Data Wrapping on the World Wide Web

by

Jessica F. Qu

Submitted to the Department of Electrical Engineering
and Computer Science in Partial Fulfillment of the
Requirements for the Degree of

Master of Engineering in Electrical Engineering and
Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1996

Author .....
Department of Electrical Engineering and Computer Science
nuary 19, 1996

Certified by .....
Michael D. Siegel
ment
visor

Accepte .....
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Data Wrapping on the World Wide Web

by

Jessica F. Qu

Submitted to the
Department of Electrical Engineering and Computer Science

January 19, 1996

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented the Generic Screen Scraper. The Generic Screen Scraper is a tool that generates data wrappers to extract requested data from data sources on the World-Wide Web. Data wrappers isolates users from interacting directly with heterogeneous data sources (i.e. SQL or non-SQL) by allowing all queries to be issued using Standard Query Language based on the relational data model. Structured or semi-structured data sources on the World-Wide can be made scrapable by the Generic Screen Scraper, as long as the data sources are registred following some specifications.

Thesis Supervisor: Michael D. Siegel
Title: Principal Research Scientist, Sloan School of Management

Acknowledgments

I wish to thank my thesis advisor, Michael Siegel, and Professor Stuart Madnick at the MIT Sloan School for giving me the opportunity to work in their group, and for their guidance and advice.

I would like to thank everyone in the Information Technology group at the MIT Sloan School of Management for providing a positive working atmosphere. In particular, I am indebted to Cheng Goh for his guidance and support throughout the project.

My friends, particularly Jenny Lee, Ann Chen, Alexandra Pau, Theodore Tonchev, Marilyn Chen, Vivian Tung, Melissa Lee, and Gloria Tsuen, have encouraged me to stay focused and motivated in my work. I thank them for their friendship, prayers, and support.

Last but not least, I wish to thank my parents and Sung for their love, encouragement, understanding, and support in every possible way. To them I dedicate this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Large organizations often exchange information developed on different systems, where sources and receivers have implicit preconceived assumptions about the meaning of data. It is thus not uncommon for system A and system B to use different terms to define the same thing. However, in order to achieve a useful exchange of data, the individual systems must agree on the meanings of the exchanged data. In other words, the organizations must ensure contextual (or semantic) interoperability.

The Context Interchange Network (COIN) is designed to provide for intelligent integration of contextually heterogeneous data. It provides for (1) access to the information the way user want it; (2) explanation of the context associated with a given piece of data; (3) management of large data networks. [3]

At the heart of the COIN is the Context Mediator. The Context Mediator determines the necessary transformations needed to ensure meaningful exchange of data by examining the semantic conflicts between data contexts. The Context Mediator intercepts all requests made by the receiver and translates the queries made by the receiver to the source context. It poses the translated query to the data source and captures the data returned. Finally, the returned data is converted back to the receiver context and presented to the receiver. The context mediation process enables the COIN to accurately extract disparate data from different sources and receivers and produce consistent results. [3]

The COIN has been implemented using the client/server model in the PC environment. The context mediator is implemented in LISP on the server. The user interface is an application in Powerbuilder that can run on any PC client. The client and server are connected by Remote Procedure Calls.

Since the World Wide Web has rapidly become the pool for human knowledge exchange, it makes an ideal environment for further development of COIN. The WWW's phenomenal growth and abundant resources have led to the implementation of COIN in the WWW environment to provide for semantic interoperability between heterogeneous sources and receivers on the WWW. Additional components have been implemented for the web-based COIN, namely the Multidatabase Browser, the Ontology/Context Editor, and the Generic Screen Scraper. The Mutlidatabase Browser is a query builder modelled after the QBE-like interface of Microsoft Access. It builds a query and sends it to the Context Mediator. The Ontology/Context Editor is a tool for to browsing or editing the ontologies and contexts in the COIN.

The design, implementation, and application of the Generic Screen Scraper make up this thesis. The Generic Screen Scraper generates data wrappers that isolate user from interacting directly with heterogeneous data sources (i.e. SQL and non-SQL) by allowing all queries to be issued using Standard Query Language based on the relational data model [4]. A data wrapper sends the translated query from the Context Mediator to the targeted data source and returns requested data. Disclosure is an example of SQL data sources, and NETworth quote server is an example of non-SQL data sources that can be wrapped. In fact, most of semi-structured data sources on the WWW can be made scrapable by the Generic Screen Scraper. However, these data sources must be registered in the COIN following some specifications.

1.1 Thesis Overview

Chapter 2, *Background*, describes the background of the World Wide Web in detail. In addition, some relevant works that have been done in the field of data wrapping are discussed.

Chapter 3, *Design*, presents the design specification, the design of the overall architecture as well as the main components of the Generic Screen Scraper.

Chapter 4, *Implementation Details*, provides detailed descriptions of the implementation of the Generic Screen Scraper, and some of the notable components of a data wrapper generated by the Generic Screen Scraper.

Chapter 5, *Application*, discusses why the Generic Screen Scraper is a good data wrapping tool to have, and demonstrates who can use the Generic Screen Scraper and how it can be used.

Chapter 6, *Future Works*, lists some problems with the current system, and gives some recommendation for further improvements that can be implemented to make the system more robust.

Chapter 2

Background

This chapter first describes in detail the World Wide Web. The World Wide Web is where the structured and semi-structured data sources reside. Other related data wrapping works that have been done are then discussed.

2.1 The World Wide Web

The World Wide Web (the WWW) has become a pool for human knowledge, where people can communicate and learn from each other. Its user interface is independent of the text format, platform, clients, and network protocols, and has thus made it a very popular and convenient means for information exchange and sharing. [1]

The Web defines a number of things which should be distinguished. It has come to stand for the idea of a boundless information world in which all items are retrievable by a reference. Its address system, where Universal Resource Locators (URL) are strings used as network addresses of objects (i.e. homepages) on the WWW, makes this boundless information world possible despite many different protocols. The WWW's network protocol, Hypertext Transfer Protocol (HTTP), is an Internet protocol for transferring information with efficiency necessary for making hypertext jumps. The basic language of interchange for hypertext on the WWW, Hypertext Markup Language (HTML), is used for the transmission of text, menus and simple on-line help information across the net. [1]

All these distinguishable qualities have made the WWW an ideal implementation environment for the COIN. It offers not only boundless resources, but also real-time data generated dynamically by the Common Gateway Interface. The Common Gateway Inter-

face (CGI) is a standard for external gateway programs to interface with information servers and it has become the mainstay of Web interactive communication via HTML.

It is also worth pointing out that the WWW does not preserve states because HTTP is a stateless protocol. HTTP runs over a TCP connection that is held only for the duration of one operation. [2]

2.2 Related Works

There have been some interesting and relevant works developed to access data sources available on the WWW. Although they do not necessarily provide the same functions as the Generic Screen Scraper, they are related to data wrapping to some extend.

2.2.1 dbWeb

dbWeb (*http://www.aspectse.com/Product/dbWeb/dbWeb.html*) is basically a data source publisher that provides users with the ability to offer multi-platform access to data sources on the WWW or on a Local Area Network (LAN). It is a Common Gateway Interface between Open Database Connectivity (ODBC) data sources and Web server that enables easy access to information on the WWW or on the internal LAN without specialized client software.

dbWeb provides database connectivity for HTTP Web Servers running under Microsoft Windows NT. It provides real time access to ODBC data sources through Web browsers such as Netscape. dbWeb is equipped with full insert/update/delete capabilities as well as query-by-example record selection for dynamic SQL and stored procedures. Records are returned in tabular, detail or custom forms. Moreover, records can optionally contain "SmartLinks" that allow hypertext-style navigation within a data source.

The administration tools provided by dbWeb makes database publishing in the WWW possible and easy. The administrative utility helps set up a data source. A dbWeb data

source has to be created for each ODBC data source that is to be made accessible through a Web server. The administrative utility also maintains the dbWeb database repository which contains "schemas". A Schema contains all the information that dbWeb needs to generate Web pages for query-by-example, tabular, and freeform display modes. These information includes *tables*, *views*, *stored procedures*, *attributes*, *properties*, and *relationship* between objects in the database.

2.2.2 Other Works

Accessing a Database Server via the World Wide Web (*http://cscsun1.larc.nasa.gov/~beowulf/db/web-access.html*) created and maintained by Jeff Rowe is a useful document that explains everything about accessing databases on the WWW. He explains how Common Gateway Interface can be used to access databases on the WWW, the user interface types, the methods of access, and the access control of database operations.

GSQL (*http://www.santel.lu/SANTEL/SOFT/starthere.html*) is a simple Mosaic gateway to SQL databases. It parses a SQL-specification file (called a PROC file) to create a form, and then with the user-inputs, call a database backend program to process the SQL query. The PROC file maps components of the SQL string to widgets (fields, buttons, pull-down menus, etc.) for user input or selection.

dbCGI (*http://www.progress.com/webtools/dbcgi/dbcgi.htm*) is an embedded SQL toolkit for connecting databases to the World Wide Web. It is a CGI gateway which gives easy access to SQL databases. dbCGI is capable of producing formatted output, forms, tables and complex reports using SQL calls embedded in an HTML document.

DB2WWW (*http://www.software.ibm.com/data/db2/db2wfac2.html*) is IBM's WWW interface to their DB2 database product. It uses standard HTML and SQL. The HTML forms and SQL queries are stored as macro files on the Web server machine. The macro

files are processed by DB2WWW when the user requests data, and performs variable substitution to access the requested data.

Chapter 3

Design

The focus of this thesis is the Generic Screen Scraper which is basically a data wrapper generator. The Generic Screen Scraper generates the data wrapper for a data source by using the configuration information of the data source. The data wrapper then takes a SQL query, parses it, and scrapes from the data source the requested data as specified in the SQL query.

The Generic Screen Scraper is so-called because there is only one generic script written for generating data wrappers for all different data sources. The configuration file for a data source is all that needs to be created in order to have the Generic Screen Scraper scrape the data source. This chapter, after discussing the design specification, describes the design of the Generic Screen Scraper and some of its main components.

3.1 Design Specification

The design specification of the Generic Screen Scraper has been drawn according to its role in the COIN. In addition, certain design goals have been kept in mind in the design process.

3.1.1 Role in COIN

The Generic Screen Scraper is incorporated into the Context Interchange Network as a black box. The Multidatabase Browser builds a SQL query from the user input. It sends the query to the Context Mediator where the Context Mediator reformulates the query. Ideally, if the query requests for data from multiple data sources, the Context Mediator splits a multiple-data-source query into multiple single-data--source queries. Each single-data-source SQL query is eventually sent to its corresponding data wrapper generated by

the Generic Screen Scraper. The data wrapper communicates with the data sources by means of gateways. After getting either error or data back from the data sources, the wrapper tabulate the result and sends it back to the Multidatabase Browser for display.

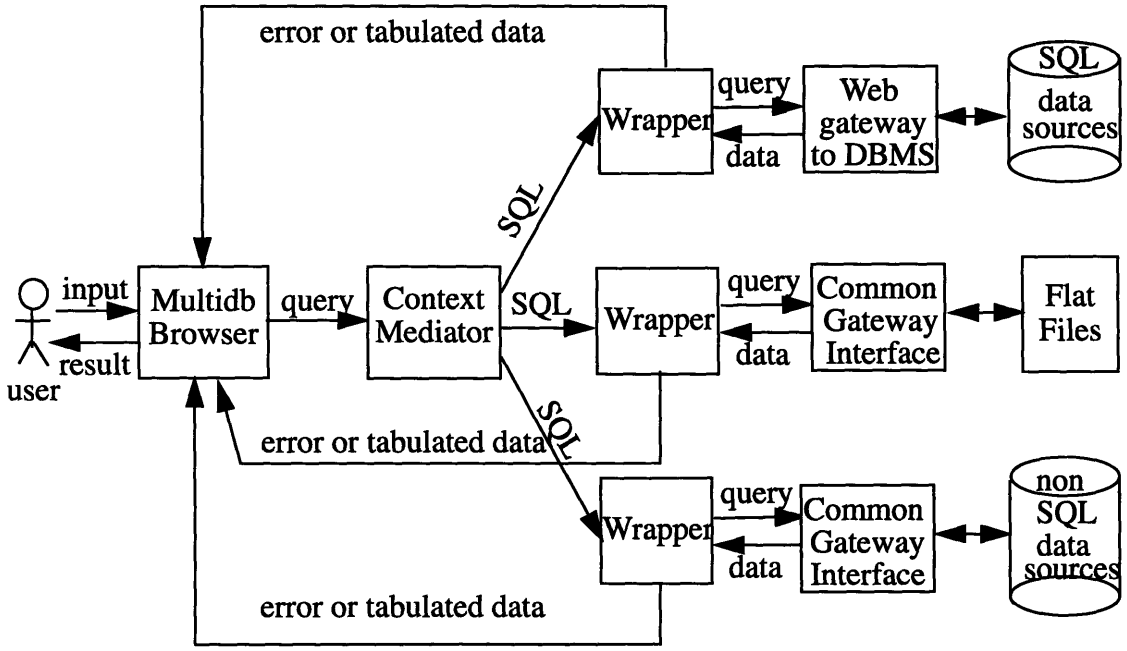

Figure 3.1: Wrapper used in COIN.

3.1.2 Design Goals

Data wrappers are expected to wrap around SQL databases, flat files, as well as Non-SQL database as illustrated in Figure 3.1. The SQL databases are easy to wrap, since any SQL query can simply be passed along to the underlying DBMS's. The SQL capabilities in this case depend on what the underlying DBMS's can handle. The Generic Screen Scraper focuses on generating data wrappers for the latter two types of data sources: flat files and non-SQL databases. These two types of data sources are also called semi-structured data sources.

NETworth Stock Quote Server (*http://quotes.galt.com/*) and 1995 FactBook (*http://www.odci.gov/cia/publications/95fact/index.html*) are examples of semi-structured data

sources. Ideally, data wrappers generated by the Generic Screen Scraper can process some sample SQL queries sent to them and return meaningful results.

Example of SQL queries that the wrappers should be able to answer are:

- **select** *
  from networth
  where networth.ticker = 'intc'
- **select** networth.high, networth.low
  from networth
  where networth.ticker in ('orcl', 'msft')
- **select** FactBook.Currency, FactBook.Capital
  from FactBook
  where FactBook.Country = 'France'
- **select** FactBook.Currency, FactBook.ExchangeRate
  from FactBook
  where FactBook.Country = 'Ch%'

3.2 Overall Architecture

The Generic Screen Scraper is designed to be as generic as possible. It is also designed to be a a stand-alone system that can be used independently for querying on different data sources.

The Generic Screen Scraper takes the name of the server (where the configuration file resides) and the name of the configuration file for a data source as inputs. It then generates a data wrapper, passing on to it the server and document information. The wrapper expects a SQL query input from the user. At the submission of a SQL query, the wrapper springs into action to look for the query result. See Figure 3.2 for the overall architecture of a data wrapper.

As the initial step, the data wrapper parses the SQL query with the SQL parser. If the parsing is successful, the wrapper checks the parsed results against the Export Schema to see if the selected tables and attributes exist for the particular data source. Assuming the query passes the Export Schema test, the wrapper issues a network protocol (using the

server and the document information from the Generic Screen Scraper) to retrieve the document that contains the configuration information for the data source being queried on.

With the information found in the configuration file, the wrapper eventually obtains the document containing requested data as specified in the query. Consequently, it scrapes from the document the requested data using regular expressions, and returns them in a tabular form.

3.3 Main Components

To further explain the design, some of the important components in Figure 3.2 are discussed in more detail in this section.

3.3.1 Export Schema

Each data source has an Export Schema that contains the data elements available for the data source. An Export Schema lists table(s) and attributes defined for the particular data source. For an example of an Export Schema, see section A.1 in Appendix A.

3.3.2 Configuration File

Configuration file is one of the vital parts of the system because without it, the Generic Screen Scraper would not be able to start executing. A configuration file has to be created for each data source. It should contain information about the capabilities and the flow control of the data source as well as the regular expressions created for the corresponding attributes defined for the data source. Section A.2 in Appendix A lists the configuration file for NETworth.

3.3.3 Regular Expressions

A regular expression is defined for every attribute available in a data source. The regular expression corresponding to the attribute *Last* for NETworth as seen in section A.2 of

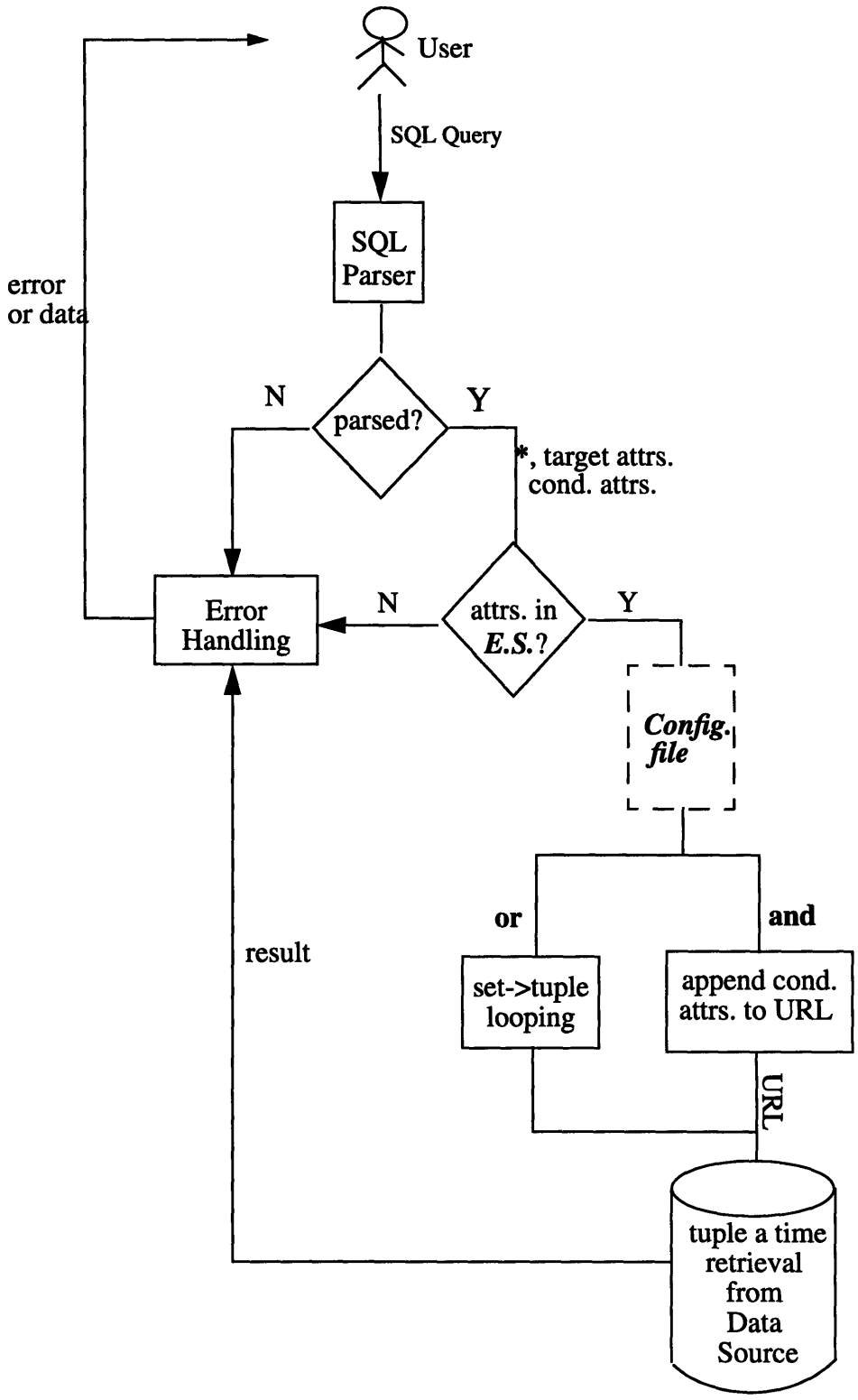

Figure 3.2: Overall Architecture of a data wrapper.

Appendix A is

Last\W/A\W\s+(.*?)\s+\WA

This regular expression is saying that the value for *Last* can be found between the strings **Last\W/A\W\s+** and **\s+\WA,** where **\W** means any non-alphanumeric such as "<", **\s** means white space, and **+** means repeated one or more times. For more information on regular expressions, consult the perl on-line manual.

3.3.4 SQL Parser

The SQL parser processes the SQL query submitted to the data wrapper. The parser takes an SQL query as the input and returns the selected table, attribute(s), and search condition if the parser successfully parses the SQL query. The parser returns the string "SQL parse failed" if it fails in parsing the SQL query. Due to the limited scope of the problem, the parser does not support all the SQL capabilities. It handles simple SELECT statement with certain capabilities listed below:

- * (wild card in the target list which implies all attributes).
- AND in the search condition.
- OR in the search condition.
- IN in the search condition.

IN is an short hand for writing multiple OR's. For example,

select networth.high, networth.low
from networth
where networth.ticker in ('orcl', 'msft')

is equivalent to

select networth.high, networth.low
from networth
where networth.ticker = 'orcl' **or** networth.ticker = 'msft'

In the case of a SQL query containing an "IN", the SQL parser automatically expands the search condition into a SQL query with multiple OR's.

3.3.5 Finite State Automata Representation of Flow Control

One major problem that the Generic Screen Scraper encountered is that not all the data sources have the same user interface, that is, most data sources have different flow control. NETworth, for instance, goes directly to the ticker query page as soon as the URL is keyed in. Typing in the name of a ticker at this point would bring up the page containing stock information. The 1995 World FactBook, which provides information on every country in the world, however, can not be handled as easily. The documents for countries are numbered in alphabetical order. In order to reach the document for a particular country, two different URL's are needed: one to get the document containing the table of contents for countries so the file name for the country can be looked up (see Figure 3.3); one to get the actual document for the country (see Figure 3.4) using the file name obtained from the previous document.

An an initial approach, this problem is solved by modifying the configuration file. If there are to be more than one screen (or URL) to bypass before reaching the document containing the requested data, there has to be a regular expression after the intermediate URL(s). The regular expression serves as the target and its found value can be appended to the next URL to get to the subsequent screen. For instance, the URL *http://www.odci.gov/cia/publications/95fact/index.html* takes the user to Figure 3.3, namely the FactBook Table of Contents. The regular expression **href="(.*?)">** is used to look up the file name for the country being queried on from Figure 3.3. In this case, the file name *ch.html* is appended to the URL *http://www.odci.gov/cia/publications/95fact/* to make the transition to Figure 3.4 possible. Data can then be scraped from Figure 3.4 which contains the currency, capital, and other information for China.

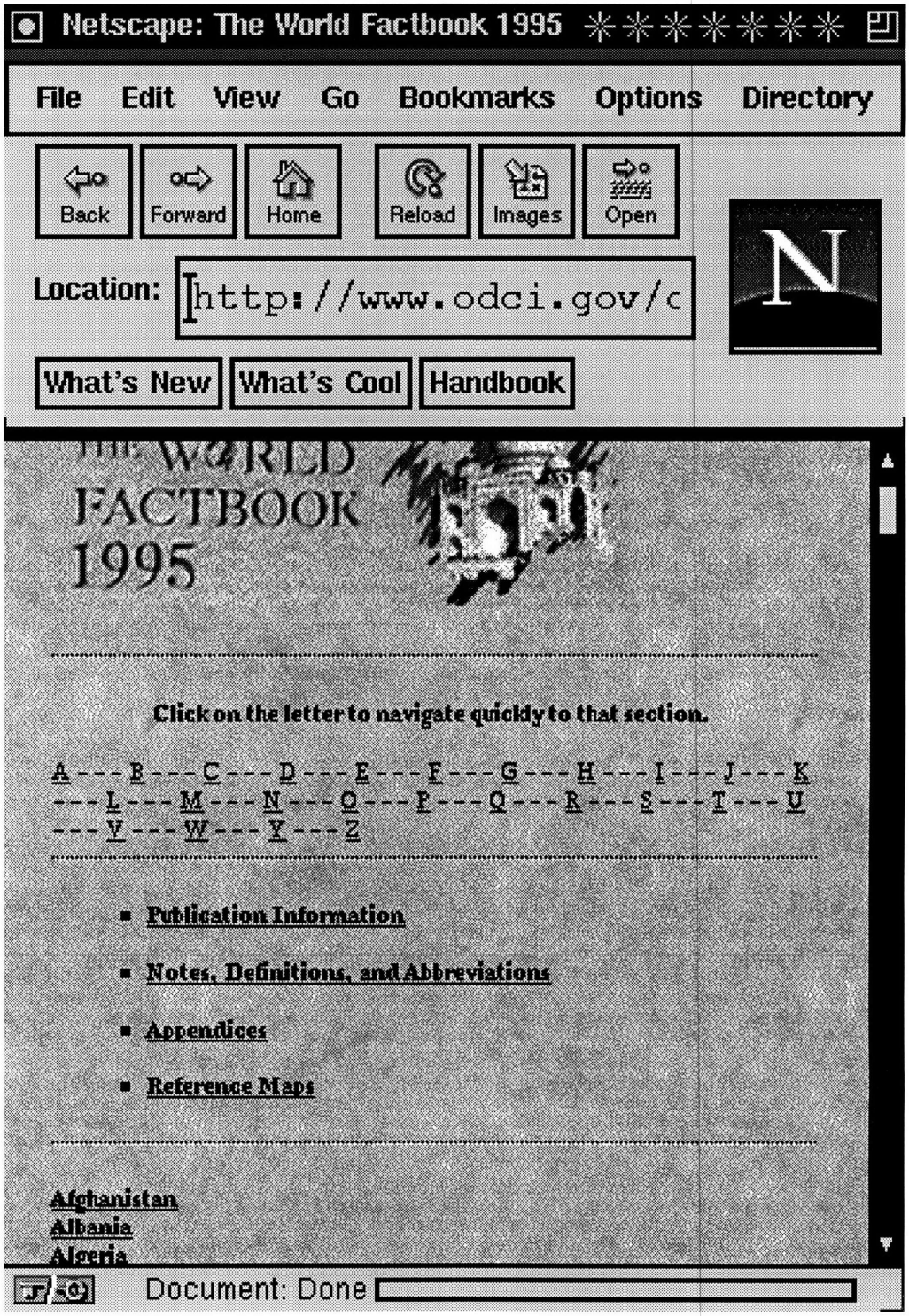

Figure 3.3: FactBook Table of Content

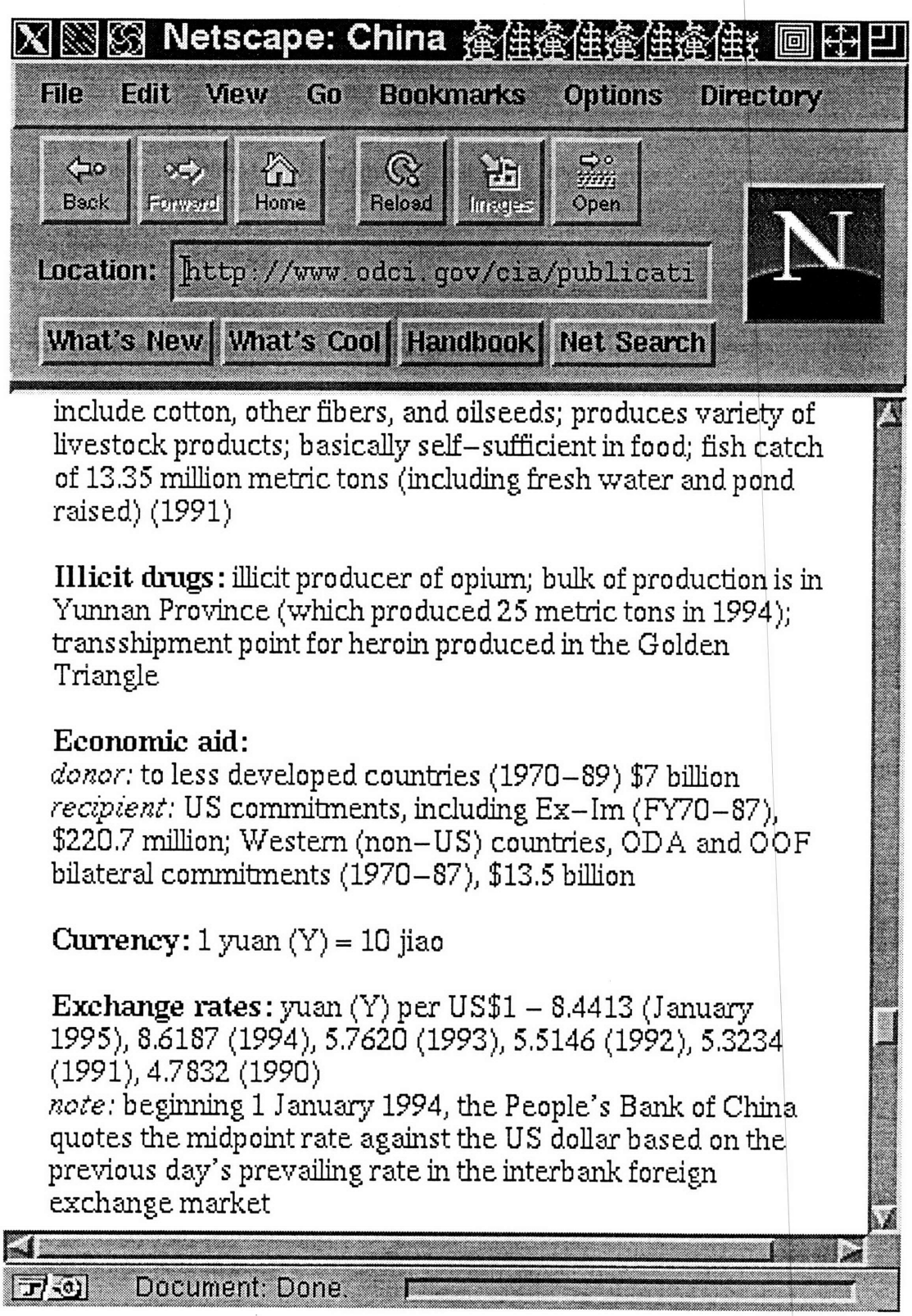

Figure 3.4: FactBook Homepage for China

The initial approach, however, evolves to be something more sophisticated. It turns out that the flow control of data sources can be represented using the finite state automata model.

A finite state automaton -- "finite" because the number of possible states and the number of input are finite, and "automaton" because the change of states is totally determined by the input -- is a collection of three things:

1. A finite set of states, one of which is designated as the **start state**, and some (or none) of which are designated as **final states**.

2. An alphabet of possible inputs that governs one state to the next.

3. A finite set of **transitions** that tell for each state and each input which state to go next. [2]

The flow control of each data source can be thought of as a finite state automaton. Each web page is a state in a transition diagram. The initial SQL query can be interpreted as traversing a path beginning at the homepage of a data source (start state). The document containing the requested data can be thought of as the final state. Each state outputs information that is needed for the next transition. The URL's (plus result from regular expression search, if any) that traverse through the one or many Web pages before reaching the final desirable page make up the alphabet of possible inputs. This alphabet of URL's governs one state to the next. The many paths that can be taken to reach the final informative page make up the finite set of transitions.

The flow controls of the NETworth Quote Server and the 1995 World FactBook are represented as finite state automata as shown below in Figure 3.5 and Figure 3.6 respectively.

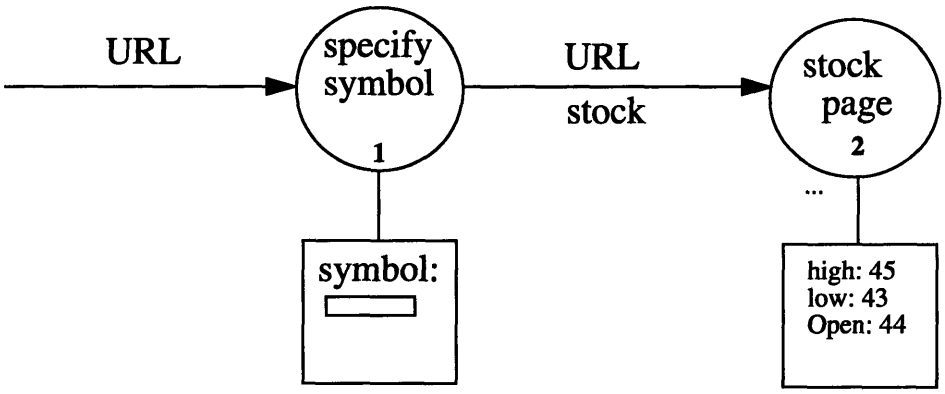

Figure 3.5: NETworth Transition Diagram.

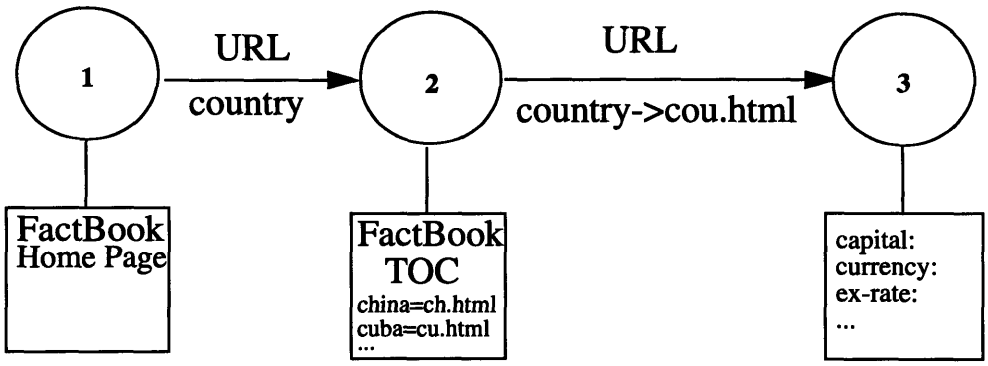

Figure 3.6: FactBook Transition diagram.

The transition diagrams can be easily translated into state transition tables. The state transition tables, in turn, are represented in the configuration file. The details are discussed in the next chapter.

3.4 Possible Errors

There are three main types of errors that can occur in this system.

- Invalid SQL Queries
- Data Source Capability Problems
- Network Errors

The first type of invalid Queries are SQL statements that are syntactically ill-formed. Any query not following the SQL standards (i.e. missing **from** or **where**) are detected by the SQL parser to be invalid. SQL queries that references attributes or tables undefined in the Export Schema for the particular data source are also invalid SQL queries. They are detected during the Export Schema check.

Queries that are beyond the capabilities of the data sources are declared invalid. Most data sources have certain limitations. For example, NETworth only allows search on *ticker*, and FactBook only allows search on *country*. This implies that both data sources can not process queries that have AND embedded. A SQL query such as

select networth.high, networth.low
from networth
where networth.ticker = 'orcl' **and** networth.symbol = 'oracle'

is not valid, since it searches on *symbol* and it contains an AND. This type of invalid queries is detected at the state transition level.

The third type of errors, network errors, are mainly problems that can possibly result from network protocols. They can also be characterized as web-related errors. Such errors are displayed to the user through the WWW Browsers. "Not found" errors resulted from

bad url's are the most frequently occurred problems. They include server not found and document not found. "Server busy" is another possible error. Most popular Web Sites with limited number of ports have this problem. Network errors are sometimes associated with access authorization. This is not really a big issue because only the data sources that allow general access to the general public should be registered.

Chapter 4

Implementation Details

This chapter discusses the specific details of the implementation of the Generic Screen Scraper, and some main components of its generated data wrapper.

4.1 Overall Implementation

The Generic Screen Scraper is implemented as a Common Gateway Interface (CGI) program written in Perl. The Common Gateway Interface is a standard for external gateway programs to interface with information servers. It has become the mainstay of World Wide Web interactive communication via HTML. Each time a client requests the URL corresponding to the Generic Screen Scraper, the server executes it in real-time. The output is displayed directly to the client. Perl is the chosen language for the Generic Screen Scraper because it is excellent for text management and data-parsing.

CGI.pm, a Perl5 library, has been a very useful tool to deal with the stateless problem of the World Wide Web. This Perl library uses objects to create Web fill-out forms on the fly and to parse their contents. It provides a simple interface for parsing and interpreting query strings passed to CGI scripts. The stateless problem of the World Wide Web is solved because the value of the previous query is used to initialize the form, so that the state of the form is preserved from invocation to invocation.

4.2 Main Components

Data wrappers generated by the Generic Screen Scraper are defined by some important components such as HTTP-GET, SQL Parser, Finite State Automata whose implementations deserve to be discussed in detail. The flow of the data wrapping process is shown in Fig. 4.1.

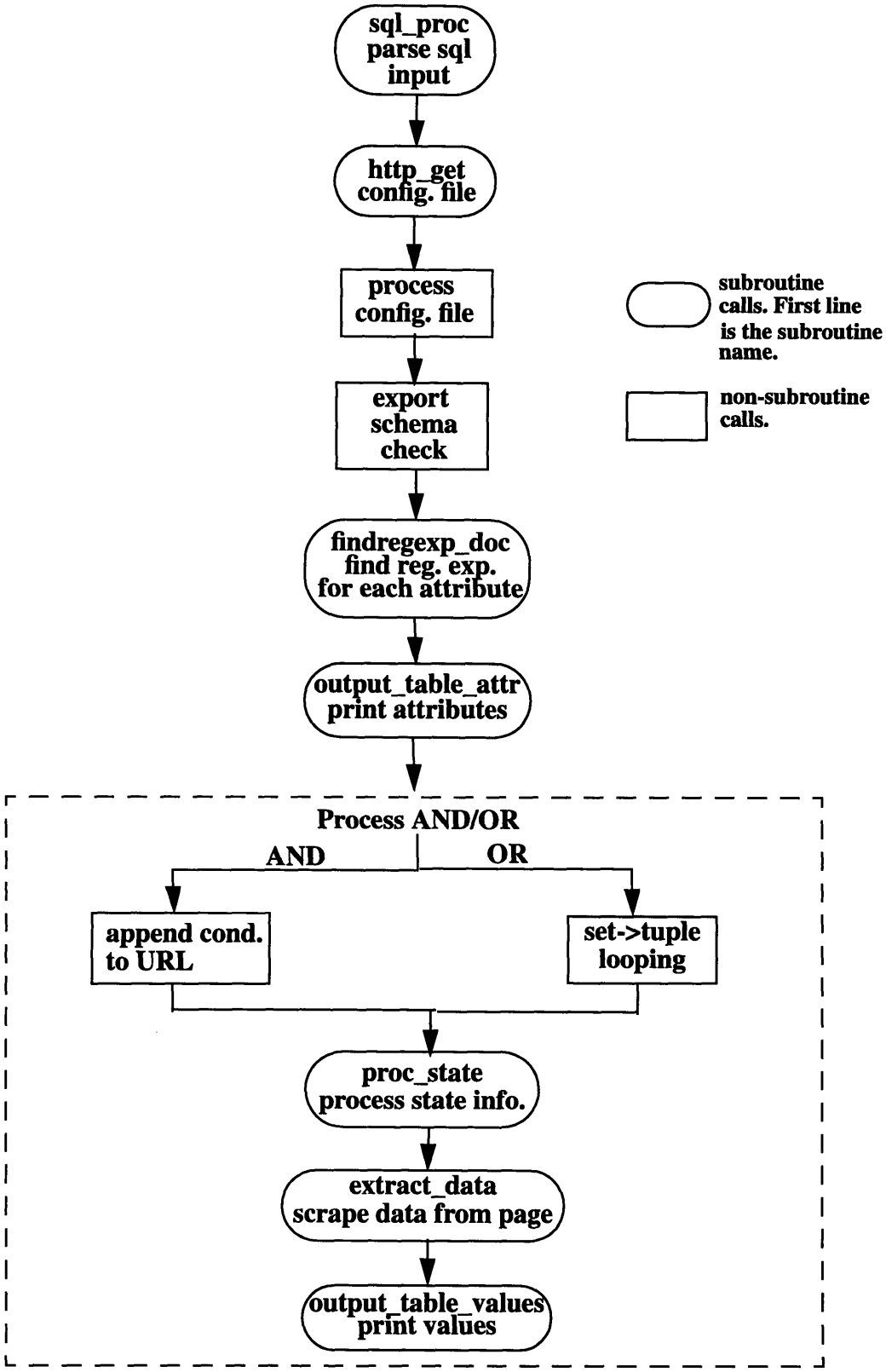

Figure 4.1: The Wrapping Process.

4.2.1 HTTP-GET

HTTP-GET is a CGI program written in Perl and copyrighted by MIT. It takes two arguments *server* and *document* (first part and second part of a URL for a Web page) and issues an HTTP-GET request across the network and returns the resulting Web page packet by packet to standard output.

The Generic Screen Scraper uses a modified HTTP-GET to grab web documents, then uses regular expression match on the documents to look for the requested data. It eventually returns the data or error in a tabular format. The original HTTP-GET was modified slightly to work for the Generic Screen Scraper. An empty text string is created initially. After each socket call, each packet is appended to the text string. The text string, instead of being sent to standard output, is returned as a long text string.

4.2.2 SQL Parser

The SQL parser is a modified version of the SQL parser provided in *Lex & Yacc*. Lex and Yacc are used to create compilers and interpreters that transform structured input. In programs with structured input, two tasks are often involved: 1) dividing the input into meaningful units, 2) discovering the relationship among the units. The division into units or tokens is known as *lexing*. Lex takes a set of descriptions of possible tokens and produces a C routine called a *lexer*. The process of establishing the relationship among the token is known as *parsing* and the set of rules that define the relationships is a *grammar*. Yacc takes a concise description of a grammar and produces a C routine that can parse that grammar called a *parser*. The Yacc parser automatically detects whether or not a sequence of input tokens matches one of the rules in the grammar. [5]

Given an SQL statement, the Lex and Yacc SQL parser merely determines whether or not the SQL parser worked parsing the SQL statement. This information, however, is not sufficient for the data wrapper. The wrapper needs to know what table is selected, what

attributes are in the target list, and what the search condition is. Therefore, the SQL parser has been modified to meet the data wrapper's needs. Many additional data structures and procedures were created in C to enable the parser to pull out what the wrapper needs to know in addition to determining whether the SQL query is parsed.

Since the SQL parser is written in C, an executable file has been created and invoked in Perl scripts through piping. The SQL query input from Perl is piped in to the executable, the parsed result is, in turn, piped out back to Perl.

4.2.3 Export Schema Check

The data wrapper takes the target list returned from the SQL parser and proceeds with the Export Schema Check. If the target list consists of "*", then the wrapper automatically translates it into all the attributes for the data source by extracting them from the Export Schema. If the target list consists of specified attribute(s), the wrapper checks the attribute(s) against the Export Schema for the data source. The goal of the Export Schema check is to make sure that the attribute(s) in the SQL query are defined in the Export Schema for the data source. Specifically, all the attributes defined in Export Schema are read into an attribute array. The Export Schema check compares the attribute(s) in the SQL query to the members of the attribute array to see if the requested attributes are defined for the data source.

4.2.4 Capabilities Check

After the Export Schema Check, the data wrapper proceeds to check the query against the capabilities the data source can handle. The common SQL capabilities that are implemented for the wrapper are AND, OR, and IN, where IN is basically equivalent to multiple OR's.

When there is no AND, OR, or IN in a query, the data wrapper simply uses the only search condition and gets a tuple of data back (assuming that there is no error). This is called a tuple-at-a-time search.

When the wrapper sees an AND in the query, it checks the capability list in the configuration file to see if AND is allowed for the particular data source. If AND is allowed, the wrapper appends the multiple search conditions to the URL, then proceeds as if doing a tuple-at-a-time search.

If OR appears in a query, the wrapper splits the query into multiple tuple-at-a-time searches each given one search condition. IN in a query is translated to multiple queries by the SQL parser as mentioned previously.

4.2.5 Finite State Automata

The Finite State Automata Representation of different data sources can be represented in corresponding state transition tables. For example, see the state transition tables for NETworth and FactBook in tables 4.1 and 4.2 respectively. The **Condition** field specifies the searchable attribute(s) that are allowed for the particular data source. It is also what is required for the transition to take place. Table 4.1 shows that *ticker* is the searchable attribute for NETworth. The **Expression** field specifies the regular expression that is required to transit to the next state when the **Condition** field is not sufficient to make the transition happen. For example, knowing the value to the attribute *country* in FactBook does not mean that we can find the homepage for the specified country. The regular expression **href="(.*?)">{}** in the **Expression** field is used to search for the file name of the specified country in the first Web page, namely, the FactBook Table of Contents as seen in Figure 3.3. The search result is appended to the URL for the transition to the homepage for the requested country. Furthermore, the **Output** field lists all the attributes

that can be found at the Web page. Every attribute has a regular expression associated with it.

From	To	URL Transition	Condition	Expression	Output
State 1	State 2	http://quotes.galt.com/stock-clnt?stock=	ticker	none	o Company \WFONT\sSIZE=\d\W(.*?),\s+\W\w+\W o Ticker ,\s\W(.*?)\W\W/FONT\W o Last_Trade \Wlast trade:\s(.*?) EST\W o High Day Range\W/A\W\s\WTD\W.*-(.*?)\WA o Low Day Range\W/A\W\s\WTD\W(.*?)-\s*\d ...

Table 4.1: NETworth State Transition Table

From	To	URL Transition	Condition	Expression	Output
State 1	State 2	http://www.odci.gov/cia/publications/95fact/index.html	country	href="(.*?)">{}	code.html
State 2	State 3	http://www.odci.gov/cia/publications/95fact/	code.html	none	o Country title\W(.*?)\W/title\W# o Capital Capital:\s*\W/b\W\s+(.*?)\s+ o Currency Currency:\s*\W/b\W\s*\d*\s+(.*?)\s+\(o Exchange_Rate per\sUS\$1\s+-\s(.*?)\(

Table 4.2: FactBook State Transition Table

4.3 Error Recovery

Data wrappers handles errors by printing error warnings. The execution of the CGI program halts immediately whenever an error has been detected. A Perl subroutine is written to handle all three types of possible error that the data wrappers might encounter. The subroutine takes two arguments, namely, type and explanation, and use them to display a warning message to the user.

Chapter 5

Applications

This Chapter addresses the issues about who can use the Generic Screen Scraper and how it can be used. The Generic Screen Scraper can be used in two different ways. As discussed in Chapter 1, it can be used as a black box in the COIN. It can also be a stand-alone system. As long as a data source is registered in the COIN, the Generic Screen Scraper can generate a data wrapper for the data source that is capable of extracting data from it. Any users can then use it to look up information they would like to know by just entering an SQL query.

5.1 Data Source Registry

Data Sources can be registered with the system easily, but certain assumptions and guidelines have to be followed before registering a data source.

5.1.1 Assumptions

A data source is assumed to have Web pages with "GET" methods, since HTTP-GET does not work with web pages with "POST" methods. The access control of a data source should be as general as possible, that is, it should not have only authorized access. Due to the limited power of regular expressions, the layout of the WWW page should have the format *attribute*: *value,* not *value* displayed below *attribute* on the next line.

5.1.2 Guidelines

If all the assumptions are met, a data source can then be registered. The following steps have to be followed:

- Create Export Schema file.
- Create Configuration file.

- Create Descriptor file.
- Add Descriptor file of data source to the general data source registry.

The Export Schema file, Configuration file, and Descriptor file should have the same exact name with different extensions. The Export Schema file should end in "text". The Configuration file should end in "config". The descriptor file should end in "html". See Appendix A for the specific formats of the three files that have been created for NETworth.

5.2 Users

Normal users can use the Generic Screen Scrape to issue SQL queries to semi-structured data sources as long as they knows where the configuration file resides. After keying in the URL for the Generic Screen Scraper, a screen like Figure 5.1 is displayed. Figure 5.1 is the user interface for the Generic Screen Scraper. At this point, users are expected to enter the name of the server (where the configuration file resides) and the name of the configuration file. "Context.mit.edu" and "Networth.config" are entered as server and document for NETworth configuration file as shown in Figure 5.1. Submitting at this point would bring up the subsequent data wrapper screen. Figure 5.2 is the data wrapper interface which displays the data elements defined for the NETworth and a sample query. Users are supposed to enter a SQL query following the given example. The query

select networth.ticker, networth.company, networth.high, networth.low
from networth
where networth.ticker = 'orcl'

is entered in Figure 5.2. Eventually, data are returned and displayed in tabular format to the users (if no error has been detected) as seen in Figure 5.3.

The other option is to go through the Multidatabase browser which lists all the data sources registered. User can pick a data source from there, then build a query. The multi-

database browser sends the query along with the configuration information to the Generic Screen Scraper and displays the tabulated data returned by the wrapper to the user.

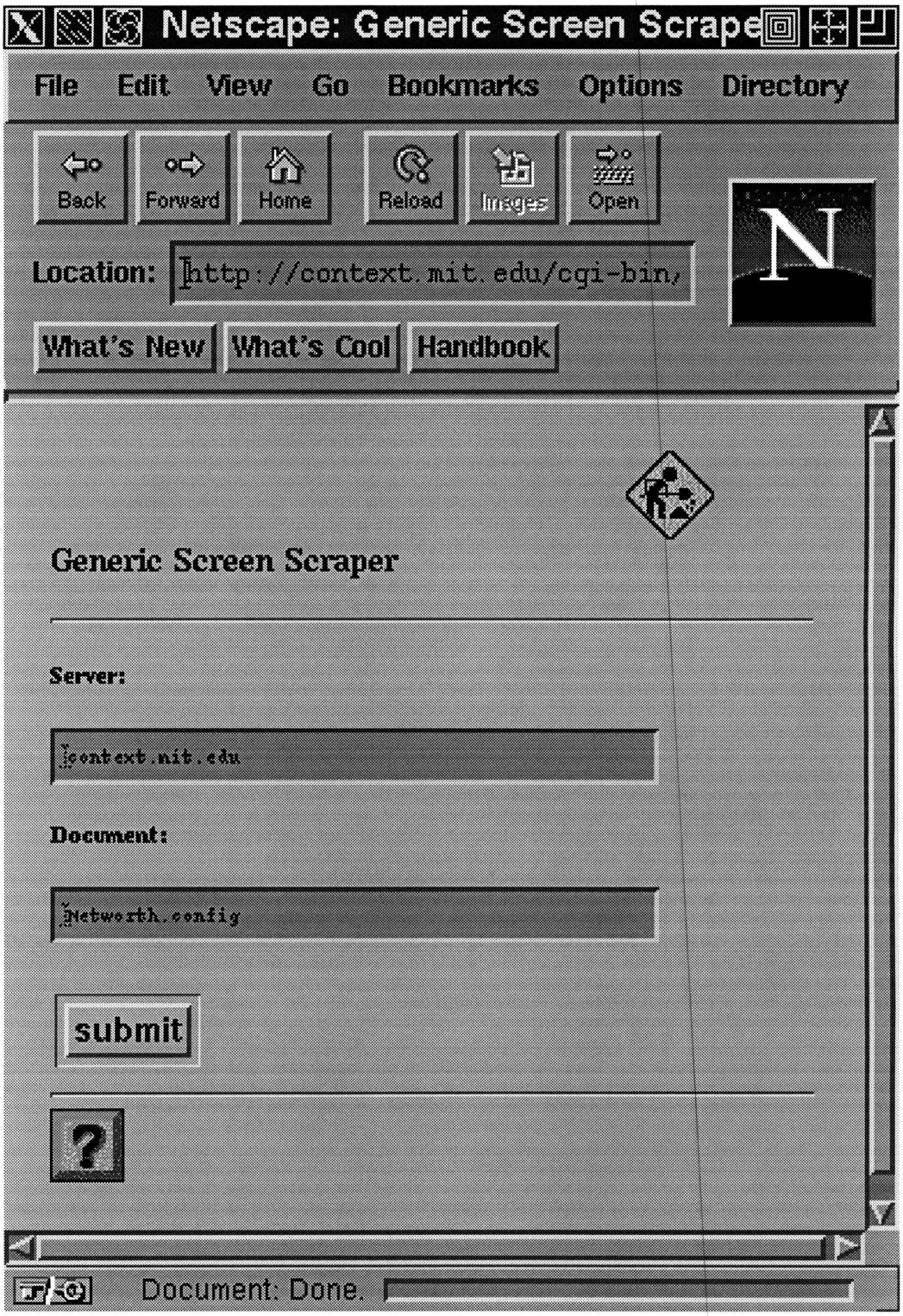

Figure 5.1: Generic Screen Scraper User Interface.

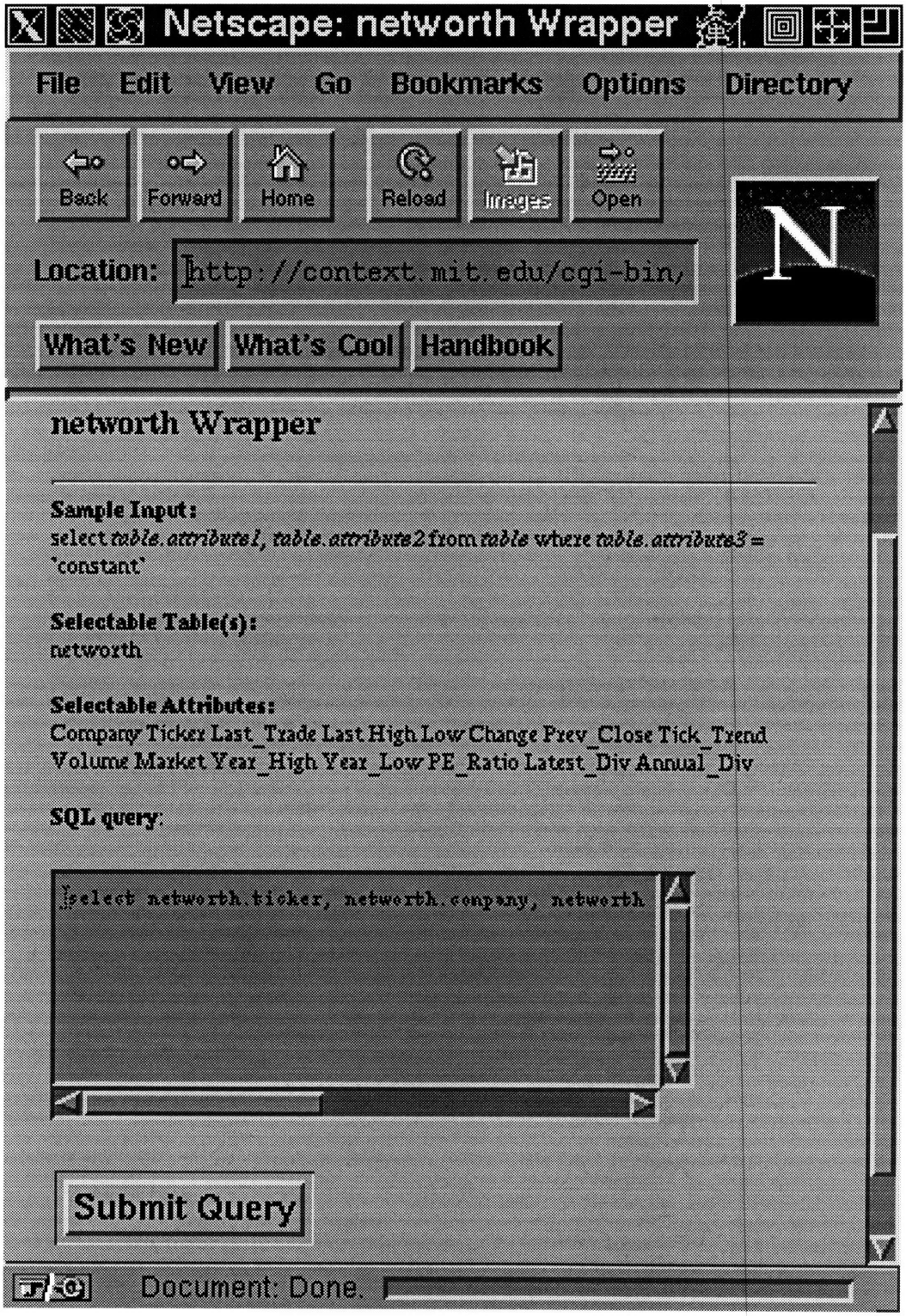

Figure 5.2: NETworth Wrapper User Interface.

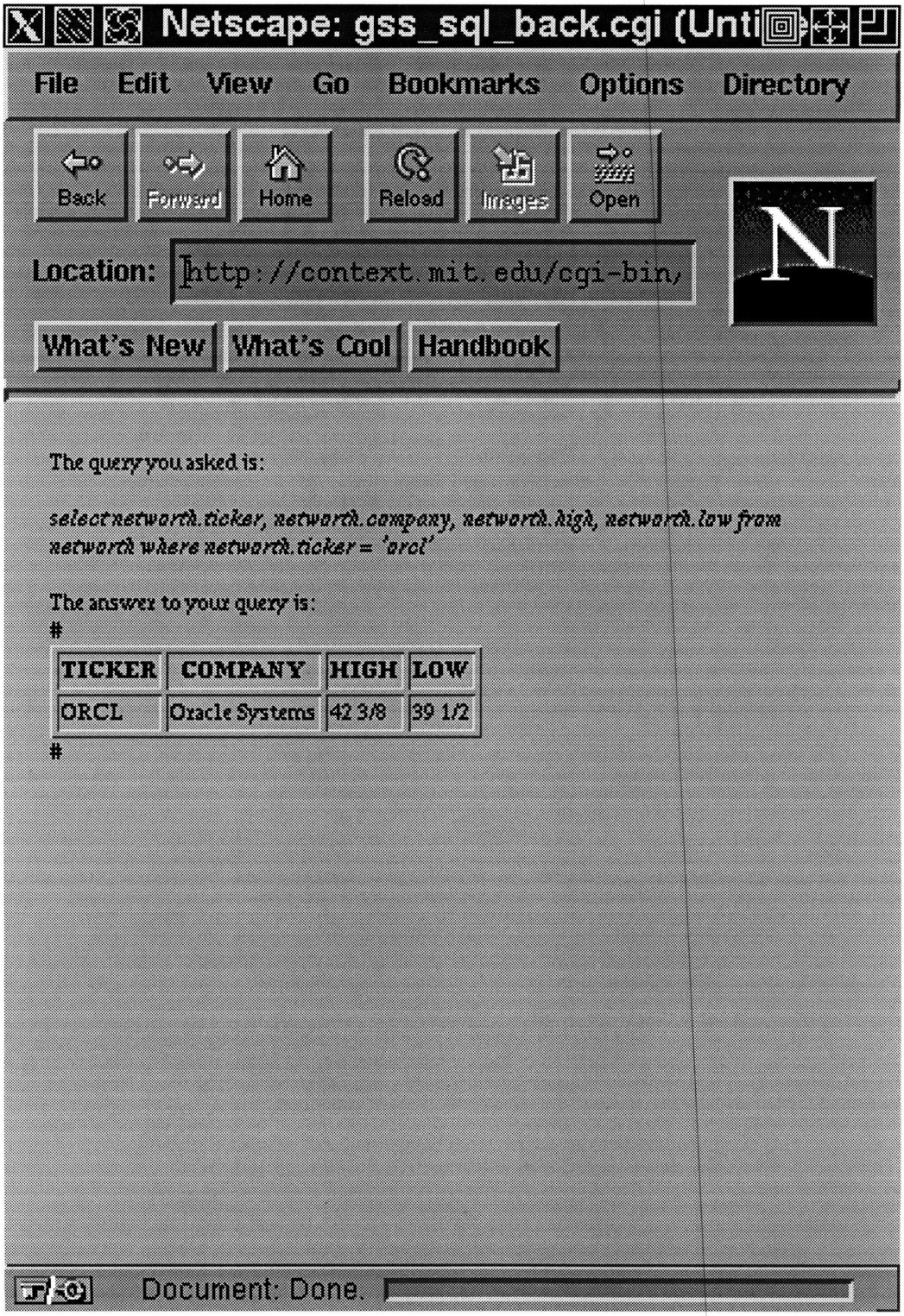

Figure 5.3: Final Page With Tabulated Data.

Chapter 6

Future Work

The Generic Screen Scraper has been a useful tool for generating data wrappers that isolate users from interacting directly with heterogeneous data sources by allowing all queries to be issued using Standard Query Language. It has been smoothly incorporated into the COIN.

The Generic Screen Scraper still has some limitations. It has not been able to scrape data off documents that have "POST" method, probably due to the fact that HTTP-GET does not get documents with "POST" methods successfully. More testing should be done with more data sources to see how these "problematic" documents can be scraped. If HTTP-GET proves to be useless in this case, programs should be written to getting documents with "POST" method.

HTTP-GET is not very stable due to the uncertainties related to the World Wide Web. Data sources often make changes to their Web interfaces as their needs change, as they become more experienced working on the WWW, or as more advanced development tools have become available. The FactBook URL has been changed because 1995 statistics replaced 1994 statistics. Therefore, a URL search engine would be useful to make sure that the right URL is being used, even if the location or the document name of the data sources have been changed. There has been some development in the implementation of a name service that will allow documents to be referenced by name, independent of their location. [1] This implementation would help the URL search engine if possibly incorporated. Another major problem is often encountered when data sources improve their user interfaces by changing the page layout or the flow control. NETworth changed the layout of their stock page when tables formats by Netscape were adopted widely by other WWW

users. Along with the layout change, the flow control was also changed. To solve these problems, the data source registerer should be alert to update the configuration file as soon as possible. The finite state automata representation of the data source in the configuration file should be checked to see if it is still compatible with the new flow control and updated according. If it is no longer compatible, the state transitions in the configuration file should be re-created. When the page layout changes, the data elements defined in the Export Schema should be updated. Moreover, regular expressions for newly defined attributes should be created just as regular expressions for old attributes should be deleted.

There are some layouts that the data wrappers can not handle due to either the limited functions of Perl or the limitation of regular expressions. Regular expressions are useful to extract information they are on the same line (i.e. the value is right next to the attribute). However, when it comes to a table that lists attributes in one row above and the values below in the next row, regular expressions become ineffective. The Stock Quote Service (*http://www3.dbc.com/dbcc/quote.html*) provided by Data Broadcasting Corporation is an example that the Generic Screen Scraper can not scrape.

In addition, data Source Registerer might find it very cumbersome to create the Export Schema, the configuration file (especially writing the regular expressions), and the descriptor file. It would be more convenient to have an automatic script that takes care of this. The automatic script should first create the Export Schema, asking the user to enter the name for the Export Schema file, then the tables and the attributes the data source being registered. Secondly, it should create the configuration file. Perhaps a recording mechanism can be created to remember all the possible paths from the start state to the final state. State transitions can then be easily generated from the recordings. A separate subroutine has be written for automatically generating regular expressions. The subroutine should have the user input the strings around the data that is to be extracted. Ideally, it

should convert the user input such as *space* and < into regular expressions such \s and \W respectively. Although this is not an easy task, it would be very useful for registering and maintaining data sources. It would come in very handy when a data source changes its page layout. Finally, the script can prompt the user to input the location for the Export Schema and the configuration file. The script should automatically write the information to the descriptor file, then add the descriptor file to the data source registry.

Last, but not the least, the Generic Screen Scraper has been used as a black box by the Multidatabase Browser or a stand-alone system. These developments have depended on the client/server model of the WWW. But some useful data sources do not reside on the WWW. As a further development, it would be good to extend the Generic Screen Scraper and make it available for ODBC compliant applications such as Excel or Access.

References

[1] Bernes-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. *The World Wide Web.* Communications of the ACM, Vol. 37. No. 8, August 1994.

[2] Cohen, Daniel I. A., *Introduction to Computer Theory.* JohnWiley & Sons, Inc., New York, 1986.

[3] Daruwala, A., Goh, C. H., Hofmeister, S., Hussein, K., Madnick, S., and Seigel, M. *The Context Interchange Network Prototype.* (Atlanta, GA, May10-Jun 2 1995). Forthcoming.

[4] Goh, C. H., Madnick, S., and Seigel, M. *Ontologies, Context, and Mediation: Representing and Reasoning about Semantic Conflicts in Heterogeneous and Autonomous Systems.* 1995.

[5] Levine, J. R., Mason, T., and Brown, Doug. *lex & yacc.* O'Reilly & Associates, Inc., Sebastopol, 1992.

[6] Tompa, F. WM., *A Data Model for Flexible Hypertext Database Systems.* University of Waterloo, ACM Transactions on Information Systems, Vol. 7 No. 1, January 1989.

Appendix A

Contents of NETworth Export Schema, Configuration, and Descriptor file

A.1 NETworth Export Schema file: Networth.text

#

networth = Company Ticker Last_Trade Last High Low Change Day_Range Prev_Close Tick_Trend Volume Market Year_High Year_Low PE_Ratio Latest_Div Annual_Div

A.2 NETworth Configuration file: Networth.config

<HTML>
<HEAD>
<TITLE>Networth Configuration</TITLE>
</HEAD>
<BODY>
<H1>Networth Configuration</H1>

<B> Method </B>
GET

<B> SQL Capabilities </B>
OR, IN

<B> Possible Paths: 1 </B>
State 1 -> State 2#

<B>Transition Table</B>
State 1 -> State 2:
URL: http://quotes.galt.com/cgi-bin/stockclnt?stock={}
Condition: ticker&
Expression: None
OUTPUT:
Company \WFONT\sSIZE=\d\W(.*?),\s+\W\w+\W#
Ticker ,\s\W(.*?)\W\W/FONT\W#
Last Last\W/A\W\s+(.*?)\s+\WA#
Last_Trade \Wlast trade:\s(.*?) EST\W#

High Day Range\W/A\W\s\WTD\W.*-(.*?)\WA#
Low Day Range\W/A\W\s\WTD\W(.*?)-\s*\d#
Change Change\W/A\W\s+(.*?)\n#
Prev_Close Prev. Close\W/A\W\s(.*?)\n#
Tick_Trend Tick Trend\W/A\W\s+\WTD\W(.*?)\W/TD\W#
Volume Volume\W/A\W\s+(.*?)\s+\WA#
Market Market\W/A\W\s+(.*?)\n#
Year_Low 52 week Range\W/A\W\s+(.*?)\s+#
Year_High 52 week Range\W/A\W\s+.*-\s(.*?)\n#
PE_Ratio P/E Ratio\W/A\W\s+(.*?)\s+\WA#
Latest_Div Latest Div.\W/A\W\s+(.*?)\WA#
Annual_Div Annual Div.\W/A\W(.*?)\n#
end
<P>

</BODY>
</HTML>

A.3 NETworth Descriptor file: Networth.html

<HTML>
<HEAD>
<TITLE>Networth Descriptor</TITLE>
</HEAD>
<BODY>
<H1>Networth Descriptor</H1>

<UL>
<LI>export schema<A HREF=http://context.mit.edu/Networth.text>
http://context.mit.edu/Networth.text</A>
<LI>wrapper<AHREF=http://context.mit.edu/cgi-bin/gss.cgi?server=context.mit.edu&document=nwregexp.html&>
http://context.mit.edu/cgi-bin/gss.cgi?server=context.mit.edu&document=nwregexp.html&</A>
<LI>source<A HREF=http://quotes.galt.com>
http://quotes.galt.com</A>
<UL>

</BODY>