[] {
                                     new ForwardItemExtractor("<font face=\"arial\" size=\"5\" " +
                                                              "color=\"ffffff\">",
                                                              "</font>")});
                rb.addRuleToRule("country", "capital",
                                 new ItemExtractor[] {
                                     new ForwardItemExtractor("<b>Capital:</b>",
                                                              "</table>"),
                                     new ForwardItemExtractor("<font face=\"arial\" size=\"2\">",
                                                              "</font>")});
                rb.addRuleToRule("country", "coordinates",
                                 new ItemExtractor[] {
                                     new ForwardItemExtractor("<b>Geographic coordinates:</b>",
                                                              "</tr>")});
                rb.addRuleToRule("coordinates", "latitude",
                                 new ItemExtractor[] {
                                     new ForwardItemExtractor("<font face=\"arial\" size=\"2\">\n", ",")});
                rb.addRuleToRule("coordinates", "longitude",
                                 new ItemExtractor[] {new ForwardItemExtractor(", ", "\n")});
                addRule(rb.rule());
            }
        };
        Label l = w.label("http://www.cia.gov/cia/publications/factbook/menugeo.html");

        new LabelPrinter().print(l, args[0]);
    }
}
```

Figure 5: Wrapper for the CIA World Factbook

simple interface, while clients of `ItemExtractor` can use `TextPage` to avoid having to manipulate strings directly. Only classes within ELIXIR use `TextPage`. Users deal directly with `ItemExtractors`. Note that `TextPage` is never mentioned in `WorldFactbookExample`.

The rules of the wrapper are represented using the `Rule` interface and its implementors `PrimitiveRule`, `CompositeRule`, and `BranchRule`. This is an implementation of the Composite design pattern (Gamma et al., 1994). `PrimitiveRules` have no children, while `CompositeRules` and `BranchRules` do. Each `Rule` has a stack of `ItemExtractors` and a `Cardinality` object that represents the maximum number of times that a rule should be matched. The `Cardinality` object may be initialized to the constant `Cardinality.MANY`, in which case the rule should be matched as many times as possible. A `Rule` matches when the result of matching each `ItemExtractor` in its stack to the string, and using the output of one `ItemExtractor` as the input for the next one results in a value that is not null. For `PrimitiveRules` this value becomes the result of the rule, while for `CompositeRules` this value is passed on to its children. `BranchRules` are evaluated specially. The input string is passed to each child rule until a child matches. Then the output from that child is considered to be the output of the `BranchRule`. The output of `Rule` is a list of `Attributes`. A list of `Attribute` trees are constructed from a `Rule` tree by the `getValue()` method of `Rule`.

To hide the fact that there are three different kinds of rules from the user, the `RuleBuilder` class is used. `RuleBuilder` is an implementation of the Builder design pattern (Gamma et al., 1994). The user may use the `RuleBuilder` to design a `Rule` without having to declare what parts of the tree are `PrimitiveRules`, `CompositeRules`, or `BranchRules`. This is accomplished by starting each new `Rule` as a `PrimitiveRule` and changing it to a `CompositeRule` or `BranchRule` when the user adds a child or branch. To facilitate changing rules, the `MutableRule` class is used. `MutableRule` is an adapter (Gamma et al., 1994) that implements the `Rule` interface by delegating all `Rule` methods to a `Rule` member. This member can be changed by clients using the `set()` method.

In `WorldFactbookExample` a `RuleBuilder` is used in the `defineRules()` method (lines 11–42). When the `RuleBuilder` is constructed (line 12) it contains a single `PrimitiveRule` named "countries" that has one `LinkFollower` that will treat its input as a URL and download the page that URL refers to. When the method `addRuleToRule()` is used to add a rule named "country" (lines 14–20), the rule named "countries" is changed to a `CompositeRule` that has a `PrimitiveRule` named "country" as a child. Note that the user does not have to indicate which rules should be `PrimitiveRules` and which should be `CompositeRules`.

`Wrapper` defines a framework for writing wrappers. Users create specializations of `Wrapper` by overriding the `defineRules()` method and using a the `addRule()` method to add rules to the `Wrapper`. The `label()` method of `Wrapper` uses its input as input to the rules in the `Wrapper`. This method returns the trees of

```
<?xml version="1.0"?>
<countries>

...

<country>

<name>
United States
</name>
<coordinates>

<longitude>
97 00 W
</longitude>
<latitude>
38 00 N
</latitude>
</coordinates>
<capital>

Washington, DC

</capital>
</country>
<country>

<name>
Uruguay
</name>
<coordinates>

<longitude>
56 00 W
</longitude>
<latitude>
33 00 S
</latitude>
</coordinates>
<capital>

Montevideo

</capital>
</country>

...

</countries>
```

Figure 6: Section of output from WorldFactbookExample

`Attributes` returned when the `Rules` are evaluated in a `Label` object. A `Label` can be printed by using a `LabelPrinter` and its `Attributes` can be accessed directly using the `Attribute.AttributeIterator`. The `LabelPrinter` provides a way for the user to specify the format used to output the data. The default format is an XML-like document, but it is easy to modify `LabelPrinter` to produce any kind of structured document. `Attribute.AttributeIterator` allows the user to access the values of an `Attribute` tree directly in a Java program, which provides an efficient way to integrate multiple extraction tasks.

In `WorldFactbookExample` a specialization of wrapper is defined (lines 10–43) using an inner class. In the `defineRules()` method the rules for the wrapper are defined as described above, and the `addRule()` method is used to add the rule tree to the wrapper (line 41). A `Label` is created in by the call in line 44.

This `Label` is printed to a file given as a command line argument using a `LabelPrinter` (line 46). Figure 6 contains a section of the output printed by the `LabelPrinter`.

Evaluation

The number of non-comment source statements (NCSS) is used to determine how much work a developer has to do in order to implement a wrapper using a given set of tools. The lower the NCSS, the more work the tools are doing for the user. NCSS provides a way to compare ELIXIR both to an earlier version of itself, and to SPHINX, another Java library with related goals.

Amazon.com

Data from Amazon.com is used in the LIBRA book recommending system (Mooney and Roy, 1999). The spider used in LIBRA is now out of date due to changes in the format of Amazon.com's pages, so a new one was needed. Figure 7 shows the rule tree for the data that is needed from Amazon.com. ELIXIR itself grew largely out of lessons learned while developing a spider for Amazon.com, and an earlier version of the spider is compared to the current spider.

To see the benefits that ELIXIR gives a user, the Amazon.com spider (`AmazonSpider`) written with ELIXIR to is compared to another spider written using a prototype of ELIXIR (`OldAmazonSpider`). Most of the facilities of the `textutils` package (which contains the `ItemExtractor`s) are available in the prototype, and it makes use of `Slot`s, which are an early version of `Attribute`s and `Rule`s. The most notable difference is that `Slot`s cannot be nested. In addition, `Slot`s contain the rules for extraction and the values output by the rules, so a `Slot` is kind of a combination of a `Rule` and an `Attribute`. Slots do not support any kind of conditional branching, and there is no way to download pages from the Internet using an `ItemExtractor` in the prototype. Note that the format of Amazon.com has changed between the time that `AmazonSpider` and `OldAmazonSpider` were written, but the information extracted and the difficulty of specifying how to find that information using the facilities of the `textutils` package are similar.

Amazon.com presents an interesting challenge for the library. In response to a search, a page is returned with the URLs of the first twenty-five books. On that page is a pointer to a similar page containing the next twenty-five books, and so on. These search results pages (see figure 8) appear to be generated dynamically on a per-session basis. After a long enough time, Amazon.com will stop providing URLs to more search results pages. For very large crawls a problem arises because the library matches the entire rule, causing it to download and process each book information page on a search results page before moving on to the next search results page. In practice, this has caused Amazon.com to stop supplying new search results pages after a few thousand books have been processed. This problem is solved by writing two wrappers and taking advantage of the fact that the wrappers are Java programs. One wrapper extracts all of the URLs for books from a page with a list of books from a search, and the wrapper also extracts the URL of the page containing the next results from that search. This wrapper spends less time on each search results page, and is able to retrieve the URLs for many more books. A short Java routine is used to apply this wrapper iteratively starting on the first page, then each successive page. The URLs from each page are stored in a list. The second wrapper extracts book information from a book page and is applied to each of the URLs in the list.

`AmazonSpider` is contained in the file AmazonSpider.java. This file contains a wrapper that extracts the URLs for book information pages from a search results page (`AmazonBookURLWrapper`), a wrapper that extracts book information given a URL to a book information page (`AmazonBookWrapper`), and code to combine the two wrappers and print the output. The code for `OldAmazonSpider` is spread among three files. OldAmazonBookPage.java defines a class (`OldAmazonBookPage`) that extracts information from a book information page. The URLs for book information pages are extracted by `OldAmazonSearchResultsPage`, defined in OldAmazonSearchResultsPage.java. OldAmazonSpider.java contains methods for running `OldAmazonSearchResultsPage` and `OldAmazonBookPage`, and a way to print out the extracted information.

Table 1: NCSS values for `AmazonSpider` and `OldAmazonSpider` wrappers

Spider	Search results page	Book information page	Combining wrappers
AmazonSpider	8	34	23
OldAmazonSpider	40	364	200

The comparison of NCSS values for `AmazonSpider` and `OldAmazonSpider` (table 2) illustrates the amount of work that the library saves the user. Although `AmazonSpider` and `OldAmazonSpider` extract approximately the same number of fields, `AmazonSpider` is significantly shorter. The processing of the search results page is very straightforward, and has the lowest NCSS values in both systems. Most of the additional complexity in this portion of `OldAmazonSpider` comes from logic to determine if there are more URLs for book information pages available, or if the client should be told to use the next search results page. ELIXIR encapsulates this complexity. For the new `AmazonSpider`, `AmazonBookURLWrapper` contains only rules to extract the URLs for book information pages and the URL for the next search results page. Extracting book information is the most complicated task, and thus its NCSS value is the highest in both systems. Since `Slots` cannot be nested, `OldAmazonSpider` needs logic to use the value of one `Slot` as input to another `Slot`. In ELIXIR, these values are passed along automatically. The NCSS value for the logic for combining wrappers in `OldAmazonSpider` is particularly high because of the fact that functions had to be written to print out the final extracted values. `AmazonSpider` uses `LabelPrinter` to handle the printing. In addition, `OldAmazonSpider` was originally designed to be multi-threaded, while ELIXIR is not. Thus the NCSS values for combining the wrappers should be closer together. However, the smaller NCSS values that ELIXIR needs to process the search results pages and the book information pages demonstrate the amount of work that the library can save the user.

ResearchIndex

SPHINX (Miller and Bharat, 1998) is a Java toolkit and development environment for writing Web crawlers. Noppadon Kamolvilassatian used SPHINX to create a spider that extracted name and homepage URLs from the second page of the ResearchIndex most cited author list, available at http://citeseer.nj.nec.com/cited2.html (figure 9). SPHINX is more suited to writing crawlers that focus on retrieving entire files, for example all of the images found on a page, rather than extracting specific substrings from languages. Since SPHINX crawlers are written in Java, however, there is an opportunity to compare the amount of work it takes to write a wrapper for the second page of the ResearchIndex most cited author list using SPHINX, and a wrapper for the same page using ELIXIR.

Table 2: NCSS of wrappers for the second page of the ResearchIndex most cited author list

Library	NCSS for wrapper
SPHINX	158
ELIXIR	15

Although neither wrapper requires a lot of work, ELIXIR was designed for this kind of extraction task. The results are in table 2. Most of the extra work needed to write a wrapper using SPHINX involves writing the operations needed to combine and present the extracted data, since the wrapper extracts the names and URLs separately and they are combined later. ELIXIR allows the user to describe more of the structure of the document, so the wrapper written using ELIXIR automatically associates the names and URLs in its XML-like output, which is generated automatically by a `LabelPrinter`. However, if a different output was desired, the wrapper written using ELIXIR would need to specify a way to generate that output, such as a new specialization of `LabelPrinter`. This task of generating a different output would add to the amount of work required to write a wrapper using ELIXIR.

Related Work

ELIXIR provides a way to combine text extraction and spidering in wrappers. Wrappers are written in Java and are thus easy to integrate into Java programs, and wrappers can also be used to output a structured text document. By implementing new `ItemExtractors`, the user can easily extend the functionality of the ELIXIR. The model for wrappers is similar to that presented in (Hammer et al., 1997), in which extracted values may be used as input for later extractions. ELIXIR also adopts the cascade design from (Sahuguet and Azavant, 1998), in which extracted values are passed as input to the next extractors in the hierarchy.

The text extraction tools presented in (Hammer et al., 1997) are intended to retrieve data in Object Exchange Model format of the TSIMMIS system. Extractors are written in a declarative specification language. The extractor specification is used to parse a web page or set of web pages, and the results are returned in a format that can be queried using tools from the TSIMMIS project. Users specify rules to extract text into variables. Once a variable has been extracted, its value may be used as the input to the extraction rules for another value. However, the user must explicitly specify that a variable should take another value as its input. Thus extractors may use the values of previously extracted fields, including following extracted URLs, but must do so explicitly.

In the WysiWyg Web Wrapper Factory (Sahuguet and Azavant, 1998), wrapper specifications are written in a declarative language with the assistance of a GUI interface. The values to extract can be specified using a combination of the HTML tree and regular expressions. Extracted values are passed to embedded extractors in a cascade fashion. This cascading is the only way that previously extracted values can be used elsewhere in the extractor. The output of the wrapper is a Java object to be used in a higher level application that controls the wrapper. There is no way to follow URLs extracted by a wrapper except through a higher level application that uses one wrapper to extract the URLs and another for the pages the URLs point to. ELIXIR does not provide support for navigating a document as a tree, although support could be added by providing an appropriate set of `ItemExtractors`. Also, `RuleBuilder` provides a natural way for a GUI builder to construct wrappers on the fly.

WebL (Kistler and Marais, 1998) is a general purpose programming language for web document processing. Operators for retrieving web pages and performing extraction on HTML documents are provided by WebL. WebL is implemented in Java, and it is possible to use the functionality of WebL directly from Java programs. Writing wrappers in WebL involves writing a program that explicitly performs the extraction by manipulating the document rather than writing a specification in a declarative language.

The SPHINX toolkit (Miller and Bharat, 1998) provides a framework and interactive development environment for site-specific crawlers. SPHINX provides many ways to specify which pages to download and which URLs should be followed on each page. Part of the framework supports text extraction on pages, but SPHINX does not provide any libraries that assist extraction. One of the experiments discussed earlier shows that using ELIXIR instead of SPHINX to write a wrapper resulted in a 90% reduction in the number of non-comment source statements.

Some people have looked at ways to automatically generate wrappers using machine learning techniques. A survey of systems that learn a wrapper from a set of examples is provided in (Kushmerick et al., 1997) and also (Muslea, 1999). ELIXIR does not provide any support for learning from examples, although it would not be difficult for learning programs to either create wrappers through a `RuleBuilder` or by directly writing a Java file.

Future Work

The amount of code that has to be written to combine the two wrappers for the `AmazonSpider` suggests a need for a method to control the way ELIXIR processes data, and an improved model for iteration and conditional matching of rules. Ideally, there would only be one wrapper in `AmazonSpider`, and that wrapper would have approximately the combined complexity of the current `AmazonBookWrapper` and `AmazonBookURLWrapper`. However, more cases should be considered so that the solutions to these problems are generally useful.

A richer set of `ItemExtractor`s would enhance the convenience and flexibility of ELIXIR. HTML aware `ItemExtractor`s could be written to get attributes from a specific tag, or to write extractors that make use of the DOM trees of documents. Also, the `ItemExtractor` interface itself could be generalized to work with arbitrary objects. This generalization might be useful for combining text and non-text extraction and retrieval. A subclass called `StringItemExtractor` could take the place of the current `ItemExtractor`.

Improvements could also be made in ways to assist the user in writing wrappers. One way would be to develop a special purpose specification language for wrappers. Users could then write their wrappers in a text file that could be compiled into a Java source file, or interpreted. Another improvement would be a graphical user interface to help the user design and test wrappers. Such a system could generate a Java source file containing the wrapper definition, build the wrapper directly using a `RuleBuilder`, or both. A third improvement would be applying machine learning techniques to automatically generate wrappers for a given site.

Conclusion

ELIXIR is a library for writing wrappers in Java. ELIXIR provides a way to combine text extraction and spidering in wrappers. Since wrappers written using ELIXIR are Java programs, they are easy to integrate with other Java programs. The user can also extend the functionality of ELIXIR by implementing new `ItemExtractor`s. In an experiment, a wrapper written using ELIXIR showed an 89% reduction in non-comment source statements from a wrapper written using a prototype of ELIXIR. In another experiment, a wrapper written using ELIXIR showed a 90% reduction in non-comment source statements from a wrapper written using SPHINX (Miller and Bharat, 1998), a Java toolkit for writing spiders.

Acknowledgements

Noppadon Kamolvilassatian provided the wrapper for ResearchIndex that uses SPHINX. Thanks are also due to Dr. Ray Mooney, who supervised this work, and Dr. Inderjit Dhillon for agreeing to be the second reader.

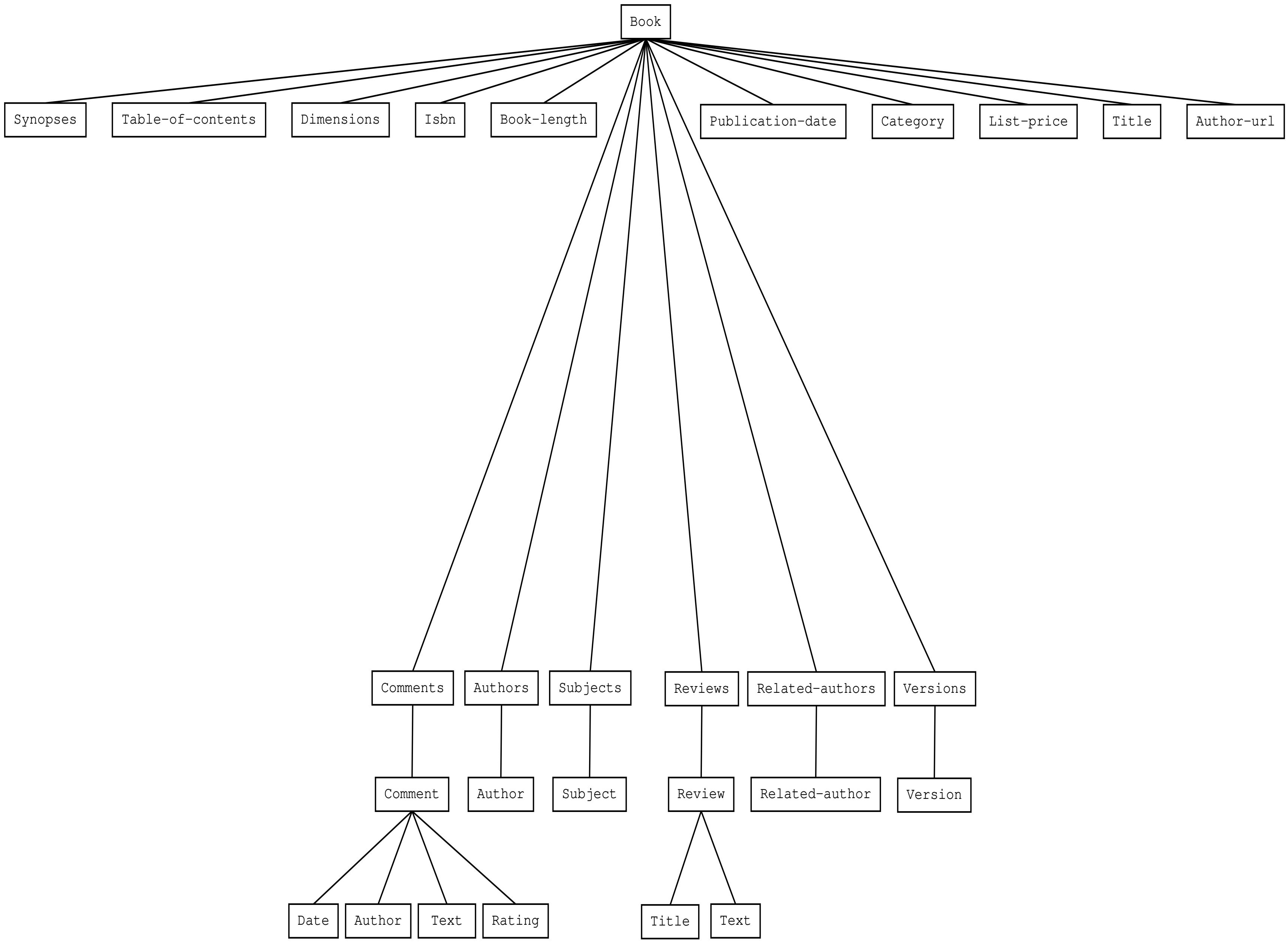
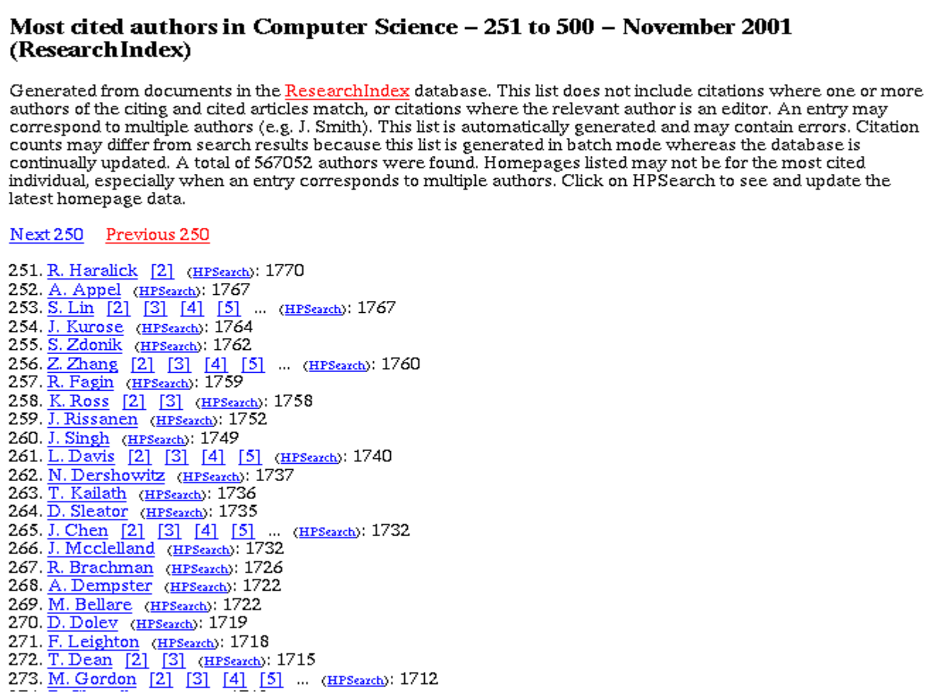


Figure 7: Rule tree for AmazonSpider

Figure 8: Search results page for "java" on Amazon.com



Most cited authors in Computer Science – 251 to 500 – November 2001 (ResearchIndex)

Generated from documents in the ResearchIndex database. This list does not include citations where one or more authors of the citing and cited articles match, or citations where the relevant author is an editor. An entry may correspond to multiple authors (e.g. J. Smith). This list is automatically generated and may contain errors. Citation counts may differ from search results because this list is generated in batch mode whereas the database is continually updated. A total of 567052 authors were found. Homepages listed may not be for the most cited individual, especially when an entry corresponds to multiple authors. Click on HPSearch to see and update the latest homepage data.

Next 250 Previous 250

251. R. Haralick [2] (HPSearch): 1770
252. A. Appel (HPSearch): 1767
253. S. Lin [2] [3] [4] [5] ... (HPSearch): 1767
254. J. Kurose (HPSearch): 1764
255. S. Zdonik (HPSearch): 1762
256. Z. Zhang [2] [3] [4] [5] ... (HPSearch): 1760
257. R. Fagin (HPSearch): 1759
258. K. Ross [2] [3] (HPSearch): 1758
259. J. Rissanen (HPSearch): 1752
260. J. Singh (HPSearch): 1749
261. L. Davis [2] [3] [4] [5] (HPSearch): 1740
262. N. Dershowitz (HPSearch): 1737
263. T. Kailath (HPSearch): 1736
264. D. Sleator (HPSearch): 1735
265. J. Chen [2] [3] [4] [5] ... (HPSearch): 1732
266. J. Mcclelland (HPSearch): 1732
267. R. Brachman (HPSearch): 1726
268. A. Dempster (HPSearch): 1722
269. M. Bellare (HPSearch): 1722
270. D. Dolev (HPSearch): 1719
271. F. Leighton (HPSearch): 1718
272. T. Dean [2] [3] (HPSearch): 1715
273. M. Gordon [2] [3] [4] [5] ... (HPSearch): 1712

Figure 9: Second page of the ResearchIndex most cited author list

Bibliography

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

Hammer, J., Garcia-Molina, H., Cho, J., Crespo, A., and Aranha, R. (1997). Extracting semistructured information from the web. In *Proceedings of the Workshop on Management for Semistructured Data*.

Kistler, T. and Marais, H. (1998). Webl: a programming language for the web. In *WWW7*, Brisbane, Australia.

Kushmerick, N., Weld, D. S., and Doorenbos, R. B. (1997). Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737.

Miller, R. C. and Bharat, K. (1998). Sphinx: A framework for creating personal, site-specific web crawlers. In *WWW7*, Brisbane, Australia.

Mooney, R. J. and Roy, L. (1999). Content-based book recommending using learning for text categorization. In *Proceedings of the SIGIR-99 Workshop on Recommender Systems: Algorithms and Evaluation*, Berkeley, CA.

Muslea, I. (1999). Extraction patterns for information extraction tasks: a survey. In *AAAI'99 Workshop on Machine Learning for Information Extraction*. AAAI Press.

Sahuguet, A. and Azavant, F. (1998). W4f: a wysiwyg web wrapper factory. Technical report, University of Pennsylvania.

Appendix A: AmazonSpider source listing


```
package amazon;
/*
 * AmazonSpider.java
 * crawls amazon.com to get information about books for the LIBRA system.
 * May 23, 2001
 * Ted Wild
 */

import wrapper.*;
import textutils.*;
import java.util.*;
import java.io.*;

/**
 * <code>AmazonBookWrapper</code> retrieves information starting at a
 * page containing book information from amazon. com.
 *
 * @author Ted Wild */
class AmazonBookWrapper extends Wrapper {
    public AmazonBookWrapper() {
        super();
    }

    protected void defineRules() {
        RuleBuilder rb = new RuleBuilder("book",
                                         new ItemExtractor[] {
                                             new LinkFollower("http://www.amazon.com/exec/obidos/ASIN/")});
        rb.addRuleToRule("book", "title",
                         new ItemExtractor[] {new ForwardItemExtractor("<b class=\"sans\">", "</b>")});
        rb.addRuleToRule("book", "authors",
                         new ItemExtractor[] {new ForwardItemExtractor(">\n\nby", "<br>"),
                                              new ForwardItemExtractor(">", "</a>")},
                         Cardinality.MANY);
        rb.addRuleToRule("book", "list-price",
                         new ItemExtractor[] {new ForwardItemExtractor("<b>List Price</b>: <strike>", "</strike>")});
        rb.addRuleToRule("book", "category",
                         new ItemExtractor[] {new ForwardItemExtractor("Category:", "</b></font>"),
                                              new ForwardItemExtractor(">", "<")});
        rb.addRuleToRule("book", "publication-date",
                         new ItemExtractor[] {new ForwardItemExtractor("pages", "<br>"),
                                              new ForwardItemExtractor("(", ")")});
        rb.addRuleToRule("book", "book-length",
                         new ItemExtractor[] {new ForwardItemExtractor(null, "Editorial Reviews"),
                                              new ReverseItemExtractor("-", "pages")});
        rb.addRuleToRule("book", "isbn",
                         new ItemExtractor[] {new ForwardItemExtractor("ISBN:", "\n")});
        rb.addRuleToRule("book", "dimensions",
                         new ItemExtractor[] {new ForwardItemExtractor("; Dimensions (in inches):", "\n<br>")});
        rb.addRuleToRule("book", "table-of-contents",
                         new ItemExtractor[] {
```

```
                        new ForwardItemExtractor(null, "editorial reviews"),
                        new ReverseItemExtractor("<a href=\"", "\">table of contents</a>"),
                        new LinkFollower("http://www.amazon.com"),
                        new ForwardItemExtractor("<b class=\"h1\">Table of Contents</b>", "<hr noshade size=1>")});
rb.addRuleToRule("book", "synopses",
                    new ItemExtractor[] {
                        new ForwardItemExtractor(
                            "<b><font face=verdana,arial,helvetica color=#CC6600>Table of Contents</font></b>",
                            "<hr noshade size=1>"),
                        new ForwardItemExtractor("<p>\n\n\n\n", "\n\n")});
rb.addRuleToRule("book", "author-url",
                    new ItemExtractor[] {new ForwardItemExtractor(" all books by <a href=\"", "\">")});
rb.addRuleToRule("book", "type",
                    new ItemExtractor[] {new ReverseItemExtractor("<br", "pages"),
                                         new ForwardItemExtractor("<b>", "</b>")});
rb.addRuleToRule("book", "subjects",
                    new ItemExtractor[] {new ForwardItemExtractor("<b>Search for books by subject:</b>",
                                                                  "<input type=submit")});
rb.addRuleToRule(new String[] {"book", "subjects"},
                    "subject",
                    new ItemExtractor[] {new ForwardItemExtractor("value=\"", "\">")},
                    Cardinality.MANY);
rb.addRuleToRule("book", "versions",
                    new ItemExtractor[] {new ForwardItemExtractor("<b>Other Editions: </b>",
                                                                  "<b>Amazon.com Sales Rank: </b>")});
rb.addRuleToRule(new String[] {"book", "versions"},
                    "version",
                    new ItemExtractor[] {new ForwardItemExtractor("ASIN/", "/")},
                    Cardinality.MANY);
rb.addRuleToRule("book", "related-authors",
                    new ItemExtractor[] {new ForwardItemExtractor(">Customers who bought titles by",
                                                                  "<hr noshade size=1>"),
                                         new ForwardItemExtractor("<ul>", "</ul>")});
rb.addRuleToRule(new String[] {"book", "related-authors"},
                    "related-author",
                    new ItemExtractor[] {new ForwardItemExtractor("<li>", "a>"),
                                         new ForwardItemExtractor(">", "<")},
                    Cardinality.MANY);
rb.addRuleToRule("book", "reviews",
                    new ItemExtractor[] {
                        new ReverseItemExtractor("<a href=", ">editorial reviews</a></font></td></tr>"),
                        new LinkFollower("http://www.amazon.com"),
                        new ForwardItemExtractor("<b class=\"h1\">Editorial Reviews</b>", "<hr noshade size=1>")});
rb.addRuleToRule(new String[] {"book", "reviews"},
                    "review",
                    new ItemExtractor[] {new ForwardItemExtractor("<a name=", "</a><P>")},
                    Cardinality.MANY);
rb.addRuleToRule(new String[] {"book", "reviews", "review"},
                    "title",
                    new ItemExtractor[] {
```

```
                              new ForwardItemExtractor("<font face=verdana,arial,helvetica size=-1><b><i>", "</i></b>")});
        rb.addRuleToRule(new String[] {"book", "reviews", "review"},
                          "text",
                          new ItemExtractor[] {new ForwardItemExtractor("</font>", null)});
        rb.addRuleToRule("book", "comments",
                          new ItemExtractor[] {
                              new ReverseItemExtractor("<a href=", ">customer reviews</a></font></td></tr>"),
                              new LinkFollower("http://www.amazon.com")});
        rb.addRuleToRule(new String[] {"book", "comments"},
                          "comment",
                          new ItemExtractor[] {
                              new ForwardItemExtractor("<p>\n<font face=verdana,arial,helvetica size=-1>",
                                                       "<form method=\"POST\"")},
                          Cardinality.MANY);
        rb.addRuleToRule(new String[] {"book", "comments", "comment"},
                          "author",
                          new ItemExtractor[] {new ForwardItemExtractor("Reviewer:\n<b>\n\n\n\n\n\n\n\n\n", "\n\n\n\n</b>")});
        rb.addRuleToRule(new String[] {"book", "comments", "comment"},
                          "date",
                          new ItemExtractor[] {new ForwardItemExtractor("</b>, ", "\n\n<br clear=left>")});
        rb.addRuleToRule(new String[] {"book", "comments", "comment"},
                          "rating",
                          new ItemExtractor[] {new ForwardItemExtractor("alt=\"", "\">")});
        rb.addRuleToRule(new String[] {"book", "comments", "comment"},
                          "text",
                          new ItemExtractor[] {new ForwardItemExtractor("</font><br>\n", "<br clear=left>")});
        addRule(rb.rule());
    }
}

class AmazonBookURLWrapper extends Wrapper {
    public AmazonBookURLWrapper() {
        super();
    }

    protected void defineRules() {
        RuleBuilder rb = new RuleBuilder("list-page",
                                         new ItemExtractor[] {
                                             new LinkFollower()});
        rb.addRuleToRule("list-page", "next",
                          new ItemExtractor[] {
                              new ReverseItemExtractor("<a ", "alt=\"More Results\"></a>"),
                              new ForwardItemExtractor("href=", ">")});
        rb.addRuleToRule("list-page", "url",
                          new ItemExtractor[] {new ForwardItemExtractor("<a href=\"/exec/obidos/ASIN/", "\">")},
                          Cardinality.MANY);
        addRule(rb.rule());
    }
}
```

```
public class AmazonSpider {

    public static void main(String args[]) {
        Wrapper w = new AmazonBookURLWrapper();
        Label l = w.label(args[0]);
        List urls = new LinkedList();
        int numberOfBooks = Integer.parseInt(args[2]);

        for (int i = 0; i < numberOfBooks; ++i) {
            Attribute.AttributeIterator ai = l.getAttribute(0).iterator();
            ai.child("url");

            while(ai.hasNext())
                urls.add(ai.next().getValue());
            ai = l.getAttribute(0).iterator();
            ai.child("next");
            l = w.label("http://www.amazon.com" + ai.next().getValue());
        }
        System.out.println("Got urls: " + urls);
        w = new AmazonBookWrapper();
        LabelPrinter lp = new LabelPrinter();
        try {
            Writer fw = new FileWriter(args[1]);

            for (Iterator i = urls.iterator(); i.hasNext(); )
                lp.print(w.label((String) i.next()), fw);
            fw.close();
        }
        catch (IOException e) {
            System.err.println("Problem writing to " + args[1] + ": " + e);
        }
    }
}
```

Appendix B: Wrapper for ResearchIndex source listing

```java
package examples;

import wrapper.*;
import textutils.*;
import java.io.*;

public class ResearchIndexExample extends Wrapper {

    protected void defineRules() {
        RuleBuilder rb = new RuleBuilder("ResearchIndexList",
                                        new ItemExtractor[] {
                                            new LinkFollower(),
                                            new ForwardItemExtractor("Previous 250", "Next 250")});
        rb.addRuleToRule("ResearchIndexList", "CitedAuthorEntries",
                          new ItemExtractor[] {
                              new ForwardItemExtractor(".", "<font size=-1>")},
                          Cardinality.MANY);
        rb.addRuleToRule("CitedAuthorEntries", "CitedAuthorEntry",
                          new ItemExtractor[] {
                              new ForwardItemExtractor("onmouseover=", "</a>")},
                          Cardinality.MANY);
        rb.addRuleToRule("CitedAuthorEntry", "name",
                          new ItemExtractor[] {
                              new ForwardItemExtractor(">", null)});
        rb.addRuleToRule("CitedAuthorEntry", "url",
                          new ItemExtractor[] {
                              new ForwardItemExtractor("\"self.status=\'", "\';")});
        addRule(rb.rule());
    }

    public static void main(String[] args) {
        Label l = new ResearchIndexExample().label("http://citeseer.nj.nec.com/cited2.html");

        new LabelPrinter().print(l, new PrintWriter(System.out));
    }
}
```