

8-7-2004

Ad Hoc Integration and Querying of Heterogeneous Online Distributed Databases

Liangyou Chen

Follow this and additional works at: <https://scholarsjunction.msstate.edu/td>

Recommended Citation

Chen, Liangyou, "Ad Hoc Integration and Querying of Heterogeneous Online Distributed Databases" (2004). *Theses and Dissertations*. 374.
<https://scholarsjunction.msstate.edu/td/374>

This Dissertation - Open Access is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.

AD HOC INTEGRATION AND QUERYING OF HETEROGENEOUS
ONLINE DISTRIBUTED DATABASES

By

Liangyou Chen

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2004

Copyright by
Liangyou Chen
2004

AD HOC INTEGRATION AND QUERYING OF HETEROGENEOUS
ONLINE DISTRIBUTED DATABASES

By

Liangyou Chen

Approved:

Julia E. Hodges
Professor of Computer Science and
Engineering, and Department Head
(Major Professor)

Hasan M. Jamil
Associate Professor of Computer Science
Wayne State University
(Research Director)

Eric Hansen
Associate Professor of Computer Science
and Engineering
(Committee Member)

Ioana Banicescu
Associate Professor of Computer Science
and Engineering
(Committee Member)

Walter J. Diehl
Professor of Biology Science
(Committee Member)

A. Wayne Bennett
Dean of the College of Engineering

Name: Liangyou Chen

Date of Degree: August 7, 2004

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Julia E. Hodges

Director of Dissertation: Dr. Hasan M. Jamil

Title of Study: AD HOC INTEGRATION AND QUERYING OF HETEROGENEOUS
ONLINE DISTRIBUTED DATABASES

Pages in Study: 177

Candidate for Degree of Doctor of Philosophy

This dissertation provides an ad hoc integration methodology to manage and integrate heterogeneous online distributed databases on demand. The problem arises from an impending demand from scientific users to conveniently manage existing Web data along with the complexity involved in the construction of a functional data federation system using existing data integration technologies. We close this gap with a databases management framework accompanying novel Web data specification languages, wrapper generation technologies, and distributed query processing techniques. A major achievement of this dissertation is the establishment of a sound relational data model for Web data. Under this model, the Web becomes a synthetic extension of the traditional database systems. Consequently, a novice user of our system can cheaply integrate a large number of distributed Web sources with in-house databases for daily scientific data analysis purpose.

The relational Web modeling leads to a practical ad hoc integration system – the Meteoroid system (a Methodology for ad hoc inTEgration of Online distributed heteROgeneous Internet Data) – in the context of biological data interoperability. We identify that a main difficulty for ad hoc integration lies in the lack of a fully automated wrapper generation and maintenance technique for general semi-structured data such as HTML, XML and plain text documents. We address this issue through a thorough study of characteristics of online Web data and devise various automated wrapper techniques to facilitate robust data wrapping tasks. With this technique, form-based Web data and table-based Web data can be treated like traditional relational databases. A seamless interoperation environment for Web data and in-house databases is possible.

Another difficulty impeding ad hoc integration is in the query processing for heterogeneous distributed sources, where conflict of data is common and on demand mediation of distributed sources is desirable. The dynamicity and unpredictability of Web data further complicate the query processing task. We studied limitations posed by the Web environment for integration query processing and developed innovative techniques to expedite the early appearance of available results.

Finally we demonstrate a prototype system for ad hoc integration of heterogeneous biological data. In the system, visual Web-based interfaces guide the integration of heterogeneous data for novice users. A declarative environment is supported for ad hoc querying and management of distributed data sources.

DEDICATION

To my dear Mom and Dad, who always stand by me and support me.

ACKNOWLEDGMENTS

First, I thank my dissertation advisor, Dr. Hasan Jamil, for his continuous support and coaching of my PhD work. He has granted me generous financial aid in all my PhD study years and allowed me to focus on issues from the research. He has given me the freedom to investigate problems in the direction I am most interested and continuously inspired me with fresh ideas. He has devoted much effort in helping me to improve the quality of this study and skills to compose research papers. I have been benefited much from his supervision in the Intelligent Database Systems Research Group in the Department of Computer Science and Engineering.

I also want to thank my major professor, Dr. Julia Hodges, for her continuous revision of this dissertation and helping me refine several key concepts in it. She also kindly guided me through the procedures for my graduation. Without her help, I will not able to complete my program in time. Thanks also to Dr. Eric Hansen, Dr. Ioana Banicescu and Dr. Walter Diehl for serving on my committee and providing valuable comments for me.

I would like to thank the National Science Foundation (NSF) and the Engineering Research Center (ERC) at Mississippi State University for their financial support (grants EPS-0082979 and EPS-0132618). I will thank Dr. David Thompson for his help in securing my financial source in year 2003–2004.

I want to take this chance to thank all my friends in Mississippi State University who care about my progress, especially peoples in the Intelligent Database Systems Research Group including Zhijie Guan, Qing Zhao, Yin Zhang and Jianming Shi and my friends Zhen Liu, Wei Li and Yong Wang.

TABLE OF CONTENTS

| | Page |
|---|------|
| DEDICATION | ii |
| ACKNOWLEDGMENTS | iii |
| LIST OF TABLES | ix |
| LIST OF FIGURES | x |
| CHAPTER | |
| I. INTRODUCTION | 1 |
| 1.1 A Motivating Example | 5 |
| 1.2 The <i>Meteoroid</i> Ad Hoc Integration System: An Overview | 8 |
| 1.3 Definition of Concepts | 9 |
| 1.4 Contribution | 11 |
| 1.5 Performance Measurement | 13 |
| 1.6 Dissertation Overview | 15 |
| II. RELATED WORK | 16 |
| 2.1 Review of Heterogeneous Database Integration Systems | 16 |
| 2.2 Review of Functional Wrapper | 20 |
| 2.3 Review of Wrapper Generation | 21 |
| 2.4 Review of Wrapper Maintenance | 25 |
| 2.5 An Overall Comparison to the Ariadne Approach | 27 |
| III. REMOTE USER-DEFINED FUNCTIONS | 30 |
| 3.1 Introduction | 31 |
| 3.1.1 A Motivating Example | 33 |
| 3.1.2 Organization of This Chapter | 38 |
| 3.2 Tag Tree Data Model for Semi-Structured Documents | 38 |
| 3.3 Remote User-Defined Functions | 41 |
| 3.3.1 Using the Internet Functions | 43 |

| CHAPTER | Page |
|---|------|
| 3.3.2 Internet Function Definition Language (IFDL) | 44 |
| 3.3.3 Hyper Text Query Language (HTQL) | 46 |
| 3.4 Semi-Automatic IFDL Wrapper Generation using <i>PickUp</i> | 51 |
| 3.4.1 The <i>PickUp</i> System | 52 |
| 3.4.2 Architecture of the <i>PickUp</i> System | 53 |
| 3.4.2.1 The Data Navigation Module | 55 |
| 3.4.2.2 Wrapper Generation Module | 55 |
| 3.4.2.3 Algorithm for Wrapper Generation | 57 |
| 3.4.2.4 The Filtering and Recommendation Module | 59 |
| 3.5 LifeDB: A Prototype Database Query Interface Based on HTQL and IFDL | 62 |
| 3.5.1 The LifeDB Web-based Interface | 62 |
| 3.5.2 System Architecture | 65 |
| 3.6 Summary and Future Research | 66 |
| | |
| IV. AUTOMATIC TABLE WRAPPER GENERATION | 68 |
| 4.1 Introduction | 68 |
| 4.2 Hierarchical Repeated Structure Recognition | 73 |
| 4.2.1 Structural Relationships of HTML Elements in Tag-Tree Data Model | 73 |
| 4.2.2 Discovery of Regular Structures | 75 |
| 4.2.2.1 Target Structure Recognition | 76 |
| 4.2.2.2 HTQL Path Expression Generation | 78 |
| 4.2.2.3 Structural Relationship Recognition | 79 |
| 4.2.2.4 Model Generation and Validation | 82 |
| 4.3 Experiment of HRSR | 86 |
| 4.3.1 Experimental Results | 86 |
| 4.3.2 Comparison to Related Work | 87 |
| 4.3.3 Examples | 91 |
| 4.4 Composite Wrappers | 93 |
| 4.5 Summary | 94 |
| | |
| V. AUTOMATIC WRAPPER MAINTENANCE | 96 |
| 5.1 Introduction | 97 |
| 5.2 Automatic Wrapper Maintenance | 102 |
| 5.3 Formal Model of Table Wrapper and Wrapper Maintenance | 105 |
| 5.3.1 Model of Semi-structured Data | 105 |
| 5.3.2 Model of Table Wrapper | 105 |
| 5.3.3 Model of Type, Schema and Criterion | 106 |

| CHAPTER | Page |
|---|------|
| 5.3.4 Problem Formulation | 108 |
| 5.4 Automatic Table Wrapper Generation | 110 |
| 5.4.1 Target Structure Recognition | 110 |
| 5.4.2 HTQL Path Expression Generation | 111 |
| 5.4.3 Structural Relationship Recognition | 111 |
| 5.4.4 Performance Analysis of Table Wrappers Generation | 112 |
| 5.5 Pattern Construction | 112 |
| 5.6 Wrapper Verification | 117 |
| 5.7 Wrapper Maintenance Algorithms | 121 |
| 5.7.1 Wrapper Adjustment with Candidate Paths | 121 |
| 5.7.2 Re-targeting the Table | 122 |
| 5.7.3 Regenerate Wrapper Fields | 124 |
| 5.8 Experiment and Results | 125 |
| 5.9 Conclusion and Future Work | 130 |
| | |
| VI. THE <i>METEOROID</i> AD HOC INTEGRATION SYSTEM | 132 |
| 6.1 Introduction | 133 |
| 6.2 A Motivating Example | 136 |
| 6.3 Ad Hoc Data Integration Solution | 138 |
| 6.4 Declarative Support for Web Data | 141 |
| 6.5 Multi-layer Table and View Logical Design | 144 |
| 6.5.1 Data Source Definition | 144 |
| 6.5.2 Virtual Table Definition | 145 |
| 6.5.3 Virtual View Definition | 146 |
| 6.5.4 Advantage of Multi-layer Logical Design | 147 |
| 6.6 Scheduling-oriented Query Processing | 150 |
| 6.6.1 Dependencies Among Data | 150 |
| 6.6.2 Query Rewriting | 152 |
| 6.6.3 Query Transformation | 153 |
| 6.6.4 Query Scheduling | 153 |
| 6.6.5 Results Mediation | 156 |
| 6.7 Visual Interfaces for Ad Hoc Integration | 157 |
| 6.7.1 <i>Picking Up</i> Web Data From Scratch | 158 |
| 6.7.2 <i>Fusing</i> Distributed Web Sources | 161 |
| 6.7.3 Monitoring Query Results | 162 |
| 6.8 Conclusion | 164 |
| | |
| VII. CONCLUSION | 166 |

| | |
|----------------------|------|
| | Page |
| REFERENCES | 170 |

LIST OF TABLES

| TABLE | Page |
|--|------|
| 4.1 Automatic wrapper generation experiment results | 88 |
| 4.2 Comparison of wrapper construction work | 90 |
| 5.1 The hyper-pattern-strings (HPs) for the first tuple in Figure 5.2 | 113 |
| 5.2 The text-pattern-strings (TPs) for the first tuple in Figure 5.2 | 115 |
| 5.3 Data fields in the wrapper maintenance experiment | 126 |
| 5.4 Wrapper maintenance results for position changes | 128 |
| 5.5 Wrapper maintenance results for record structure changes | 129 |
| 5.6 Wrapper maintenance results for combined table position and record structure changes | 130 |

LIST OF FIGURES

| FIGURE | Page |
|---|------|
| 3.1 The NCBI BLAST input interface | 34 |
| 3.2 The second step: The submission ID and estimated wait time message page . | 35 |
| 3.3 The final step: BLAST results | 35 |
| 3.4 Example of an item graph. | 40 |
| 3.5 Example of a tag tree graph. | 41 |
| 3.6 Document D_1 | 49 |
| 3.7 Document D_1 displayed by a Web browser | 50 |
| 3.8 A snapshot of a wrapper generation session in PickUp | 54 |
| 3.9 The LifeDB user interface | 63 |
| 3.10 A sample LifeDB response corresponding to a query involving RUDFs. . . . | 64 |
| 3.11 The system architecture for LifeDB query interfaces | 65 |
| 4.1 The set of Homo sapiens ovarian tissue genes found in NCBI CGAP Database | 71 |
| 4.2 MGC clones for the gene NM_001614 shown in figure 4.1 | 71 |
| 4.3 The structural table from figure 4.2 recreated by the wrapper | 91 |
| 4.4 A list of books from NCBI represented using loose table structures without table tags | 92 |
| 4.5 A faithful recreation of the books table in figure 4.4 by the wrapper generated by PickUp. | 92 |

| | | |
|-----|--|-----|
| 5.1 | The NCBI nucleotide database search results | 100 |
| 5.2 | A structural view of the NCBI nucleotide database | 100 |
| 5.3 | The relationship of automatic wrapper maintenance components | 109 |
| 5.4 | Wrapper verification rules in XML | 120 |
| 5.5 | An NCBI record entry in HTML | 131 |
| 6.1 | Definition of the LocusLink data source | 145 |
| 6.2 | An example of the GenBank virtual table definition | 146 |
| 6.3 | A virtual view definition connecting DBGET/gene and DBGET/LinkDB Web tools | 148 |
| 6.4 | <i>Picking up</i> a table of Web data in a click | 159 |
| 6.5 | Customizing RUDF table creation | 160 |
| 6.6 | Connecting tables incrementally | 162 |
| 6.7 | Generating SQL queries from wizard | 163 |
| 6.8 | Monitoring in progress query results | 164 |

CHAPTER I

INTRODUCTION

Sharing data and applications as digital documents is routine in disciplines such as the life sciences. However, heterogeneity in representation, terminology and semantics in particular makes it extremely difficult for the biologists to exploit the wealth of available online resources in a coherent and ad hoc fashion. New applications in biology demand automated translation, transformation and manipulation of data from various data repositories, and processing in a non-trivial way using a complex set of tools in a pipelined fashion. Traditionally, biologists have relied on developing customized tools and data repositories. They often have done so by replicating data and tools in a way that have resulted in unmanageable redundancy and inconsistency management problems. As the applications grew complex and available data became huge, this approach to data manipulation failed from the scalability and effectiveness standpoints. It is thus safe to claim that customized portals, such as NCBI GenBank, do not provide adequate data and application support in flexible ways and often restrict their use by the vision and use anticipated by their designers. Consequently, secondary processing and remedial querying are the only ways researchers can complete their data processing needs. Such manual and offline pro-

cessing is expensive and error prone. It is desirable that a declarative query language with support for compositionality and closure property is available for such applications.

There are considerable contributions in the literature trying to explore a federated databases approach to managed Web data. The main idea is to treat Web sources as standalone databases and exploit traditional federated databases technologies to process queries across Web boundaries. It is, however, difficult for novice users to set up the necessary data federation environment in order to integrate existing Web data. For example, a user of an existing system may require days to train a wrapper for each Web source to meet the federation requirement, or each Web source may be required to provide a predefined interface that can be recognized only by the federation system. These limitations impede life science researchers to fully utilize the advantages set forth by the data integration solution.

The work described in this dissertation provides a new foundation for Web data integration under a relational framework. The main contribution is the development of a set of techniques to robustly and automatically transform Web data into relational entities such as tables and functions. The closure property of relational algebra ensures that computation of Web data can be recursively applied to distributed Web sources regardless of their physical location and representation. The robustness of our approach is re-ensured with an automatic wrapper maintenance technique. Our system is more scalable than peer systems in that both the wrapper generation and wrapper maintenance tasks have been fully auto-

mated. As a result, our system can easily incorporate a large number of Web sources, and the tedious and brittle wrapper development problem faced by other systems is avoided.

The results of this dissertation research address a problem for life science researchers to effectively utilize and exploit the explosive biological data and tools published every day in the post-genomic era. Combining biological data from distributed Web repositories and performing computations on that data have been major concerns in this work. The Web has been designed mainly for human navigation and includes little help for computers to recognize the data content. Our techniques allow computers to extract interesting structured information from Web data automatically and allow biologists to query the Web in an ad hoc fashion. Complex Web navigation and data mediation is captured and realized in a simple declarative language. Large volume data computations can be scheduled and computed automatically. Furthermore, the declarative language allows Web data and tools to be defined electronically and reused repeatedly, providing a more effective way for researchers to share data and tools world-wide.

This research closes a gap between the end-user ad hoc data integration requirement and the advantage of data federation endeavor. We show that a convenient and robust database management environment for heterogeneous online distributed database integration is possible. Under this environment, a novice user can deploy a Web data integration system on demand and computer-aided programs can be easily developed to further facilitate scientific data discovery processes.

The Web documents that biologists use to share information are primarily of two kinds - forms and documents. Forms may be embedded into a document to make it complex. Again, these documents can be classified as static or dynamic based on how they are created. Static documents are usually created once by some agent and are stored for users to access as HTML or XML documents. Dynamic documents on the other hand are created by systems in response to some stimuli - a query or update request through a form, or asp/jsp programs from stored information. Dynamic documents are context sensitive, created to respond to a particular query, and are used to meet the information up-to-date requirement.

The research described in this dissertation achieved ad hoc data integration in three steps. First, we developed an SQL abstraction for HTML forms so that such forms can be viewed as remote user-defined functions. Such an abstraction aims at reducing the impedance of mismatch between SQL databases and the HTML forms and supports a seamless integration of form documents with relational databases. As a second step, we developed simple and composite wrapper generation techniques to extract information generated by forms in response to a query. We also developed techniques to process and analyze extracted data at a local machine. Third, we developed query processing techniques for residual processing. We have done so by developing a declarative language for genomic applications that supports compositionality and closure property. Finally, we developed a prototype system to demonstrate that the ideas explored in the dissertation are sound and practical.

The research stands out in several ways in comparison to contemporary research. The emergence of the Internet renewed the interest of database researchers in investigating issues related to the interoperability of Web-based data repositories. Systems such as TSIMMIS [18], SIMS [5], HERMES [1], Information Manifold [57], WebFindit [14], Ariadne [49], and so on attempt to address this issue from several different standpoints. However, these systems, no matter how sophisticated they are, do not address ad hoc integration. Second, these systems generally adopt the wrapper-mediator architecture, which may incur a heavy burden in wrapper maintenance, but fail to exploit opportunities that exist for automatic wrapper generation and maintenance. Finally, most of these systems require expensive hardware and expert involvement for the construction of an interoperable system and hence are not suitable for occasional or small-scale users. From these standpoints, our research is unique and novel.

1.1 A Motivating Example

Accompanying the complete sequencing of whole genomes of several species, a larger scale genetic information interpretation is underway. Prediction of biological functions, for example, now has more options such as single gene similarity searches, biological pathway comparisons and genetic network references. Various biological data repositories have developed databases and Web tools facilitating genetic data analysis. For example, the BLAST tool at NCBI [83] allows searching of similar gene sequences to infer similar gene functions. KEGG (Kyoto Encyclopedia of Genes and Genomes) [12] developed

integrated pathway/genome databases allowing prediction of metabolic pathways from genome sequences. LocusLink at NCBI [59] provides a single query interface connecting curated sequences with descriptive information including official nomenclature, sequence accessions, map locations and related websites. It is, however, a painstaking job for biologists to actively utilize these distributed Web resources for even simple data analysis tasks given more and more genomic data in hand.

Consider a simple gene expression prediction scenario where a scientist wants to utilize the KEGG pathway database to predict from a set of gene IDs the relative degree that each gene is contributing to a certain gene function. Since KEGG cannot recognize a gene ID to search for participating pathways, the user needs to go through a sequence of data discovery procedures:

1. Go to the LocusLink database to retrieve the description of each gene corresponding to the gene ID;
2. Use a special tool DBGET/gene [26] at the KEGG site to convert the gene description into an entry name recognizable by KEGG database.
3. The entry name is then pasted into a DBGET/LinkDB [27] interface to retrieve the participating pathways.
4. Each pathway is connected to an XML/HTML file describing the relationships and reactions among genetic objects.
5. The pathway files are then analyzed to detect activating gene products.
6. The number of pathways each gene activates or inhibits reflects the relative degree of the gene affecting a gene function.

The above procedure needs to access various distributed data sources including:

A: The set of gene IDs for analysis in a local database;

B: The LocusLink database at the NCBI website returning a record of gene description given a gene ID;

C: DBGET/gene at the KEGG site returning a set of entry names given a gene description;

D: DBGET/LinkDB at the KEGG site returning a set of pathway links from an entry name;

E: Pathway description files at KEGG in XML/HTML format.

In this example, dataset *A* is a structured database, datasets *B*, *C* and *D* are Web tools and dataset *E* is a set of semi-structured data files. In order to analyze a gene function (for example, to identify a human cancer-activating gene), a biologist needs to go through the above data discovery procedure over and over for each relevant gene from a human genome and apply certain filtering conditions at each discovery step such as limiting the entry name to start with a 'hsa:' string (representing human gene entries).

The above data discovery procedure may turn out to be manually frustrating when the number of genes to be analyzed is large. Unfortunately, it has become routine for many biologists to have to deal with such kind of problems with even more complex operations. It is ideal that the heterogeneous data sources be managed under a uniform framework, queries be recorded and submitted in a declarative way and the complexity behind the physical data operations be hidden from biologists. From the database point of view, the above query can be simply expressed in an algebraic expression as:

$$\pi_{\text{gene-id,pathway,type}} \sigma_{\text{entry like 'hsa: \%'} (A \bowtie B \bowtie C \bowtie D \bowtie \sigma_{\text{type='activate'}} E) \quad (1.1)$$

It is the purpose of this dissertation to set up a foundation for direct application of high-level data manipulation operations – algebra operations – in the heterogeneous Web data environment. By this approach, the Web becomes a synthetic extension of the traditional database systems. Automated tools and visual interfaces can be easily developed provided with a sound database management support.

1.2 The *Meteoroid* Ad Hoc Integration System: An Overview

We have developed an ad hoc data integration system, namely *Meteoroid* (short for A Methodology for ad hoc inTEgration of Online distributed heteROgeneous Internet Data), for biologists to integrate experimental data with online resources. A navigation oriented Web interface is designed to allow users to ‘pick up’ interesting data sources and attributes from the Web by employing our automated *PickUp* wrapper technique. ‘Picking up’ a piece of information usually means a user clicking or selecting a data item. For example, in order to retrieve gene descriptions from LocusLink, a user can ‘pick up’ the search form from the LocusLink website, enter a search term, submit the search, and ‘pick up’ the gene description from the result page. A user’s behavior in this sequence of operations is captured by the Meteoroid system and is transformed into internal wrappers. A *wrapper* is a set of data extraction rules that converts semi-structured or unstructured data (such as XML and HTML data) into structured data (such as table and views in relational databases). Wrappers are learned and maintained automatically with our automatic wrapper techniques. Schema constraints and interoperability rules can be confined by user

requirements. In this example, the gene description retrieval operation can be represented as a relational view with fields of gene ID and gene description. Distributed Web data access can then be realized with a declarative language such as SQL. A visual wizard interface is designed to compose SQL expressions automatically for biological users.

The realization of this picture of ad hoc integration requires that one resolve the interoperability issue of general Web-based semi-structured data with structured data. We demonstrate the soundness of our solution through the reductions of form-based Web data into functions and data intensive Web pages into tables. The reductions make access to Web data congruent to the standard of SQL:2003 [31], which guarantees closure and compositionality. The robustness of our solution is secured with a set of automation technologies. We have verified that a certain degree of variation in Web data will not break our system.

1.3 Definition of Concepts

This section defines several concepts that will be used in this dissertation. We will assume the reader is familiar with basic database concepts such as tuple, record, schema, relational data model, etc. We will also assume the reader is familiar with the Web and HTML data.

Definition 1 (Structured data) *Structured data are data that conform to a well defined schema.*

Data managed by existing commercial database management systems are typical structured data.

Definition 2 (Semi-structured data) *Semi-structured data are tag-enriched documents that do not conform to a predefined schema. Rather, semi-structured data are self-described by the tags encoding with the data.*

HTML and XML documents are typical semi-structured data.

Definition 3 (unstructured data) *Unstructured data are plain files that are neither tag-enriched nor conform to a predefined schema.*

Examples of unstructured data are plain text files, image files, and multimedia files. This dissertation considers a limited form of unstructured data in plain text format.

Definition 4 (Data heterogeneity) *Data heterogeneity refers to the existence of data in multiple formats that conform to different data models and have different semantic meanings.*

Definition 5 (Data integration) *Data integration is the development of a centralized data management and query platform for physically distributed data sources that may or may not conform to a single data model.*

Definition 6 (heterogeneous database integration system) *A heterogeneous database integration system provides a centralized data management framework for structured, semi-structured, and unstructured data sources under a unified data model.*

Definition 7 (Wrapper) *In the information integration field, a wrapper refers to a program or a set of data extraction rules that converts semi-structured or unstructured data into structured data for data integration purpose.*

Definition 8 (Mediator) *A Mediator is a system to integrate and refine data from multiple sources.*

Definition 9 (Data interoperability) *Data interoperability is the need for meaningful integration of heterogeneous data.*

Definition 10 (Ad hoc integration) *Ad hoc integration is a data integration approach to provide timely management and querying of distributed Web data sources for non-expert users.*

An ad hoc integration system is essentially a heterogeneous database integration system. However, ad hoc integration provides greater ease of use for non-expert users. As a result, ad hoc integration addresses new issues that allows the users to acquire and integrate new data sources in a timely manner.

1.4 Contribution

The major contribution of this dissertation is the deployment of a relational database concept for Web data. The overall approach is to robustly and automatically transform Web data into relational entities such as functions and tables. Specifically, three transformation

techniques synthesize this dissertation: transformation of form-based Web data into functions, transformation of table-based Web data into tables, and transformation of integration queries into distributed Web queries. As a result, three specific contributions are made by this research. The first contribution is development of a Remote User-Defined Function (RUDF) concept for the integration of form-based Web data. RUDF makes definition and querying of dynamic Web data trivial from a database system. Development of RUDF is assisted with a novel PickUp technology and is user-friendly. The second contribution is invention of an automatic wrapper for table-based Web documents. Such *table wrapper* discovers and reconstructs structured information from Web data and becomes a building block for algebra-based Web computing. The automatic wrapper is enhanced with an automatic wrapper maintenance technique to further ensure the robustness of table wrappers and allows ad hoc integration systems to be easily scaled large. Finally we extend the traditional relational database model with the two new constructs of remote functions and remote tables and study new query transformation techniques for this extension. A novel pipeline-scheduling technique is devised to cope with a variety of Web querying problems. This results in a homogeneous database environment for heterogeneous and distributed Web data. The ability to automatically construct remote functions and remote tables ensures integration can be done in an ad hoc fashion according to the users' requirement. This resolved a problem for life science researchers to have to integrate and combine data from a rapidly increasing number of Web repositories. As a conclusion,

this dissertation establishes an *ad hoc integration* framework to resolve the integration and interoperability problem for heterogeneous and distributed Web data.

1.5 Performance Measurement

An automated wrapper technique is a main ingredient of an ad hoc integration system. High performance and robust wrapper technologies will free users from tedious manual wrapper development and maintenance tasks. We have measured the performance of our wrapper technique and compare them to related work to explain how it uniquely facilitated the ad hoc integration purpose.

First, we have measured the learning and wrapping time of our wrapper technique and compared it to reported results in related work. *Wrapper generation time* (or *wrapper induction time*) is the time to learn a wrapper from a given Web page (or a sample data set). *Wrapper execution time* is the time to execute a generated wrapper against a test page. Since wrapper learning is more complex than wrapper execution, a wrapper generation time is typically greater than a wrapper execution time. A small wrapper learning and wrapping time is favorable in an ad hoc integration system. The time to generate an effective wrapper also depends on other factors such as the necessity to develop an application ontology, the need to collect more than one sample and the time to label the sample data for wrapper training and generation purposes. We have compared these factors between various existing wrapper generation tools and shown that our wrapper tool is the most labor-free tool and is ideal for ad hoc integration.

Second, we measured the effectiveness of automatic table wrappers. This was done by measuring the success rate of a wrapper against a collection of sample table-based documents. We chose examples from major biological databases that return a table of results and popularly tested commercial Web sites that include table contents. Automatically wrapped attributes include tags that have non-blank text or special tags such as image tag. The generated wrapper was validated against 20 other pages from the same website. The error rate of each validation is the ratio of missing attributes by the generated wrapper and the total table attributes in the page. A low error rate is desirable for an automated wrapper.

Third, we measured the maintainability of our wrappers. This was done by manually changing the source documents and measuring the success rate of the wrappers to adapt to the changes. There are typically three kinds of changes for table content Web pages: table movement, where the position of the target table moves significantly in the page; record structure change, where attributes of the target table may change in column positions; and a combination of table movement and record structure change. We have manually enumerated possible changes in a Web page and create some artificial test sets based on real example pages. Since the artificial test data that was used was complete for each test purpose, the experiment results reflect the robustness of our wrapper technique for changing Web pages. Our fully automated wrapper maintenance technique is unique in the literature.

Finally we have demonstrated the applicability of ad hoc integration through the experiment of several life science examples including the example shown in the motivating

section. We have demonstrated that the experiments can be recorded in a set of declarative instructions, and the composition of the declarative instructions can be trivial for life science users. Our work is the first to provide an ad hoc integration solution for novice users to integrate and query distributed Web data sources without any expert help.

1.6 Dissertation Overview

The remainder of this dissertation is organized as follows. Chapter II reviews related work in heterogeneous database systems and wrapper generation techniques. Chapter III presents the *Remote User-Defined Function* (RUDF) technique for modeling form-based Web data. Chapter IV presents the automated wrapper generation technique for table-based Web data. Chapter V discusses an automatic wrapper maintenance technique for table wrappers. Chapter VI discusses query processing techniques for the integration of distributed Web databases. Finally, Chapter VII concludes this dissertation.

CHAPTER II

RELATED WORK

This chapter summarizes related work of heterogeneous databases integration systems and sub-systems that will be covered by this dissertation. Since our system is most similar in architecture to the ongoing Ariadne approach by Knoblock, et al. [49], an overall comparison of our work and the Ariadne approach will be given.

2.1 Review of Heterogeneous Database Integration Systems

Heterogeneous database integration has long been the focus of research in the database research community. The wrapper-mediator architecture has played a central role in heterogeneous database systems. Early systems such as TSIMMIS [18], Garlic [41], HERMES [1], and Information Manifold [57] provide tools to access multiple information sources that are available on the World Wide Web in a uniform way. A wrapper-mediator architecture is adopted in these systems.

TSIMMIS [18] develops a simple Object Exchange Model (OEM) as a common data model to describe heterogeneous data sources. Information sources are logically converted to the OEM model by wrappers. Mediators in TSIMMIS are based on patterns or rules that can be generated semi-automatically from high-level integration descriptions.

Garlic [41] uses an object-oriented model based on the ODMG standard [16] to describe data sources. Wrappers are used to encapsulate ODMG objects and their attributes. Source query capabilities are also provided in the wrappers. Garlic produces execution plans by optimizing their cost as the sum of local processing costs, communication costs, and the costs to initiate subqueries and methods.

HERMES [1] uses a logic-based language to integrate heterogeneous data sources. External programs (domains) are expressed uniformly in a special predicate of the form $in(X, d : f(Args))$, which can be interpreted as “execute function f in domain d with arguments $Args$ and store the set of results in variable X ”. A set of cost-based modules are responsible for the selection of a best execution plans that are based on the statistics cache.

Information Manifold [57] presents a declarative language to describe information sources and their query capabilities. Query plans are created based on the source descriptions. Information Manifold claims a good scalability with its source pruning algorithms.

These information integration projects provide a strong basis for the wrapper-mediator architecture in the construction of heterogeneous database integration systems. However, these systems assume wrappers for heterogeneous data sources are properly provided by wrapper experts. In real world information systems, wrapper construction is a tedious and time consuming task. This limitation restricts the availability, scalability and economicity of the integration systems.

The second generation heterogeneous database systems such as Araneus [60], InfoSleuth [10], WebFindit [14], and Ariadne [49], commonly adopt a wrapper-mediator archi-

ecture and are more focused on the practical use of the integration system. Typically they provide data fusing strategies such as domain ontologies, data exchange protocols, and visual wrapper tools to facilitate integration of distributed data sources.

Araneus [60] aims to introduce tools and techniques to manage Web bases. A Web base can be viewed as a data repository that manages Web data in a database style. Araneus models structured data in the relational model and models semi-structured data in an ADM data model, which shares many properties with the ODMG data model. Wrappers in Araneus can be constructed with the help of the Editor program [7]. Editor manipulates semi-structured documents based on two simple instructions: the “search” instruction to select ranges in a document, and the “cut & paste” instruction to restructure the document.

InfoSleuth [10] aims to mediate semantically and syntactically heterogeneous information sources in a dynamically changing environment. InfoSleuth includes a set of network agents that communicate in a high-level query language KQML [34]. Data sources and the relationships among the data are subscribed to a domain ontology. An Integrated Management Tool Suite (IMTS) provides a set of GUI tools to assist an integration administrator in analysis and creation of ontologies.

WebFindit [14] proposes the World Wide Database (WWDB) to integrate all Internet-accessible structured databases. The main purpose is to achieve scalability through an incremental construction of relationships between Web accessible databases. CORBA objects are used to register new data sources in an integration database.

Ariadne [49] is a planning system for the integration of Web information sources. Ariadne is an extension of the SIMS mediator architecture [5]. Information sources are defined with a *domain model*, where a single terminology is used. The STALKER [6, 49] tool provides a “demonstration-oriented user interface” for users to teach the wrapper generation system with a set of examples, and the approach developed by Muslea, Minton and Knoblock [63, 64] improves the wrapper generation work with active learning algorithms.

Biological data integration systems can be found in K2/Kleisli [25, 84], DiscoveryLink [42], TAMBIS [8, 80], OPM-based multidatabase [54], etc. None of these systems has tried to provide a solution for the ad hoc integration purpose. Thus, although they can setup centralized digital libraries to transparently access distributed data sources, they cannot be easily adopted by individual researchers with varying on demand purposes.

Heterogeneous database integration has also attracted recent commercial interests including the Denodo [66] and DB2 [47] systems. These systems, however, still require wrapper experts in the wrapper development stage. It typically requires hours of work to semi-automatically develop a single wrapper, and the developed wrapper has difficulty coping with the frequently changing Web data. When the wrappers collection grows large, which is a common situation for an integration system considering the huge amount of Web data, the requirement to repair invalid existing wrappers can quickly outpace any manual effort. This difficulty motivates a large body of semi-automatic and automatic wrapper generation techniques, as will be reviewed in the next section.

2.2 Review of Functional Wrapper

Web data modeling and integration using database techniques has been gaining popularity in recent years. Early research models Web data as static information sources and develops SQL-like query languages to query Web data. Integration of Web data is much similar to the integration of traditional multi-databases. As a result, research has been focus on developing query languages for Web data. The Web calculus proposed by Mendelzon and Milo [61] is an SQL-like Web query language for querying the Web. A similar language LOREL [70] is a lightweight SQL-like object query language for accessing and querying semi-structured data whose schema is unknown. W4F [74] is a fully declarative lightweight language to query HTML documents. These research projects do not consider the active aspects of the Web documents (ASP and JSP like processes) and their relationships with the databases with which they interact.

Recent research models dynamic Web data as sources of limited capability [57]. Some data sources may only accept a subset of queries. For example, querying all data from a form-based Web database (such as the GenBank database [65]) is usually impossible. Furthermore, the pattern to access and retrieve information from a Web data source is usually limited by the supplied inputs in the Web form. The limited query ability of a data sources is called *negative capability*. In contrast, some data source has extra query capability such as join of relations, sorting of data sets, etc. and is called *positive capability*. The issue of different source capabilities affecting the query planning and optimization is discussed in [41, 57, 85]. A general rational is query processing should be pushed to the sources

as much as possible for positive capability data sources, while the access pattern should be limited according to the negative capability of data sources [35, 53, 57, 71]. These research are related to our work. However, our modeling dynamic Web data as functions and is different from their approaches.

To our knowledge, the Jaguar [21] project comes very close to the approach we take for function integration. In this project, the issue of portable query processing is addressed. One of the main aims of this project is to investigate how a portable database engine can be developed based on Java user-defined functions. The goal is to exploit the opportunity of Java UDFs' portability and allow migration and execution of codes from the client side to the server side (and vice versa). The disadvantage of this system is that the Java UDFs must be coded specifically for the Jaguar system. In other words, codes not written specifically for Jaguar cannot be used. So, it eliminates the possibility of exploiting free-floating functions on the Internet as part of the database querying.

2.3 Review of Wrapper Generation

Early systems [40, 43] generate wrappers by manually specifying rules for data extraction. In [43], wrapper implementers provide templates in a high-level declarative language and actions associated with each template to design wrappers for data sources. Gruser, Raschid, Vidal and Bright [40] provide graphical interfaces and specification languages to describe extraction rules for data sources.

Manual wrapper development apparently is tedious and error prone since the raw structure of Web data can be hardly read by human. Approaches by Adelberg [2], Baumgartner, Flesca and Gottlob [9] and Azavant [75] use graphical user interfaces (GUIs) to assist the wrapper development. Tools and algorithms are provided for users to semi-automatically generate wrappers for Web data sources. However, graphical wrapper development is still time consuming and requires a great deal of wrapper knowledge to effectively operate the wrapper tools.

Graphical wrapper development cannot remove the involvement of wrapper experts. People begin to apply machine learning techniques to automatically induce wrapper rules. The most distinguished work are the STALKER [6, 63] and WEIN [50] approaches. STALKER uses a semi-automatic method to generate wrappers in three steps. First, the sources are structured by identifying the tokens and the hierarchical structures of one or more sample pages. Tokens and the hierarchical structures can be obtained semi-automatically with a set of sample pages. Users need to correct unexpected tokens or rules that describe hierarchical structures. Then a parser for the source pages can be generated automatically and communication capabilities can be added. WEIN [50] uses six wrapper classes to set up the framework for wrapper induction. A set of Web pages along with a set of label examples is trained with the six wrapper classes to generate wrappers. These approaches greatly relief wrapper developers from interacting with the wrapper programs. However, these approaches still require manually labeling sample pages, which is

itself time consuming. As a result, it requires hours' work to develop a wrapper using these programs.

In order to reduce the sample labeling work, Muslea, Minton and Knoblock [62, 64] developed a technique that uses active learning with multiple views to learn labels from samples. Target concepts are learned independently with different views. Mistakes can be recognized from views that disagree with the labeled data. The robust multi-view learner interleaves semi-supervised and active multi-view learning for the problem of incompatible or correlated views. In these techniques only Crescenzi, Mecca and Merialdo [24] generate wrappers in a fully automatic manner, and all of them require a large set of example pages as input.

RoadRunner [24] proposes techniques to remove the necessity of sample labeling. It automates the wrapper generation process by comparing two example pages at a time. Patterns are discovered from the similarities and dissimilarities of the two studied pages. Mismatches are used to identify relevant structures. The training process converges after a few examples and the computing times are generally a few seconds. The above machine learning approach has limitation that they depend on multiple sample pages. In cases where multiple sample pages are unavailable or are difficult to acquire, these approaches become invalid.

Automatic wrapper generation techniques from a single Web page can be found in XWrap [58], AutoWrapper [36], BYU tool [32, 33] and Island Wrappers [39]. All these systems except AutoWrapper and BYU tool generate wrappers from single Web docu-

ments with substantial help from the user. Although some of the systems are not fully automatic, it is instructive and pertinent to compare PickUp with these systems as we too generate wrappers for table structures from single Web sources but in a fully automatic fashion.

XWrap uses an autonomous heuristic driven boundary discovery method for the recognition of meaningful objects in a source document. It then encodes the information in XML and generates meta-data. The heuristics used can be selected by the user to influence the discovery process of XWrap. But XWrap requires a significant amount of user input and guidance and often fails to generate correct wrappers, especially when an inappropriate heuristic is selected making it an almost manual and trial-and-error based system. In contrast, PickUp does not require user guidance for non-ambiguous tables¹ and hence can induce wrappers much faster than XWrap.

AutoWrapper on the other hand uses Smith-Waterman algorithm [79] based textual similarity learning for repeated structure identification. However, unlike PickUp, AutoWrapper cannot identify table structures represented without HTML table tags, nested tables, empty tables, or even single row tables. Finally, the Island Wrapper system requires that users mark an appropriate set of example texts (training set) for it to generate the wrappers. It uses advanced elementary formal system (AEFS) to automatically learn wrappers. But the quality and success of the induction depends entirely upon the ability

¹A table structure is considered ambiguous if there exists more than one candidate tables in a Web document. Users may optionally mark the intended table structure in PickUp to disambiguate the identification correct structure.

of the users to appropriately select sufficiently expressive examples so that the system can learn patterns correctly. The strength of the Island Wrapper system lies in the fact that it is based on a formal system and the correctness of the induced wrappers can be reasoned and predicted.

Finally, the system proposed by Embley et. al [32, 33] (called the BYU tool) takes a different approach. To be able to construct a wrapper, the BYU tool requires an accurate ontology designed by expert users manually. Once the ontology is supplied, the system can map and consolidate records from multiple sources to the ontological schema. The approach is restrictive in many ways. We discuss several major limitations. First, applications involving sites with wide variations in ontological structures will extract far less information. On the other hand, PickUp can perform in applications where accurate ontologies are not available or are difficult to construct. Second, this system will not support ad hoc extraction which is the focus of our approach. Third, the approach is not suitable for table structures at higher depths. PickUp is not limited by nesting depth of tables.

Reviews of wrapper generation work can also be found in surveys by Eikvil [30] and Laender et. al. [55]

2.4 Review of Wrapper Maintenance

There is relative little work on the automatic wrapper maintenance problem. This work is typically done by wrapper experts and turns out to be tedious and error prone. We will have on a close look at recent research and point out the short-comings of existing methods.

The work most closely related to ours is by Lerman et al. [56], where new wrappers are automatically re-induced when changes in data is detected. The changes are detected by computing the features of the start patterns, the average number of tuples-per-page, the mean number of tokens, the mean token length, and the density of alphabetic, numeric, HTML-tag and punctuation types. The wrapper is re-induced by taking a set of known positive examples and a set of pages from the same data source to retrain the STALKER data extraction rules. The shortcoming of this approach is that the wrapper maintenance task becomes a heavy and time-consuming work with the mixture of supervised and unsupervised training. Furthermore, the wrapper re-induction process requires a reasonable collection of both past data and new data set, which incurs a data management overhead especially when the number of wrappers to be monitored is large. In contrast, our approach is fully automated and lightweight. Our compact and succinct wrapper verification rules are encoded with the wrapper rules. The wrapper re-induction process can be carried out stand-alone without requiring past examples. These advantages allow our wrapper and maintenance rules to be easily transmitted, stored and processed at any place and can scale up well when the wrapper collection is large.

Kushmerick [51] introduces a domain-independent wrapper verification algorithm, RAPTURE, to statistically verify the numeric features of data. Numeric features used include the digit density, upper-case density, lower-case density punctuation density, HTML density, fraction of '<' and '>' characters, length, word count and word length. The wrapper re-induction problem, however, is not present in the work. Compared to this work, our

verification algorithm works on a higher syntax level on word and HTML and thus is more sensitive to change in HTML data.

There is a set of automatic wrapper generation algorithms such as RoadRunner [24] and EXALG [4]. One may easily claim that wrapper re-induction is trivial with the ability to regenerate a wrapper once the old wrapper becomes invalid. However, in the data integration environment, high-level information such as the correlation of fields from one source to other sources may depend on the resulting data formats extracted with an original wrapper, while the regeneration of a wrapper may cause fields to be added or deleted and the format of the extraction results to be different from that of the original wrapper. Another shortcoming is that the recollection of training examples is time-consuming and some changes to the data may affect the quality of a regenerated wrapper when the effect of the change is over estimated. As a result, the naive wrapper regeneration is inappropriate for consistent wrapper extraction.

2.5 An Overall Comparison to the Ariadne Approach

Since our integration system is most similar in system architecture to the Ariadne system, we now compare the techniques used in the two systems. The wrapper technique Ariadne develops is the STALKER [6, 49] technique. STALKER works in a semi-automatic fashion. A user must provide a set of human-labeled examples to the system in order to generate a wrapper. Although active learning algorithms are provided to reduce the work of labeling data, the unavoidable human interference limits the scalability of the Ariadne

system. In contrast, our automatic wrapper technique will enable software agents to generate wrappers automatically for new websites, and the system will be able to scale up easily. The benefit with the scalability is that an integration system can easily incorporate a large number of data sources, while little extra work is needed for the development and maintenance of the integration system.

To cope with the frequent changes of Web data, Ariadne has proposed wrapper maintenance techniques to automatically detect changes and repair wrappers. However, STALKER repairs wrappers by re-inducing data extraction rules with a collection of both past data and new data sets, which incurs a data management overhead, especially when the number of wrappers to be monitored is large. Their wrapper maintenance work is a mixture of supervised and unsupervised training processes, which further impedes the system from scaling up. Our wrapper maintenance technique will not require storage of past data sets, and the wrapper repairing process will work in a fully automatic mode. As a result, our wrapper technique will be more effective for large-scale data integration systems, and data of more dynamic and heterogeneity characteristics can be integrated.

Both our approach and the Ariadne approach integrate data in a virtual manner (in contrast to the data materialization approach). In virtual integration, source data is fetched when a query is submitted and no materialization of data is present. The benefit of a virtual approach is that it is more suitable for the dynamic nature of Web data, and no inconsistency maintenance is required. However, inconsistency may originate from data from multiple sources, which happens in the data mediation stage. We will resolve such

inconsistency through an ontology that describes how to resolve the data discrepancies from different sources.

CHAPTER III

REMOTE USER-DEFINED FUNCTIONS

Similar to most scientific studies, biological analyses demand a great deal of computations and simulations involving sophisticated tools that are often found geographically distributed over the Internet. A worldwide effort in genomics research has resulted in a powerful collection of publicly available sequence analysis tools. These tools often require specialized local services and domain knowledge to function correctly, rendering them unlikely candidates for integration into remote database applications. Thus, integration of heterogeneous “functions” still remains an open problem. Providing a reasonable framework for seamless integration of these tools with database query engines will enable application developers to exploit and harness the power of these effective analysis tools. In this chapter, we present an integration framework for such tools by enabling access to them in a user transparent way as part of database queries. In our system, such online tools are abstracted as *Remote User-Defined Functions* (RUDF). An extended SQL DDL language, called the *Internet Function Definition Language* (IFDL), is presented for the specification and definition of RUDFs. The interface between the database system and the Internet is implemented using a layer based on a language called the *Hyper Text Query Language* (HTQL). The separation of IFDL, DDL, HTQL and SQL DML offers several optimiza-

tion opportunities and makes it possible to develop an architecture for interoperability of heterogeneous databases with RUDFs in simpler and more and efficient ways.

3.1 Introduction

For numerous social, political, and ethical reasons, the global effort in genomics research has produced a vast number of public data and sequence analysis tools – the building blocks of genome informatics. There has been a serious culture of sharing data and their analysis tools among researchers across the world. Somewhat standardized tools such as BLAST [3], FASTA [69], CLUSTALX [82], Sacch3D [73], and so on, are public, open source and are available online at sites such as the NCBI and Stanford. Researchers around the world have adapted these tools to suit their needs and made these improved and enhanced tools available for public use on the Internet. Often, these tools depend on copyrighted or protected domain knowledge, and on specific hardware configurations such as cluster computing, super computers or parallel machines. This dependence on specific system requirements makes it difficult, if not impossible, to replicate the runtime environment on the user's part to exploit the power of interesting tools.

When new analysis tools are developed or existing ones are enhanced or adapted, they encode application, system and mission specific properties. Usually they handle data in uniform and known formats across platforms within the system. The output of the execution is relatively stable (constant), and the execution behavior and duration of execution are predictable. While researchers make these functions available on the Internet, the in-

tended use of the functions are usually limited to their internal system. In other words, these tools allow users to exploit in-house resources given a certain piece of data (DNA sequences or the likes) and return results of analysis on the input data, usually one piece at a time.

There has been a great deal of interest in making biological databases interoperable so that biologists could share their knowledge with greater ease and exploit experiences of others' to expedite explorations in science. One coordinated effort in this direction is the Gene Ontology project of the Gene Ontology Consortium [22]. The goal of this project is to produce a dynamic controlled vocabulary that can be applied to all eukaryotes even as knowledge of gene and protein roles in cells is accumulating and changing. Other projects include TAMBIS [68], European Union Bridge Database Project Consortium [38] and BioKliesli [77], to mention a few. Most of these projects focus on accessing the data from remote sites, possibly database systems, for integrated processing with local data. The analysis tools are assumed to be resident in the local sites. In other words, integrating remote analysis tools with database querying is not emphasized. In this way, they force code migration, as opposed to data migration, from remote sites. This approach ignores the issue of the cost of code migration and adaptation, or the impossibility of such migration because of not addressing this aspect in their framework for the interoperability.

Most online biological analysis tools are implemented using CGI, Java or ASP type platforms so that accessing them using Web forms and subsequent navigation of the document structures are possible with considerable ease. The complex nature of Web-based

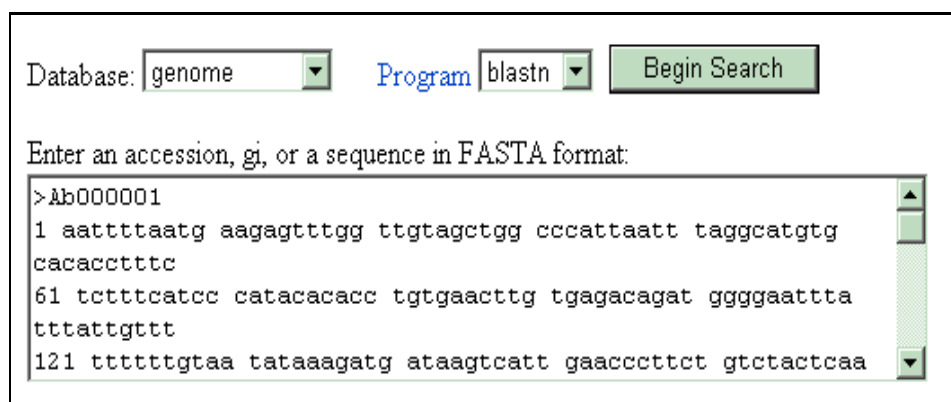
forms and their dynamic and interactive nature makes interoperability of these tools extremely difficult if code migration is not used. This is partially due to the facts that Web-based forms generally need manual input that is subjective. They produce results using complex scientific calculations, the output of one process is digested by another process to generate answers, hierarchical forms may accept inputs from the previous forms, some outputs are time dependent, some functions are hierarchical or structured, and outputs may have several pages needing further navigation to assimilate the results. Most wrapper based applications in interoperable systems for heterogeneous data sources cannot easily adapt to such complex requirements.

3.1.1 A Motivating Example

Consider an application where a biologist wants to find homologous sequences corresponding to all her sequences stored in a local database from the GenBank database at NCBI site using BLAST that meet a given similarity threshold (e-value). Currently, there are two possible ways to accomplish this goal. The biologist can submit her sequences one at a time (some systems may allow a set at a time submission) to GenBank and receive the results in an asynchronous mode by e-mail, or via an online mode after a pause which becomes increasingly longer for subsequent submissions). She can alternatively download the whole GenBank at her site along with the BLAST program, compile BLAST as a user-defined function and use it in a suitable SQL like query to obtain her desired results. But by doing so, she now becomes responsible for maintaining GenBank as a warehouse as

new sequences are being added to the GenBank at NCBI site. This essentially means that she will have to manually monitor the NCBI site for possible BLAST program upgrade if she wishes to stay current.

Both options have their advantages and disadvantages. The first option – using NCBI BLAST tool and GenBank data – is extremely slow and manual, but has the least management overhead. It works as follows. Each sequence against which a BLAST analysis is required must be manually submitted to the GenBank through an interface such the one shown below.



Database: Program:

Enter an accession, gi, or a sequence in FASTA format:

```
>Ab000001
1 aattttaatg aagagtttgg ttgtagctgg cccattaatt taggcatgtg
cacaccttcc
61 tctttcatcc catacacacc tgtgaacttg tgagacagat ggggaattta
tttattgttt
121 ttttttgtaa tataaagatg ataagtcat gaacccttct gtctactcaa
```

Figure 3.1 The NCBI BLAST input interface

Once the sequence is submitted, a request Id for the submitted analysis is assigned to the biologist through an interface similar to the one shown below. Along with the request ID, the system also informs the biologist of an estimated execution time of the BLAST request.

The request ID is

or

The results are estimated to be ready in 4 seconds but may be done sooner.

Figure 3.2 The second step: The submission ID and estimated wait time message page

A manual intervention (a mouse click) is required at the end of the estimated time of execution to finally view the BLAST hits that are displayed as an HTML document similar to the one shown below. The biologist will have to repeat this process for all her sequences and collect the results manually for continued processing at her site. In [46], a conservative estimate for a similar query session was discussed. It was demonstrated that one needs to visit about 16,000 pages and spend more than 20 hours in wait time alone.

```
>ref|NT_025938.2|Hs22_26094 Homo sapiens chromosome 22 working draft sequence segment
      Length = 65461

      Score = 50.1
      bits (25), Expect = 0.002
      Identities = 40/45 (88%)
      Strand = Plus / Minus

Query: 261   tcgatgaagaacgcagcgaaatgcgataagtaaatgtgaattgcag 305
          ||| | | | | | | | | | | | | | | | | | | | | | | | | | | |
Sbjct: 16868 tcgatgaagaacgcagctagctgcgagaattaatgtgaattgcag 16824
```

Figure 3.3 The final step: BLAST results

The latter option of warehousing GenBank is equally expensive in terms of management and investment as discussed earlier. So, the question remains: is it possible to use BLAST at the NCBI site as part of a database operation at a remote site without code and

data migration? We answer positively in this dissertation and show that we can achieve identical results by treating BLAST as a *remote user-defined function* with greater efficiency and ease of use compared to the first approach.

The main idea is as follows. As most analysis tools on the Internet are form-based and accept several input values to generate a predefined form of output, we can regard such tools as functions in the sense of conventional programming, and rightfully call them *Internet functions*. But in contrast to conventional functions, Internet functions are not very particular about input and output data types as they are able to resolve some of the disparities themselves. Hence, in the set up that we envisage, the determination of the output type is somewhat difficult without some server-side assistance as we plan to use these functions within a database query engine which usually is serious about the data types it handles.

We propose that such Internet functions are defined at the database level as a remote user-defined function with the help of an extension of SQL data definition language, called the *Internet function definition language* (IFDL), proposed in this dissertation. The input output behavior of the Internet function is abstracted through the IFDL expressions and the function is used as ordinary SQL user-defined function in SQL query expressions. The interface between the database system and the Internet function is implemented at a layer called the *hyper text query language* (HTQL) which is responsible for the execution of the query and gathering results over the Internet. The HTQL system layer includes a user

interface through which the user interacts with the extended database, query processor, IFDL module, HTQL engine and the local database system.

There are a few technical hurdles to overcome. First, a seamless integration strategy for the Internet function will be required since functions from multiple sources can be nested in an arbitrary fashion. Second, specialized techniques must be used to interface the function defined within a database system and the actual function accessible over the Internet. Finally, efficient and effective techniques must be developed to store, manage and manipulate intermediate results from the Internet functions before the final output can be determined.

Several other system considerations also become important. For example, communications between systems over the Internet are slow and prone to stalling and breakage. So, there is a choice of implementing the query processor in batch or pipelined fashion. There is also a choice of the appropriate time to integrate the outputs of multiple online functions. We address these issues through our language HTQL which helps bridge the two platforms – the Internet and the database system – which essentially present a relational view of Web documents to the users. The trick is now to view function or program heterogeneity as data heterogeneity and exploit well researched techniques found in the literature on database interoperability.

3.1.2 Organization of This Chapter

The remainder of this chapter is organized as follows. First we introduce a *tag tree* data model in section 3.2 for general semi-structured data as a basis for our further discussion. Then we introduce the concept and definition of remote user-defined functions (RUDF) in section 3.3. RUDF is based on an extended data definition language called *IFDL* for SQL, discussed in section 3.3.2, and a query language called *HTQL* for semi-structured data, discussed in section 3.3.3. A semi-automatic HTQL expression generation tool, called *PickUp*, for the IFDL statements is presented in section 3.4. Then, in section 3.5, we introduce a query interface for LifeDB database systems that is currently being developed at Mississippi State University. Finally, we summarize in section 3.6.

3.2 Tag Tree Data Model for Semi-Structured Documents

Our discussion of semi-structured data is based on a *tag tree* data model we developed for XML, HTML, and plain text documents. In this model, a document is composed of *items*. An item is a segment of text in a document, which can be an atomic word, a sentence, or any continuous text region in the document. An item is defined with an enclosed *start-tag* and an optional enclosed *end-tag*, presented in the form */'start-tag'~'end-tag'/. The region in which an item resides starts from the start-tag and ends at the end-tag. When the enclosed end-tag is missing, the item ends at the end of the document. The text between the enclosed tags is called a *text item*. A tag defined naturally in an HTML or XML document is called a *hyper-tag* and is presented in form *<start-tag>*, where the start-tag*

is also called the *tag-name* of the hyper-tag and the item it defines is called a *hyper-item*. In contrast, a non hyper-tag is called a *plain-tag* and the item it defines is called a *plain-item*. Since any string can be defined as a plain tag, plain-items can only be recognized at run time based on a query. The tags in a document naturally induces a partial order among the items that can be used as a basis to enforce a *reachability* relationship among the items. Roughly, an item *a* is reachable from another item *b* if the start position of *a* is within the scope of the item *b*. The *scope* of an item is define recursively as the region of the tag under a *parent scope* reachable to the item, where the root scope is the document item. The reachability relationship of items is used to generate the so called *item graph* of a document. We illustrate these concepts in the following examples.

Example 1 Consider an HTML document fragment shown below extracted from the document in figure 3.6.

```
DBSOURCE:<a href=/LocusLink/refseq.html><em>REFSEQ:</a>
NC_004088.1</em>
```

This HTML fragment has the following properties. The tags in it overlap in scope. For example, the start-tag ‘’ is inside the scope of the ‘<a>’ tag, but the end-tag ‘’ is outside the ‘<a>’ tag and hence overlaps. Considering only hyper-tags, an item graph of this fragment has two nodes, ‘<a>’ and ‘’, and an edge from ‘<a>’ to ‘’. However, if we consider two plain tags ‘DBSOURCE:’ and ‘\n’ (such tags are possible only in HTQL), then the item graph with respect to the tag set {<a>, , ‘DBSOURCE:’, ‘\n’} would be the graph shown in figure 3.4.

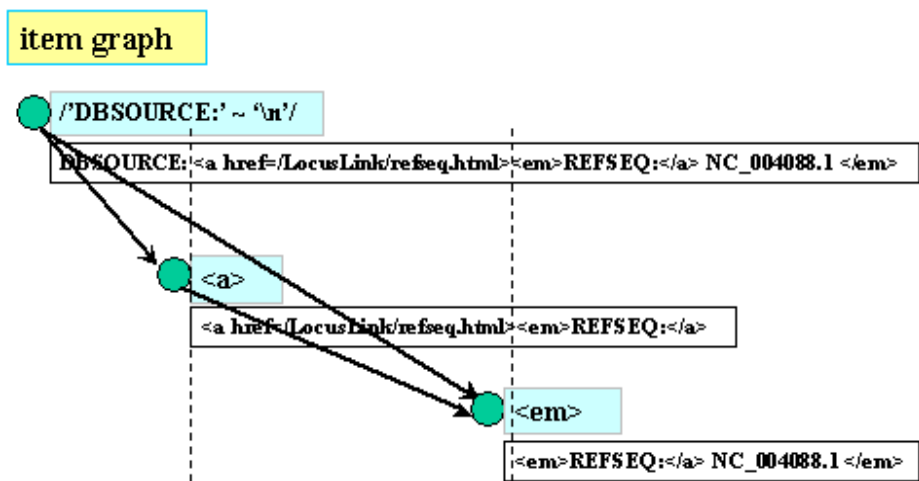


Figure 3.4 Example of an item graph.

Given an item graph, a tag tree can be developed, which essentially captures the navigational possibilities within the item graph. Such graphs are constructed at run time with respect to a query to be able to accommodate plain tags and also to reduce search space. Notice that query tag sets are significantly smaller than the document tag sets, and consequently the corresponding graphs relative to query tags are smaller. Also, the relationships lost in the graphs based on query tags are irrelevant for the query.

For now, assume that the HTQL query we have in mind is “/‘DBSOURCE:’~‘\n’/” (syntax of HTQL will be discussed shortly). Then, for the item graph shown in figure 3.4 and the query “/‘DBSOURCE:’~‘\n’/”, the *tag tree* shown in figure 3.5 can be constructed.

The tag tree graph can be effectively used to browse the document for answering a query. For example, the query below requests the “REFSEQ:” portion of the DBSOURCE in the HTML fragment in example 1.

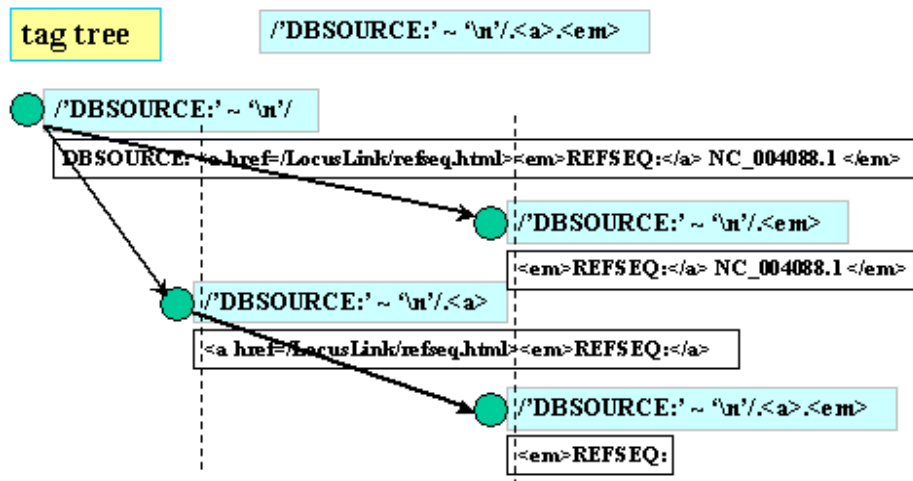


Figure 3.5 Example of a tag tree graph.

/DBSOURCE:' ~ '\n'/.<a>.

3.3 Remote User-Defined Functions

Incorporating an arbitrary function into a database system as a user defined function is by no means an easy task. For successful integration, the function must satisfy type, parameter and structure requirements in addition to several system-specific format restrictions. Once integrated, any alteration in code would necessitate a re-compilation of the user-defined function. Remote user defined functions have the potential to eliminate any restrictions of the types mentioned above and would not necessitate a re-compilation in the event of an alteration in the algorithm it encodes.

Remote user-defined functions differ from conventional user-defined functions in several ways. First, they are not compiled as part of the database. Second, they run on another system and utilize both local and remote resources. Third, they are accessible over the In-

ternet and include a communication protocol that they share with the accessing databases for establishing handshaking at run time. Most analysis tools accessible over the Internet have a form-based interface and thus, already offer the communication protocol. Hence, they do not need any additional requirement for integration. However, functions that are accessible but are not based on Web forms, may need modifications in order to address the handshaking issue. Modifications required in functions written in some languages can be extremely simple and trivial, such as in C.

In a different yet similar context, Godfrey, Mayr, Seshadri, and Eicken [37] have identified three principal ways in which (remote) user-defined functions can be exploited.

- The UDF runs at the server site and within the server process.
- The UDF runs at the server site in a process isolated from the server.
- The UDF runs at the client site.

The Jaguar project addressed the first scenario where the UDF runs at the server site within the server process. This choice over the other two is reasonable given security and efficiency concerns. In the second choice, however, a significant cost may be incurred every time a process leaves its process boundary. On the other hand, serious security risks exist in the third choice when active components of UDFs enter a client site process [37]. A further drawback is the efficiency and the latency in call invocations since now the server will have to ship data and function arguments to the client site for processing, often in a tuple-at-a-time basis.

The concept of remote user-defined functions that we introduce here is a combination of choices 1 and 3. In our system, one client process and one server process participate

in evaluating a user-defined function, which could potentially be a piece of code only and have nothing to do with a database process. The client process takes the role of a coordinator and establishes communication with the server site, sends arguments to the server process, follows required execution steps, gathers intermediate results potentially from multiple sources and integrates the results. Before integration, the client site process may store the intermediate results at the client site, but it never accepts codes from remote sites.

It may appear that by choosing this architecture for RUDF implementation, we are inviting the drawbacks associated with choice 3 above, but actually we are not. We are still evaluating the UDF at the server site but giving control of execution to the client site. Besides, this choice is unavoidable for several reasons. Many Internet functions are simply computational tools that are not part of any database system and do not access any data stored in any database whereas others do. For the purpose of abstraction, it is desirable that we encapsulate the function and view the whole as a complete system in a uniform way. This is possibly the best way to achieve a non-intrusive interoperability of such functions and focus only on its input-output behavior rather than its internal architecture or logic.

3.3.1 Using the Internet Functions

Consider a specific case where we would like to find all possible Kenyan fruit fly (*Drosophila melanogaster*) sequences from the GenBank database that are 98% similar to the sequences in a local sequence database called *local*. Let us examine how SQL, enhanced with remote

user-defined functions, enables us to attain this goal effectively and efficiently by circumventing the complicated process discussed in earlier sections.

Example 2 A simplified SQL representation of the query is presented below. All the user has to do is submit this query to the local database query interface shown in figure 3.11.

```
select b.sequence
from (select get_seq(blast(a.sequence))
      from local as a) as b
where b.organism = "Drosophila" and b.source(country)="Kenya" and
      b.e-value ≤ 0.02
```

In the above expression, *get_seq* and *blast* are two remote user-defined functions. All the expressions say is that for every sequence in the *local* table, perform a BLAST search in GenBank and obtain the sequences for each BLAST by invoking the *get_seq* function. Here, we are assuming that the *blast* function returns the request ID, and once the request ID is passed to the *get_seq* function, it extracts the output sequences along with the similarity scores (e-value) and details of each sequence. A subsequent selection on this set of sequences collects all homologous sequences of Kenyan fruit flies. Readers may have noticed that we are viewing the extracted sequences as complex tuples where traversing the complex data structure of the tuples is possible through sub-structure operator, i.e., () in the expression “b.source(country)”.

3.3.2 Internet Function Definition Language (IFDL)

The simplicity of the SQL expression presented in example 2 can be somewhat deceptive because in order to achieve such a level of clarity and intuitive simplicity, we need to

address several more complicated issues. The first issue is the definition of the remote user-defined functions. In this section, we present a simple extension of SQL's data definition language by proposing an Internet Function Definition Language.

To be able to effectively define an Internet function we need to specify the URL where the function is located, the input parameters and output values with their type restrictions, and an instruction on the location of the answer in the returned HTML document by the RUDF. Consequently, an Internet function definition is a 5-tuple $\langle \phi, v, \pi, \rho, \varrho \rangle$ where ϕ is the function name, v is the URL of the function, π is the list of input parameters with types, ρ is the output value type, and finally ϱ is the output extraction expression. Notice that the expression in ϱ will be used to extract the answer and will be stored in ρ in the local database. The expression in ϱ can be any valid query expression in some HTML or XML query language such as XML-QL [28], XQL [72], UNQL [15], etc. But for the purpose of this chapter, we will be using a new lightweight and reasonably fast query language called HTQL. A few salient features of HTQL will be discussed in section 3.3.3. The process and syntax of a pair of IFDL function definitions for the *blast* function is discussed using the example below.

Example 3 Consider the following RUDF definition for the *blast* function in the Internet function definition language (IFDL). Recall that a call to the BLAST results in a form that returns the request ID. And once the request ID is submitted again after an estimated pause, the answers can be viewed in an HTML document. The IFDL syntax used for defining the *blast* function $\langle \text{blast}, \text{"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi"} \rangle$,

```
query varchar ( 1000 ), request_id varchar ( 40 ),
```

`<form>.<input>3:value` is given below.

```
define function blast
href "http://www.ncbi.nlm.nih.gov/blast/Blast.cgi"
parameters query varchar ( 1000 )
results request_id varchar ( 40 )
htql <form>.<input>3: value;
```

The intuitive meaning of the above definition is as follows. The *blast* function found at the URL "http://www.ncbi.nlm.nih.gov/blast/Blast.cgi" accepts a variable length string up to 1000 characters long and returns a maximum 40 character long ID that can be found in the input field of the first *form* tag in the returned HTML document. Recall that the HTQL processor understands the meaning in the expression ϱ and correctly helps extract the ID from the HTML document.

Similarly, the IFDL function definition for the *get_seq* RUDF can be written as follows:

```
define function get_seq
href "http://www.ncbi.nlm.nih.gov/blast/Blast.cgi"
parameters rid varchar ( 40 )
results sequence varchar ( 10000 )
htql <pre>.<pre>;
```

3.3.3 *Hyper Text Query Language (HTQL)*

Hyper Text Query Language (HTQL) is a semi-structured data extraction language that can handle XML, HTML and plain text documents. HTQL is used in the IFDL definition for access and transformation of Internet data. This section provides an overview of its syntax. HTQL sentences follow a general structure as follows.

```

(document) [pattern expression]
  [, (document) [pattern expression]]*
  [{variable assignment}]
  [//condition involving variables and constants/]
  [(pattern construction)]

```

Note that the [] notation in the above definition means the expression inside it is optional, and * means zero or more repetitions of expressions. The pattern expression clause is similar to a path expression in other object-oriented and Web query languages. However, it is differently formed as we will discuss shortly and uses the so called *tag selection* and *dot* operations. The pattern expression clause can have repeat pattern specifications from multiple documents; such expressions form a Cartesian product of possible “items” in the documents involved. Optionally, condition clauses and pattern construction clauses can be used with pattern expressions. However, whenever a condition clause or pattern construction clause is used, a variable assignment clause may be used to facilitate proper reference to the item segments.

There are six basic operations supported in HTQL – (i) the tag selection, (ii) dot operation, (iii) plus operation, (iv) item attribute and text extraction, (v) collapse operation, and (vi) a set of extended functions. Every operator in HTQL takes a “sequence of items” as input and returns a “sequence of items” as output. In this connection, the reader may recall that a document itself is a sequence of one single item. Furthermore, the tag tree for such an arrangement is well defined. In the following series of examples, we will introduce the operators and the functions mostly in reference to the document D_1 shown in figure 3.6. The document D_1 has been synthesized from three source Web pages at NCBI – a Map

viewer page (at <http://www.ncbi.nlm.nih.gov/cgi-bin/Entrez/framik?db=genome&gi=250>), a nucleotide page (at <http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?val=22123923>), and a 3-D structure page (at <http://www.ncbi.nlm.nih.gov/cgi-bin/Entrez/tablik?gi=250>) about *Yersinia pestis KIM* so that we can effectively explain HTQL features using one source document. The HTML rendering of D_1 on the screen is shown in figure 3.7.

Tag Selection: A *tag selection* operation returns all text items within the scope of the tag. The tags that are considered are the ones reachable from the root node of the tag tree corresponding to the query. There are two forms: t , and $'p_1' \sim 'p_2'$ where t is a regular HTML or XML tag, and p_i s are plain tags or text strings. Since plain tags are defined apriori, a scope is needed as a pair of plain tags and hence the syntax $'p_1' \sim 'p_2'$. The query $(D_1) <a>$ will return the document below:

```
<a href=/htbin-post/Taxonomy/wgetorg?id=187410>Yersinia pestis KIM</a>
<A HREF=ftp://ftp.path/NC_004088.gbk>NC_004088 </A>
<A HREF=ftp://ftp.path/AE009952.gbk>AE009952 </A>
<a href=plink?cut=95&chrom=250&pid=22123923>22123923</a>
<a href=plink?chrom=250&pid=22123923&set=1&cut=95>33</a>
<a href=plink?cut=95&chrom=250&pid=22123924>22123924</a>
<a href=plink?chrom=250&pid=22123924&set=1&cut=95>7</a>
<a href=/LocusLink/refseq.html><em>REFSEQ:</em></a>
<a href=/cgi-bin/Entrez/tablik?gi=250> 3D Structure </a>
<a href=/cgi-bin/Entrez/taxik?gi=250> TaxMap </a>
<a href=/cgi-bin/Entrez/gen_blast?uid=250&rps=1> CDD </a>
```

The query $(D_1) / 'DBSOURCE:' \sim '\n'$ will return the following document:

```
DBSOURCE: <a href=/LocusLink/refseq.html><em>REFSEQ:</em></a> NC_004088.1
</em>
```

It is also possible to refer to tags by their relative position. As all tags are indexed in the tag tree, such operations are easily handled. For example, the query $(D_1) <a> 1 - 2$ will return the following items.

```
<a href=/htbin-post/Taxonomy/wgetorg?id=187410>Yersinia pestis KIM</a>
<A HREF=ftp://ftp.path/NC_004088.gbk>NC_004088 </A>
```

But, $(D_1) <a> 4$ will return only

```
<a href=plink?cut=95&chrom=250&pid=22123923>22123923</a>
```



```

<html>
<head> <title> Yersinia pestis KIM </title> </head>
<body background = 'ground.gif'>
  Organism: <a href=/htbin-post/Taxonomy/wgetorg?id=187410> Yersinia pestis KIM
    </a> <br>
  RefSeq: <A HREF=ftp://ftp.path/NC_004088.gbk>NC_004088 </A>
  GenBank: <A HREF=ftp://ftp.path/AE009952.gbk>AE009952 </A> <br>
  Total Bases: 4600755 bp <br>
  Completed: Jul 29, 2002. <br>
  3D Structure:
  <table name=table1 bgcolor=white width=400
    <tr> <td> <b>Gene</b> </td>
      <td><b>3-D</b></td>
      <td><b>protein name</b></td></tr>
    <tr> <td> <a href=plink?cut=95&chrom=250&pid=22123923>22123923</a>
      </td>
      <td><a href=plink?chrom=250&pid=22123923&set=1&cut=95>33</a></td>
      <td> initiation of chromosome replication </td></tr>
    <tr><td><a href=plink?cut=95&chrom=250&pid=22123924>22123924</a>
      </td>
      <td><a href=plink?chrom=250&pid=22123924&set=1&cut=95>7</a></td>
      <td> regulator for asnA, asnC and gidA </td></tr>
  </table>
  <p> <pre>
  AUTHORS:<em>Deng,W., Burland,V., Plunkett,G. III</em>
  DBSOURCE:<a href=/LocusLink/refseq.html><em>REFSEQ:</a> NC_004088.1
    </em>
  </pre><p> <hr> <p>
  <table name=table2 >
    <tr><td align=center>
      <FORM NAME="goto" ACTION="altvik" METHOD="GET">
      Search for gene
      <INPUT TYPE=text NAME=gene VALUE="" SIZE=16>
      <INPUT TYPE="submit" VALUE="Find"></td></tr>
    <tr><td>
      <table name=table3 width=100% >
        <tr><td> <a href=/cgi-bin/Entrez/tablik?gi=250> 3D Structure </a>
          </td><td> <a href=/cgi-bin/Entrez/taxik?gi=250> TaxMap </a>
          </td><td> <a href=/cgi-bin/Entrez/gen_blast?uid=250&rps=1> CDD </a>
          </td></tr>
        </table></td></tr>
    </table>
  </body>
</html>

```

Figure 3.6 Document D_1

```

Organism: Yersinia pestis KIM
RefSeq: NC\_004088 GenBank: AE009952
Total Bases: 4600755 bp
Completed: Jul 29, 2002.

3D Structure:
Gene      3-D protein name
22123923 33  initiation of chromosome replication
22123924 7   regulator for asnA, asnC and gidA

AUTHORS: Deng, W., Burland, V., Plunkett, G. III
DBSOURCE: REFSEQ: NC\_004088.1

```

Search for gene

[3D Structure](#) [TaxMap](#) [CDD](#)

Figure 3.7 Document D_1 displayed by a Web browser

Dot Operation: *Dot operations* can be performed along with tag selection for navigational purposes. For example, the query $(D_1)\langle a \rangle.\langle em \rangle$ will return the text item “REFSEQ:”, whereas the query $(D_1)/\text{'DBSOURCE:'} \sim \backslash n'/. \langle em \rangle$ will return the text “ $\langle em \rangle$ REFSEQ: $\langle /a \rangle$ NC.004088.1 $\langle /em \rangle$ ”.

We must mention here that tag selection and dot operations together help form a path expression. A path expression identifies an item in a document with the tag at the end of the path expression. The tag associated with the item thus identified becomes the current *context*. The next operation uses the idea of current context.

Item Attribute and Text Extraction: Using this operation, attributes of any tag and its text content can be accessed. Since tag attributes have names, the values can be accessed by simply referring to the names. However, the text inside a tag have no such identifiers. For this reason, HTQL uses a special keyword *tx* to refer to the text component of an item in current context. For example, href and tx in the context of $(D_1)/\text{'Organism:'} \sim \backslash n'/. \langle a \rangle$ means the strings “/htbin-post/Taxonomy/wgetorg?id=187410”, and “Yersinia pestis KIM”. However, such syntax can only be used in the variable assignment, condition and pattern construction clauses of HTQL sentences. Extraction can be qualified with path expressions such as $/\text{'Organism:'} \sim \backslash n'/. \langle a \rangle: tx$ to mean “Yersinia pestis KIM”

Plus Operation: A *plus operation* is used to combine two expressions to produce one single result. For example the expression $(D_1)\langle em \rangle + \langle i \rangle$ unions (in sequence) all emphasized text items with all italics that are reachable from the root of the tag

tree graph. Hence, the response of the query $(D_1) \langle em \rangle + \langle i \rangle$ of document D_1 would be the documents “ $\langle em \rangle$ Deng,W., Burland,V., Plunkett,G. III $\langle /em \rangle$ ” and “ $\langle em \rangle$ REFSEQ: $\langle /a \rangle$ NC_004088.1 $\langle /em \rangle$ ”

Collapse Operation: Given an item, the *collapse* operation returns all of the attribute values of the item. Hence for the item $\langle name \text{ first}='Bob' \text{ last}='Barr' \rangle$ Name of Employee (D) $\langle /name \rangle$, the expression $\@ \langle name \rangle$ will return ‘Bob Barr’ for some D .

3.4 Semi-Automatic IFDL Wrapper Generation using *PickUp*

IFDL and HTQL languages together make declarative specification of form-based Web data possible. It is, however, still a burden for novice users to have to remember all the syntax. It is an advantage of our declarative languages that this burden can be eliminated with a reasonably complex automated tool and algorithms.

In this section, we present a semi-automatic query and wrapper generation system, called *PickUp*, for use by mediation systems, such as the LifeDB interoperable database system introduced in this dissertation, for extracting information from large volumes of Web data. The system is equipped to visually recognize structures in Web documents and generate candidate HTQL queries for extracting contents associated with the structures. A heuristic ranking is used to identify the best candidate query for use by the wrapper or the query system. It is our contention that the dual operation mode of the *PickUp* system, user-guided and fully automatic, makes it an attractive choice for many e-commerce as well as many non-traditional applications.

The general idea of the *PickUp* system is as follows. We posit that wrapper generation is a two-step process in which, given a set of data sources and a query goal, we

learn the structural relationships of the document contents that are necessary to respond to the query and then generate appropriate wrapper codes in the subsequent step. Filtering conditions can be applied to the contents of the documents at run time, or even at the wrapper generation time, if such conditions are known apriori. To learn wrapping rules in a semi-autonomous fashion, a learner will need to study several candidate structure queries (perhaps along with content filter conditions,) corresponding to a query goal and a set of Web sites for which the wrapper is being designed. This is essentially the same concept used in ontology research for automatic ontology generation. It can then either generalize the queries (optimistic approach) or accept the query that returns the most restricted answer (conservative approach) and produce a wrapper. The PickUp system we present here serves as the candidate query generator for the learning system we have outlined.

The PickUp approach is reminiscent of the Lixto system [9]. In functionality, it almost parallels Lixto except that PickUp does not demand iterative refinement, and it is capable of producing a set of ranked candidate expressions for the extraction of the identified data. In an automatic system, a best candidate can be automatically chosen as the target query.

3.4.1 *The PickUp System*

The *PickUp* system is a graphical user interface (GUI) for automatic generation of HTQL wrappers for HTML or XML data sources. The development platform for our system is Microsoft Visual C++ on Windows. It has a Web browser control allowing users to navigate to arbitrary Web data sources and browse the site for supervised wrapper generation.

It has several sophisticated point, click and mark type region-marking options. The region marked is submitted as a target for analysis with a button click (the “Find from Selected HTML” button). The system displays a ranked set of HTQL expressions and the corresponding text that these expressions will wrap as shown in figure 3.8. The ranking of the expressions is based on the relative cost of computing the expressions. In general, it is possible to have more than one expression for extracting a document segment in HTQL. The system ‘recommends’ the best (least cost) expression as the wrapper. In PickUp, regions can be marked in two principal ways – by highlighting a portion in the traditional way (point and drag), or by highlighting a start and an end point. In the latter case, once start and end points are marked, PickUp can discover the HTML or XML structure that generates the particular region enclosed within the structural boundary of the marked text. This region may not be a regular geometric region on the computer screen in general.

In PickUp, users have the option to choose documents with hyper links or abandon them altogether by selecting only the visible plain texts. It is also equally possible to extract browser invisible texts from the documents. The “Attributes” radio button allows the selection of these choices. Figure 3.8 shows the PickUp system interface in action.

3.4.2 Architecture of the PickUp System

The *PickUp* system is composed of four modules – (i) a data navigation module for browsing Web sites, (ii) a user-guided HTQL wrapper generation module, (iii) a filtering and recommendation module for heuristic ranking of candidate wrapper expressions, and fi-

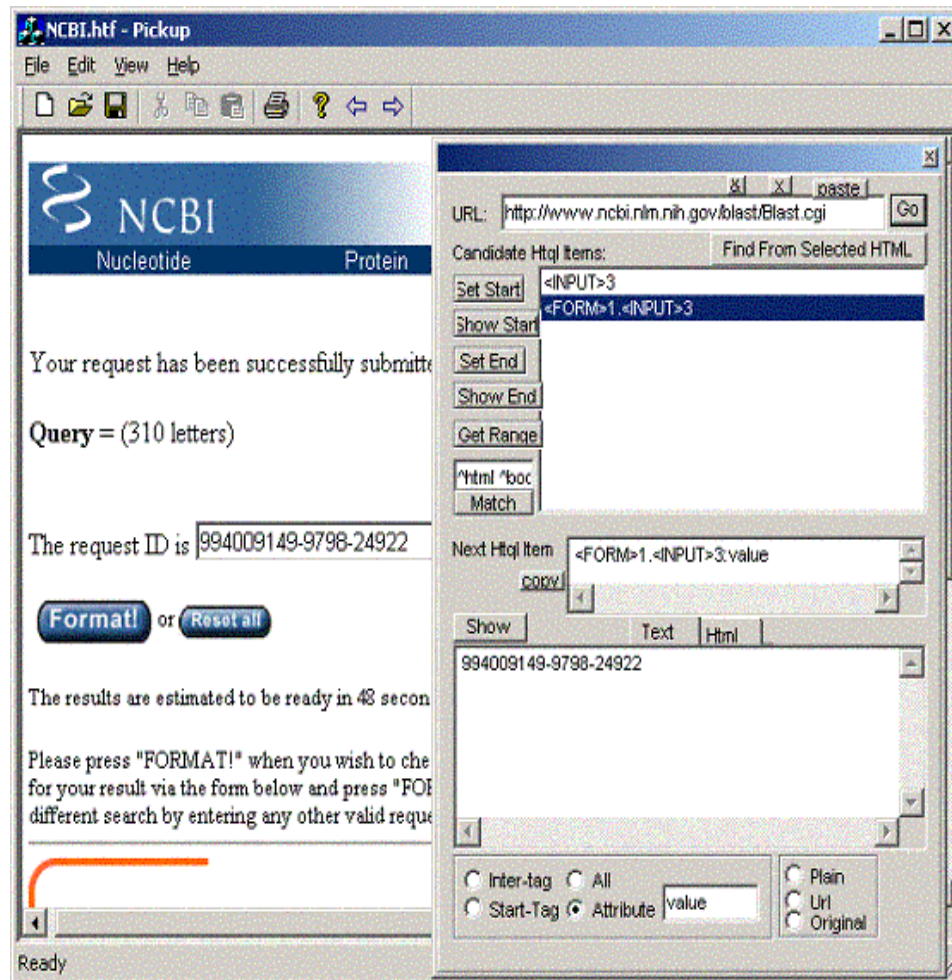


Figure 3.8 A snapshot of a wrapper generation session in PickUp

nally (iv) a validation and fine-tuning module for error trapping, validation and refinement of HTQL expression execution. In the following sections, we present a brief discussion on each of the modules.

3.4.2.1 The Data Navigation Module

The purpose of the data navigation module is to let users bring a sample HTML page from the Internet directly by accessing the URL. The “Microsoft Web Browser” ActiveX Control Object has been used to implement this module. Consequently, this module has capabilities and behavior identical to Microsoft Internet Explorer. Highlighted segments in any document can be identified using an API in this module, and necessary applications are designed to realize its functionalities.

3.4.2.2 Wrapper Generation Module

Before we proceed with the discussion on the wrapper generation process and the module which generates a set of ranked candidate HTQL expressions, we need to formally define a few concepts based on the tag tree data model and item graphs we have mentioned before.

Definition 11 (Reachable Relationship Set) Let D be a document and $T(D) = \{t_1, \dots, t_n\}$ be the set of all hyper tags in D . A tag $t_i \in T(D)$ is reachable from another tag $t_j \in T(D)$, denoted $t_i \rightarrow t_j$, if there exists an explicit edge from t_i to t_j in the item graph for D . The *reachable relationship set*, denoted Γ_D , is all such relationships in the item graph I .

Lemma 1 Γ_D can be constructed in $O(n^2)$ time.

Proof sketch: Γ_D can be constructed by comparing each pair of tags in $T(D)$. The total number of pairs needed to be compared is $\frac{n(n-1)}{2}$. Therefore, the time to determine Γ_D is $O(\frac{n(n-1)}{2}) = O(n^2)$.

Definition 12 (Reachable Parents) Let t_i be a tag in $T(D)$. Given a tag $t_i \in T(D)$, the set *reachable parent* of t_i , denoted $P_D(t_i)$, is the set $\{t_k \mid t_k, t_i \in T(D) \wedge t_k \rightarrow t_i\}$. The set *direct reachable parent* $P'_D(t_i)$ of t_i is the set of all tags t_j that are reachable from t_i but not from tags $t_k, k \neq i$ that are also reachable from t_i . In other words, $P'_D(t_i) = \{t_k \mid t_k \in P_D(t_i) \wedge \forall t_m \in P_D(t_i) \Rightarrow \neg(t_m \rightarrow t_k)\}$.

Lemma 2 For a given $t_i \in T(D)$, $P_D(t_i)$ can be constructed in $O(n)$ time and $P'_D(t_i)$ can be constructed in $O(n^2)$ time.

Proof sketch: One can construct $P_D(t_i)$ by comparing t_i with each tag in D . The time needed is $O(n)$. $P'_D(t_i)$ can be constructed by comparing each pair of tags in $P_D(t_i)$. Since the total number of tags in $P_D(t_i)$ is less than n , the comparison will not exceed n^2 . As a result, $P'_D(t_i)$ can be constructed in $O(n^2)$.

Definition 13 (Reachable Siblings) Let *tag-name*(t) denote the tag name of a tag t . For tags $t_i, t_j \in T(D)$ and a reachability relation $t_j \rightarrow t_i$, the *reachable sibling* of t_i under t_j is $\varsigma(t_i \mid t_j) = \{t_k \mid t_k \in T(D) \wedge t_j \rightarrow t_k \wedge \text{tag-name}(t_k) = \text{tag-name}(t_i)\}$.

Lemma 3 Given tags $t_i, t_j \in T(D)$ and $t_j \rightarrow t_i$, the reachable sibling $\varsigma(t_i \mid t_j)$ can be constructed in $O(n)$ time.

Proof sketch: The reachable sibling can be constructed by comparing t_i with all direct children of t_j , with a cost of $O(n)$.

Once a region selection is made by highlighting, the position information of the highlighted segments is used to map and extract the subgraph in the item graph that corresponds to that region. In fact, during the construction of the sub graph, the reachability relationships are maintained. Since the reachability relationship set traces a path from a given node to the root, multiple paths may be found, resulting in multiple candidate wrapper expressions. For example, if the text “22123924” in figure 3.6 is highlighted, the following HTQL expression will be generated as all these expressions describe the text “22123924” equally correctly.

```
<table>1.<tr>2.<td>1.<a>1
<table>1.<tr>2.<a>1
<table>1.<td>4.<a>1
<tr>2.<td>1.<a>1
<table>1.<a>2
<tr>2.<a>1
<td>4.<a>1
<a>2
```

3.4.2.3 Algorithm for Wrapper Generation

We are now ready to present the algorithm based on the concepts and complexity results discussed above. Algorithm *find_candidates* constructs candidate wrapper expressions with respect to the highlighted texts in the browser based on the direct reachable parents set of the identified tags in a recursive fashion. Essentially, it finds the reachable paths from an item in a backward fashion to the root by identifying all reachable parents.

Consider the highlighted text “22123924” as explained at the outset of this section. The highlighted tag $\langle a \rangle$ of “22123924” has the reachable parent set $\{\langle td \rangle, \langle tr \rangle, \langle table \rangle\}$. As such, in order to extract “22123924” with its associated tag $\langle a \rangle$, all the HTQL expressions $\langle td \rangle 4. \langle a \rangle 1$, $\langle tr \rangle 2. \langle a \rangle 1$, $\langle table \rangle 1. \langle a \rangle 2$, and $\langle a \rangle 2$ can be used equally. The goal of this algorithm is to generate all of these expressions automatically. Furthermore, if the reachable parent of $\langle td \rangle$ tag is selected as a basis, for instance, there are various ways to express this $\langle td \rangle$ tag. If $\langle table \rangle 1. \langle td \rangle 4$ and $\langle td \rangle 4$ are taken as two example forms for this $\langle td \rangle$ tag, the $\langle a \rangle$ tag of “22123924” can be expressed as $\langle table \rangle 1. \langle td \rangle 4. \langle a \rangle 1$ and $\langle td \rangle 4. \langle a \rangle 1$ respectively.

Algorithm. (Find all candidates) Given a semi-structured document D with the reachable relationship set of Γ_D , a tag $t_0 \in T(D)$, algorithm *find_candidates* generates a set of ranked HTQL expressions corresponding to the tag $t_0 \in T(D)$.

```

Function find_candidates ( $\Gamma_D, t_0, suffix$ )
Returns candidates
Begin
  Let  $t_p := t_0$ ;
  While  $P'D(t_p)$  is not empty
    Let the first tag in  $P'D(t_p)$  be  $t_1$ 
    Let  $t_p$  has  $k$  reachable siblings under  $t_1$  before  $t_0$ ;
    Let  $S := tag-name(t_0) + to-string(k + 1)$ ;
    If  $t_1$  is not  $D$ 
      Then
        Let  $R := find\_candidates(\Gamma_D, t_1, S)$ ;
        For each  $E$  in  $R$ 
          Append expression  $E$  with "." and suffix;
          Add  $E$  to the candidates;
      Else
        Add  $S$  to the candidates;
    Let  $t_p := t_1$ ;
    If  $tag-name(t_p) = tag-name(t_0)$ 
      Then break;

```

End Function

We now describe in detail how this algorithm works. The algorithm starts by searching expressions for the `<a>` tag of “22123924”, where t_0 is equal to the `<a>` and Γ_D is the reachable relationship set 1. The direct reachable parent of `<a>` is the `<td>` tag by the relationship `<td>4 → <a>2`, where `<a>` is the second reachable sibling under the `<td>` tag. The variables t_1 , t_p , and S are assigned respectively the tags `<td>`, `<a>` and the string “`<a>2`”. Then the `find_candidates` function is invoked in a recursive fashion to search for expressions for the `<td>` tag, which is now variable t_0 's new value. In this recursion, similar to the first recursion, the variables t_1 , t_p , and S are assigned respectively the tags `<tr>`, `<td>` and the string “`<td>1`”. The recursion continues until t_0 is equal to the `<table>` tag, t_1 is D , and S is “`<table>1`”, where a candidate result is generated as “`<table>1`”. As the recursive call terminates, the candidate expression ‘`<table>1.<tr>2.<td>1<a>1`’ is generated on exit. The first recursion loops further and takes a higher level of reachable relationship by giving variable t_p to the parent of one of its element tag. The variables t_1 , t_p , and S respectively, are assigned the tags `<tr>`, `<td>`, and the string “`<a>1`” in this loop. This time the expression ‘`<table>1.<tr>2.<a>1`’ will be generated. The process continues until all candidates are found.

3.4.2.4 The Filtering and Recommendation Module

Let the cardinality of $P_D(t_0)$ be d . The maximum branching factor of searching the candidate expressions by the algorithm `find_candidates` is thus d , which is characterized by

the number of loops in each recursion. The search depth of the problem is also d , which is represented by the maximum depth of recursions. The maximum number of candidates generated by algorithm *find_candidates* is thus in the order of $O(d^d)$. For example, if $d = 10$, it may have 10^{10} candidates, which is quite prohibitive. Hence, we discuss below a number of heuristic rules that can be used to filter non-interesting candidates and reduce complexity.

Rule 1 (Limit Expression Depth) Accept only expression with the smallest number of dot operations.

This heuristic helps lower the complexity of the search space. Consider two candidate expressions ‘<html>1.<body>1.<table>1.<tr>2.<a>1’ and ‘<table>1.<tr>2.<a>1’. The ‘<html>1.<body>1’ part of the formal expression conveys no useful information since almost every HTML page has a <html> tag and a <body> tag. Removing this part from the expression will result in an expression identical to the latter one. In general, a longer expression may contain more useless information than a shorter expression that computes an identical query.

Rule 2 (Prefer Visual Skeleton Tags) Prefer expressions containing visual structuring tags such as <table>, <tr> and <div> over non structuring tags such as
, <body>, etc.

Let us revisit the problem of wrapper generation for the text “22123924”. One possible interpretation of a user selecting the item “22123924” is that the user is interested

in the second hyper link in the sample page. If this interpretation is correct, the expression ‘<a>2’ best describes the data the user has selected for wrapping. Another possible interpretation is that the user is interested in the hyper link in the second row of the table in the sample page. In this interpretation, the expression ‘<table>1.<tr>2.<a>1’ becomes more meaningful. Generally, the <table> tag, the <tr> tag or the <div> tag is more powerful in capturing the visual effects of HTML formatted documents. Rule 2 takes advantage of this observation and in PickUp, the recommendation module pays more attention to such ‘skeleton tags’ in ranking candidate expressions. In contrast, the <body> tag or the <tbody> for example, can be neglected during the search as such tags are assumed less interesting. However, this ranking is relative and uninteresting tags may become interesting in the absence of more interesting tags which guarantee a successful search under every condition.

Rule 3 (Exploit Application Specific Feature Tags) Exploit and enforce application specific preferences in wrapper generation.

Exploiting domain specific knowledge can greatly improve the search process. Unfortunately, such knowledge incorporation can only be achieved on a case by case basis. One specific example is the data sources that are pretty stable and change rarely. Even if such sources change appearances and contents, the constructs with respect to a wrapper operation may stay stable. For example, consider the case of the NCBI BLAST page through which users may submit a nucleotide sequence for homology search against the sequences

in GenBank. While the content of this page changes from time to time, the form structure is almost stable. For this page, the `<form>` tag and the `<input>` tag can be treated as skeleton tags. For this particular case, an expression of the form '`<form>1.<input>1`' is preferable over an expression of the form '`<table>2.<tr>2.<input>1`,' even though they are identical in functionality. This is because the document may contain only a couple of submission forms while the table structure may be complex for the display of huge scientific data sets. In some cases, the form names and identifiers may also be used to generate compact wrapper expressions.

These are some of the heuristics we have incorporated in PickUp to rank the candidates and reduce the search effort. Usually, the top ranking expression is accepted unless the user selects an alternate expression. The PickUp interface lets users select any candidate generated by it, and displays the query result for that expression for the purpose of verification.

3.5 LifeDB: A Prototype Database Query Interface Based on HTQL and IFDL

3.5.1 The LifeDB Web-based Interface

LifeDB is a database system for life sciences applications that is being developed at Mississippi State University. In this section, we will introduce the LifeDB system in the context of RUDFs. We will discuss how they are implemented, the system architecture and related issues.

The LifeDB system has a Web-based query interface through which users write ad hoc queries and view answers. The interface workspace is mainly divided into three areas – query editor area, result/display area and query list area as shown in figure 3.9.

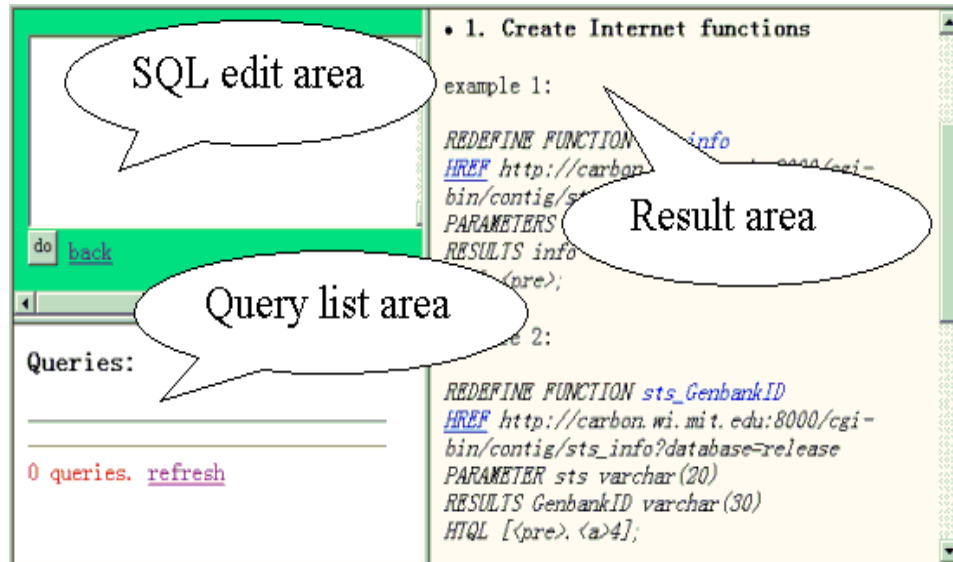


Figure 3.9 The LifeDB user interface

Ad hoc queries in HTQL and IFDL extended SQL can be submitted in the edit window. The results of the execution are displayed in the result area. For every query, a materialized view is maintained so that computed answers can be retrieved when needed. Users must explicitly delete the materialized views when they are no longer needed. Every view receives a reference number that is hot linked and a simple click on the link brings back the answer to the result area.

When a query involving an RUDF for which the function definition is already compiled successfully is submitted (as shown below), results and messages related to the query ap-

pear in the result window. Unfortunately, the computation of RUDF can be arbitrarily long due to Internet delays, traffic volume, site work load, and so on. So to assure the user of progress and to improve system performance, a pipelined output streaming is employed. The result pane is refreshed every few seconds for new computations and the output display is updated with new rows. This way, users receive answers without having to wait for a long time until the execution completes. It thereby offers an opportunity to abort the execution if the responses are not desirable.

As shown in the figure 3.10, once the computation is complete, a unique query ID is assigned to the view and is stored in a local table. The query list shows the query ID with its list of attribute names picked up from the select clause of the query.

```
select sts_code, GenbankID from genes
where sts_GenbankID(sts_code) is not
null
```

do [back](#)

Queries:

[345452](#) sts_code, GenbankID

1 queries. [refresh](#)

Results

query: **345452**

| sts_code | GenbankID |
|----------|-----------|
| WI-5270 | G04845 |
| WI-5275 | G04849 |
| WI-5276 | G04850 |
| WI-5278 | G04851 |
| WI-5279 | G04852 |

5 records found. [refresh](#)

[Return Home](#)

Figure 3.10 A sample LifeDB response corresponding to a query involving RUDFs.

3.5.2 System Architecture

The main goal of the LifeDB query interface for remote user-defined functions is non-intrusiveness at the local as well as remote systems level. Thus, the development of our interface does not require any modifications whatsoever at the local or remote database sites. The architecture presented in this section also facilitates non-intrusive future plugins and extensions. We now present the system architecture of our interface.

Figure 3.11 shows a diagrammatic view of the system which includes a user interface, a query analyzer, query controller, function execution engine, IFDL parser, HTQL engine and the local database engine. In addition to all the modules, there is also a meta data repository for various system functions as we will be discussing next.

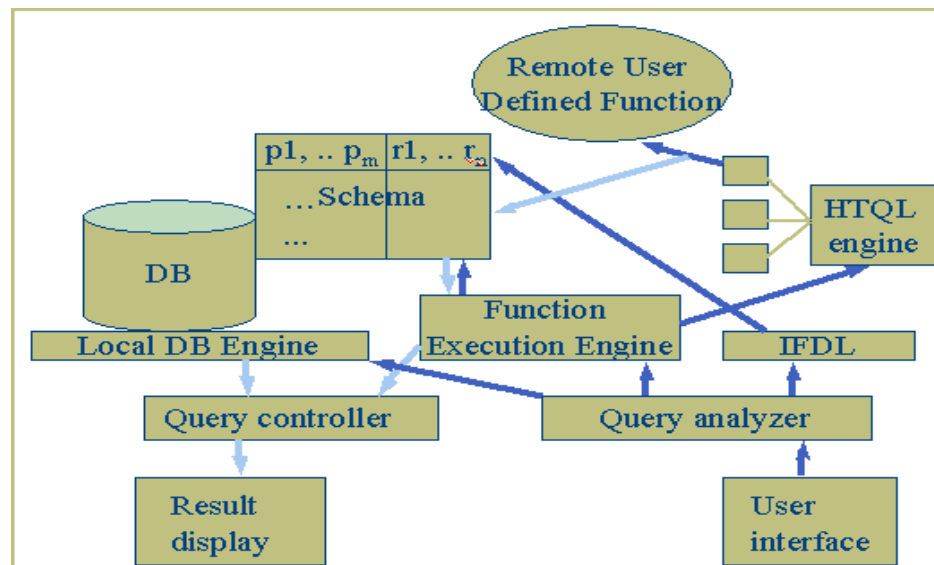


Figure 3.11 The system architecture for LifeDB query interfaces

3.6 Summary and Future Research

The primary focus of this chapter was to demonstrate that Web forms can be viewed as Internet functions and that these functions can be treated as remote user-defined functions for a local database to perform interesting queries. We have demonstrated that such an approach simplifies application development by allowing users to write query expressions involving these functions without having to worry about interfacing with remote systems or dealing with low level details. We have presented an architecture for a query interface that supports a high level abstraction of any Internet function through an extended DDL called the IFDL. The IFDL, in conjunction with HTQL, is capable of supporting declarative interfacing mechanism for the RUDF in a local database.

We have also presented a semi-automatic wrapper generation system PickUp for unstructured Web documents. We have emphasized two aspects of this system – (i) it avoids iterative refinement of the candidate wrappers needed in many systems, such as Lixto, in order to support scalability, and (ii) it uses a declarative Web query language HTQL as its wrapper, giving it a sense of platform independence and greater degree of portability. These two properties were made possible by the use of HTQL as the wrapper language. We have also presented a brief introduction to the HTQL language and the tag tree data model on which HTQL is based.

While the research at Cornell (Jaguar [37], Cougar [11] and Predator [76]) addresses the issue of code migration as a mechanism for portability, we look at the other side of the issue – portability without code migration. We buy currency, anonymity, independence

and interoperability, perhaps at the cost of efficiency. We plan to investigate the execution latency aspects of our system in our future research and hope to devise ways to speed up the execution as much as possible. However, we also were able to avoid any type of wrapper generation needed in most other systems, which is often time consuming, expensive and demands maintenance.

In our opinion, the contributions PickUp has made are significant and novel since scalable user guided wrapper generation is now possible, due mainly to HTQL and its underlying data model. As a next step, one can build a rule learner system, a change monitor system and concept to rule mapping system that can be used in concert with PickUp for an almost automated wrapper generation and management system for structured as well as unstructured data sources.

There are other smaller issues such as accuracy of data and early expiration of data. By nature, Web-based systems do not guarantee the accuracy of the data it presents and data often expires depending on several factors. For example, it is possible to receive cached responses from a proxy server, and connections may be interrupted. A robust system must take these issues into account and deal with them. We plan to address these issues also in our future research.

CHAPTER IV

AUTOMATIC TABLE WRAPPER GENERATION

Biological data analyses usually require complex manipulations involving tool applications, multiple Web site navigation, result selection and filtering, and iteration over the Internet. Most biological data are generated from structured databases and by applications and presented to the users embedded within repeated structures, or tables, in HTML documents. In this chapter we outline a novel technique for the identification of table structures in HTML documents. This identification technique is then used to automatically generate composite wrappers for applications requiring distributed resources. We demonstrate that our method is robust enough to discover standard as well as non-standard table structures in HTML documents. Thus our technique outperforms contemporary techniques used in systems such as XWrap and AutoWrapper. We discuss our technique in the context of our *PickUp* system that exploits the theoretical developments presented in this dissertation and emerges as an elegant automatic wrapper generation system.

4.1 Introduction

The importance of automatic wrapper generation for biological database interoperation has been well recognized in recent research by a number of leading research

groups [25, 42, 54, 84]. The massive efforts in database integration were motivated by several important factors including the need for large scale data analysis, distributed resource integration in post-genomic era, and to a certain extent by the conditions imposed by federal grant agencies such as NSF and NIH.

The need for automation can be justified in many different ways. Two major factors stand out – the lack of technical sophistication of the end-users of such databases, and by the ad hoc nature of the applications or queries run by them. At the application level, automation helps end-users to approach distributed resource integration on their own possibly with the aid of intelligent tools. These tools could generate wrappers in a stepwise fashion resolving any ambiguities along the way with the help of the user. It was shown in many research [45, 49, 54, 58] that in such a set up, an intelligent tool can perform well with minimal help from an average user when faced with system limitations in resolving ambiguities. It is our thesis that in a nearly homogeneous application domain, such as genomics, this assumption¹ holds true and is supported by experimental evidence.

At the technical level, automation can be justified as follows. First, manual wrapper generation is a tedious task and prone to error. Despite risks of error, manual wrapper generation is usually much more effective and reliable. But the cost is usually prohibitive. Many reliable techniques now exist for concept identification and matching in digital documents needed for wrapper induction. But the caveat is that there exists situations when all such techniques fail. Human intelligence can usually be applied to handle such situ-

¹The assumption being that automated and accurate wrapper generation with little or no help from the end-user is possible in homogeneous domains.

ations and improve the functionality of the autonomous system. Secondly, when ad hoc queries over distributed resources are concerned, integration itself becomes practically ad hoc, and in such situations manual integration or wrapper generation is impractical due to cost factors. So, a user assisted automatic integration and wrapper generation is highly preferred if high precision at a throw-away cost can be guaranteed.

Our goal in this chapter is twofold. First to demonstrate that automatic composite wrapper generation is feasible for homogeneous domains, and second, to present a procedure to identify table structures in HTML documents. We argue that table structure recognition is an important ingredient for the generation of any composite wrapper. In an attempt to convince the reader of the importance of our goals, let us consider an application that finds the *Homo sapiens* genes for the ovarian tissues from The Cancer Genome Anatomy Project (CGAP) database at NCBI for some cancer research. A search for such genes using the *gene finder* tool at CGAP site may display the table shown in figure 4.1.

If we are interested in picking up all the corresponding gene sequences from the GenBank, we would need to collect all the **Sequence IDs** in the third column, and read the sequences from the database. If, however, we are required to collect all the sequences corresponding to the full-length MGC clones for each of the genes, we need to follow the links in the **CGAP Gene Info** column to visit the subsequent pages to collect the accession numbers, and then the sequences from the GenBank database. Figure 4.2 shows one such table for the first sequence *NM_001614* in figure 4.1. Readers may have noticed that the figure 4.2 contains yet another table along with other information.

The Cancer Genome Anatomy Project

CGAP HOW TO: [Genes](#) [Chromosomes](#) [Tissues](#) [SAGE Genie](#) [Pathways](#) [Tools](#)

Gene List

GeneFinder Results For: Hs; ovary;
UniGene Build: Hs.159/Mm.120

[\[Text\]](#) [\[Clones\]](#) [\[NCI60\]](#) [\[SAGE\]](#)

Highlight common aspects of the listed genes

Common View: Cyt Loc, Pathways, Ontology, Motifs, SNPs

Displaying 1 thru 7 of 7 items

| Symbol | Name | Sequence ID | CGAP Gene Info |
|--------|--|------------------------|---------------------------|
| ACTG1 | actin, gamma 1 | NM_001614 | Gene Info |
| B2M | beta-2-microglobulin | NM_004048 | Gene Info |
| CTRB1 | chymotrypsinogen B1 | NM_001906 | Gene Info |
| PKM2 | pyruvate kinase, muscle | NM_002654 | Gene Info |
| RPLP0 | ribosomal protein, large, P0 | NM_001002 NM_053275 | Gene Info |
| SILV | silver homolog (mouse) | NM_006928 | Gene Info |
| ST13 | suppression of tumorigenicity 13 (colon carcinoma) (Hsp70 interacting protein) | NM_003932 | Gene Info |

Figure 4.1 The set of Homo sapiens ovarian tissue genes found in NCBI CGAP Database

Purchase CGAP Reagents

- [CGAP cDNA Clones](#)
- [cDNA Libraries](#)

Related Links

- [JAX MGI](#)
- [MedMiner](#)
- [MGC](#)
- [Molecular Profiling](#)

Quick Links:

- [NCI Home](#)
- [NCICB Home](#)
- [NCBI Home](#)

NATIONAL CANCER INSTITUTE

- [Two-dimensional array displays](#) (similar expression pattern in NCI60 or SAGE data)

Cytogenetic Location (from UniGene)

Cytogenetic Location: 17q25 [Mitelman Breakpoint Data](#)

Full-Length MGC Clones for This Gene

| IMAGE Id | Status | Accession | GenBank Def Line |
|-------------------------|-------------|--------------------------|---------------------------|
| 2819345 | Full Length | BC000292 | actin, gamma 1 |
| 2820489 | Full Length | BC001920 | actin, gamma 1 |
| 2962953 | Full Length | BC007442 | actin, gamma 1 |
| 3939552 | Full Length | BC009848 | actin, gamma 1 |
| 3940861 | Full Length | BC010999 | Similar to actin, gamma 1 |
| 4554944 | Full Length | BC012050 | Similar to actin, gamma 1 |
| 4842665 | Full Length | BC015695 | actin, gamma 1 |
| 4855463 | Full Length | BC015005 | actin, gamma 1 |
| 4856294 | Full Length | BC015779 | actin, gamma 1 |
| 4869628 | Full Length | BC018774 | actin, gamma 1 |
| 3533309 | Incomplete | BC004223 | |

Figure 4.2 MGC clones for the gene NM_001614 shown in figure 4.1

In both the pages, identification and manipulation of the table structures are the key operations. Consequently any wrapper generation algorithm must be equipped to do just that. It is important to remark here that the apparent simplicity of the HTML documents² is really deceptive. It is well known that tables in general can be captured in many different ways in HTML, possibly using non-standard techniques. Hence identification of HTML record structures in general is not a trivial problem, as it will be evident from the discussions in this chapter.

This chapter introduces a fully automatic wrapper generation system, called *PickUp*, for HTML documents based on table structure identification. We organize the presentation of the techniques involved in table structure identification as follows. In section 4.2, we introduce the so-called hierarchical repeated structure identification technique for table structure identification. In this section we discuss relevant theoretical backgrounds and a method that exploits these concepts for automated wrapper generation. This section also reports our automatic wrapper induction system called *PickUp*. We also present two examples to show the effectiveness of our method over leading techniques. We then finally summarize in section 4.5.

²It is important to note here that most Web documents, including all the documents at NCBI site, are still written in HTML for many practical purposes. Hence it is important that algorithms manipulating such pages are capable of handling HTML peculiarities. We also believe that translation of HTML to XML does not completely remove the problems associated with HTML documents and hence is not the solution to this problem.

4.2 Hierarchical Repeated Structure Recognition

In this section we introduce the so-called hierarchical repeated structure recognition (HRSR) method for automated table structure identification. The HRSR method is primarily based on the observation that records in table structures in Web documents share certain structural regularity. Intuitively, HRSR technique involves the identification of cells in a conceptual table, a reconstruction of rows or records from fragmented cells and the generation of a table model from a set of similar records. Cell identification is facilitated by the path expressions using the hyper text query language (HTQL) [19]. These path expressions are used also to model the table structures and the wrappers. In the following sections we will discuss each of these concepts in some details. However, at this point we refer the readers to [19] for an introduction to HTQL and its associated tag-tree data model³ on which our subsequent discussions are based.

4.2.1 Structural Relationships of HTML Elements in Tag-Tree Data Model

An HTML document in our tag-tree data model is a sequence of items. Items are of two types – tag items (HTML tags) and text items. As usual, tags items are of two types – *start tags* (or s-tags) and *end tags* (or e-tags). For any two s-tags s_1 (short form of $\langle s_1 \rangle$) and

³In the context of the discussions presented in section 4.2.1, it is important to remark that the fluidity of HTML documents and its tag structures do not break down our data model. Hence it is not necessary, as we tacitly assume in this chapter, that HTML documents are somewhat regular – have matching start and end tags.

s_2 in any document d , s_2 is said to be *reachable* from s_1 , written $s_1 \rightarrow s_2$, if s_2 follows s_1 and the corresponding e-tag of s_1 follows s_2 in d .

Thus, by definition, the s-tags in d naturally induce a partial order on all the s-tags of d – i.e., for any three s-tags a, b, c , $a \rightarrow a$, $a \rightarrow b \wedge b \rightarrow c \Rightarrow a \rightarrow c$ and $a \rightarrow b \wedge b \rightarrow a \Rightarrow a = b$ hold. Given two s-tags a and c , $path(a, c)$ holds if $a \rightarrow c$ holds, and $path(a, c) = a.b_1 \dots b_n.c$ when $a \rightarrow b_1, b_1 \rightarrow b_2, \dots, b_{n-1} \rightarrow b_n, b_n \rightarrow c$.

For any two distinct s-tags s_1 and s_2 such that $s_1 \rightarrow s_2$ holds, s_1 induces a natural indexing on all s such that $s_1 \rightarrow s$ and $s_2 = s$. For all such s , $s_1.sk$ represents the k th s after s_1 such that $s_1 \rightarrow s$ holds. Notice that $a.c$ always holds when $a.b.c$ holds. For example, when $a.b.c.d.c.e$ holds, $a.c1$ means $a.b.c$ and $a.c2$ means $a.b.c.d.c$ while $a.c$ or $a.c0$ means all c such that $a \rightarrow c$ holds.

Furthermore, it is assumed that every HTML document begins with a special *null* tag, or n-tag. So, for any s-tag s , $null \rightarrow s$ always holds. Since the n-tag is a special tag, for any s-tag s , $path(s) \equiv path(null, s)$. As such, $a.b.c.d.c.e \equiv a1.b1.c1.d1.c1.e1$.

A path p is called complete if all its tags are indexed, i.e., $a1.b1.c1.d1.c1.e1$ is a complete path. The tags in a complete path with the indices are called qualified tags. The trailing qualified tag of a complete path is called the target tag. For any complete path p , $seq(p)$ denotes the sequence of tags in p . i.e., for an arbitrary $p = a1.b1.c1.d1.c1.e1$, $seq(p) = a.b.c.d.c.e$.

Let p_1, \dots, p_n be a set C of complete paths with identical trailing tags (may differ in indices). Let 2^C be all possible subsets of C such that for each $c \in 2^C$, and for all

$p_1, p_2 \in c$, $seq(p_1) = seq(p_2)$ and all $p \in c$ share a common suffix σ_c of qualified tags.

Each such c is called a family of *structurally similar* paths.

For a given pair of sets $c_1, c_2 \in 2^C$, c_1 is preferred over c_2 if the length of every path in c_1 is longer than the length of the paths in c_2 , and the length of σ_{c_1} is longer than length of σ_{c_2} . A set $c \in 2^C$ is *most preferred* if there exists no c' such that c' is preferred over c .

However, for any $c \in 2^C$, c is called a *related* family of paths if all $p \in c$ share a common prefix π_c of qualified tags. For a given pair of sets of related family paths $c_1, c_2 \in 2^C$, c_1 is preferred over c_2 if the length of π_{c_1} is longer than length of π_{c_2} . A set $c \in 2^C$ of related family paths is *most preferred* if there exists no c' such that c' is preferred over c .

The problem of automatic table structure identification is thus stated as the identification of the largest set S of most preferred structurally similar paths and the largest set R of most preferred related family paths in document d at item offset⁴ δ such that the length of the common prefix of paths in $R \cup S$ is maximized.

4.2.2 Discovery of Regular Structures

A major assumption we exploit in our system is that record structures in a document share a structural pattern, and thus all paths to its components (columns) share a common prefix.

To discover such common substructures, we exploit the idea of repeated pattern mining in

⁴The offset δ is used to discover target table structure using the tag preceding immediately before the offset. Thus using a zero offset is tantamount to the discovery of structure appearing anywhere in the document (from the start to end of document). We will take up the discussion on the importance of δ again in section 4.2.2.1.

gene sequences [44]. The difference is that we now mine HTML tag sequence instead of gene sequence. The entire process is explained in the next few sections.

4.2.2.1 Target Structure Recognition

In a given document d , it is possible that it includes several tables. Every such table structure (not necessarily represented using HTML table tag) will have repeated tag structure or sub-trees (captured in the form of paths). With the help of a variant of SeqMiner tool [44], we discover all repeated patterns and their repeat count in the tag trees of d . Then we rank all the repeated patterns using a simple function R . The function R returns an integer value for every repeated pattern given the length of the repeat sequence and the frequency of repeat. Ranking of repeats (and thus identification of record structure) is somewhat tricky since unintended identification is possible. For this dissertation we assume that ranking of a pattern must be high if it has high frequency (many records) and long pattern length (many attributes or columns). So we use the function $R(n, m) = n * m$ where n is the length of the repeated pattern and m is the frequency of repeat. It is possible that several patterns will be assigned the same ranking using this formula. So, we choose the first pattern as a candidate for the record structure since each one of them is equally valid.

For example, from Figure 4.2 we find the pattern

```
</a></td><td></td></tr><tr><td>
```

has a best repeat rank with $n = 7$, $m = 14$, and $R(n, m) = 98$. It appears in the HTML fragments such as:

</td><td>actin, gamma 1 </td></tr> <tr valign=top><td>

We may find another candidate pattern

<td></td></tr><tr><td>

with $n = 5$, $m = 15$ and $R(n, m) = 75$, which ranks lower than the previous pattern.

The algorithm to find repeat patterns can be simplified as the algorithm *Pattern-Miner*:

Function *Pattern-Miner*

Input: A sequence of tags $t_{1...N}$

Output:

A ranked pattern set $P\{\langle \text{Pattern } p, \text{Length } n, \text{Repeat } m, R(m, n) \rangle\}$

Begin

Initialize P to empty;

Let $\wp(l)$ denote the set of distinct patterns of length l in t ;

Let $S(p)$ denotes the set of positions a pattern p appears in t ;

Let $S^{\wp(l)}$ denote the set $\{S(p) | p \in \wp(l)\}$;

Compute $S^{\wp(1)}$ from distinct tags (pattern of length 1) in t ;

For $k = 2$ **to** N

Derive $S^{\wp(k)}$ from $S^{\wp(k-1)}$ by looking ahead one tag from positions in each $S(p) \in S^{\wp(k-1)}$;

For each $S(p) \in S^{\wp(k)}$

Let $m = \text{count}(S(p))$;

Add $\langle p, k, m, R(k, m) \rangle$ to P ;

Return P ;

End

The major computation of this algorithm is in the derivation of $S^{\wp(k)}$ from $S^{\wp(k-1)}$, which cost $O(N)$, and the **For** loop. Intuitively, the algorithm runs in time complexity of $O(N^2)$. However, pattern positions $S(p)$ can be removed earlier when there is no repeats. As a result, the size of $S^{\wp(k)}$ decrease exponentially with k and the **For** loop can be terminated earlier when set $S^{\wp(k)}$ becomes empty. Exploiting this idea, we have developed an improved version of the algorithm that runs in $O(N \log N)$ of both time and space complexities.

In situations where a wrong table structure becomes the candidate, PickUp allows marking a target table on the HTML document to disambiguate the identification. This marking of an element, a row, or a table virtually marks the first tag t that precedes the marked element in d . This t is now used as a candidate for target repeated pattern identification where t is the leading tag. We then use the techniques described in the following section to generate path expressions for t and move on to generate a record structure involving t . If t falls within the boundary of the target table structure, our method is guaranteed to find the intended structure. However, it is easy to notice that in an unmarked document, the special *null* tag is assumed to be the leading tag of a candidate repeat pattern outlined above. It implies that marking is well defined and robust, and its use or omission does not break down our procedure.

4.2.2.2 HTQL Path Expression Generation

Once the repeated tag pattern is determined (or the leading tag of a candidate pattern is known through marking), we need to generate the HTQL expressions for the purpose of wrapper generation. The leading tag of the repeated pattern is used to generate all possible path expressions in HTQL. Details of HTQL language and a procedure for the efficient generation of *complete path* expressions can be found in [19].

For example, the HTML fragment of BC000292, as the best repeat pattern found in the previous section, has HTQL path expressions of:

```
<TABLE>3.<TR>1.<TD>2.<table>8.<tr>2. <td>3.<a>1
```

<TABLE>3.<TR>1.<table>8.<tr>2.<td>3. <a>1

These paths represent the leading `` tag of the BC000292 HTML fragment. The paths are ranked by criteria that are discussed in [19], and only the top k paths are fed into the next HRSR steps as candidate paths, where k is a threshold that can be set by programs. We use $k = 2$ in our experiments.

The complete paths generation cost time $O((p - 1)^{p-1})$ [19], where p is the maximum height of the tag-tree representation of the document. However, we can limit the height for the paths generation, and most valuable paths can be generated with a height limitation of 7 and can be generated within a second.

4.2.2.3 Structural Relationship Recognition

Notice that the repeated pattern identification alone is not a guarantee that a correct table structure will be discovered. So, we use the leading tag t of the repeated pattern to generate all possible HTQL path expressions in which t appears as the trailing tag. By doing so, we include more candidates for the table structure by inflating the set of path expressions. However, the table structure identified by the repeat pattern is still included in the set. The set of paths obtained at this stage is used to compute the set S of most preferred structurally similar paths as explained in section 4.2.1. And finally, the set S is used to compute the set R of most preferred related family paths. This is accomplished by iteratively generating the path expressions as explained in section 4.2.2.2 for each of the tags in the repeated pattern, and maximizing the most preferred relationship with respect

to $S \cup R$. The combination of S and R in essence captures the most significant table structure in d .

For each candidate path p generated from repeated patterns, we then search for the set of most preferred structural similar paths. We do so by producing variations of an index of path p to expand similar items (each represents a similar path). An example variation of the fifth index in candidate path:

`<TABLE>3.<TR>1.<TD>2.<table>8.<tr>2. <td>3.<a>1`

generates path:

`<TABLE>3.<TR>1.<TD>2.<table>8.<tr>0. <td>3.<a>1.`

The varied path expression expands 15 similar items in the document. Another variation in the sixth index as

`<TABLE>3.<TR>1.<TD>2.<table>8.<tr>2. <td>0.<a>1`

expands 2 similar items. The varied path expression with a maximum expansion is selected for table generation, and is called a *feature path*. The prefix in the candidate path before the variation index is called the *feature prefix*. The item wrapped by the feature prefix is called a *feature item*. In the above example, the feature prefix is

`<TABLE>3.<TR>1.<TD>2.<table>8.<tr>2,`

and the feature item is an HTML fragment representing the row including 'BC000292'. A feature path and a feature item will be found from each candidate path.

Algorithm *Feature-Path-Identification* finds the feature path from a candidate path.

Function *Feature-Path-Identification*

Input: A candidate path p , document d

Output: Feature path f_p , feature prefix f_x

Begin

Denote p as a path sequence of $t_1i_1.t_2i_2.\dots.t_ni_n$,

where t_k is a tag name, and i_k is the index ($k = 1 \dots n$);

Initialize x and m to 0;

For $k = n$ **down to** 1 **do**

Variate p by replacing i_k to 0 (means *any index*) as p' ;

c = number of items in d wrapped by p' ;

If $c > x$ **then**

$x = c$;

$m = k$;

Replace i_m to 0 in p as f_p ;

Let f_x be the path sequence of $t_1i_1.\dots.t_mi_m$;

Return f_p, f_x ;

End

From each feature path, we then search for the most preferred related family paths.

We do this by enumerating sub-items of the feature item that include text content or is a leaf-tag or is an open-tag (a tag that has no corresponding end-tag) and generate a set of related suffixes. This is described in the algorithm *Related-Suffixes-Generation*.

Function *Related-Suffixes-Generation*

Input: A sequence of items $I_{1\dots N}$

Output: A set R of HTQL expressions

Begin

Initialize variables R ;

Match begin-tag and end-tags in I ;

For $i=1$ to N

If I_i has no corresponding end-tag

or I_i is a leaf-tag

or I_{i+1} is a text-item **then**

Let r = an HTQL expression generated for item I_{i+1} ;

$R = r \cup R$;

If number of items in R is over a threshold λ **then**

Break For loop;

Return R ;

End

In this algorithm, matching of begin-tag and end-tags costs at most $O(m^2)$, where m is the number of sub-items in the feature item. The generation of HTQL costs $O((p-1)^{p-1})$, where p is the maximum depth of the tag-tree representation of the feature item. The algorithm costs $O(m^2 + m(p-1)^{p-1})$. The values of m and p are typically very small (the feature item is a small segment in the original document). We further limit the generation of related items to be less than a threshold λ (30 in our experiment) since larger related item sets tend to include more useless information. As a result the running time of this algorithm is small.

From the above feature item, the algorithm generates HTQLs extracting items “2819345”, “Full Length”, “BC000292”, and “actin, gamma 1”.

Combining the feature prefix and the related suffixes and allowing the variation index to vary, we get a wrapper that wraps a table of similar and related items, where the prefix wraps similar tuples and the suffixes wrap related fields.

From this structural relationship recognition phase, we have a set of candidate table wrappers. The candidate table wrappers will be fed into the next phase for model generation and table evaluation.

4.2.2.4 Model Generation and Validation

The most preferred sets S and R do not actually account for missing columns in rows, or missing columns in tables especially when rows are spread over a document space. This is a consequence of the assumption that the table structures need not be represented

using HTML table tags. We essentially allow any regularity to be identified as a table structure. This means that two candidate rows may differ with respect to a column. So, we generate a model of the table by conservatively generalizing a row to fit an intended table structure. This is achieved by adding a column to a row only if that column's repeat frequency is more than half of the row frequency. We also accept the type of a column as the type of the majority of the row types in that column. Finally, we generate a composite HTQL expression to generate the wrapper. Once the wrapper is generated, it is validated by recreating each of the elements in the table structure in d by the unit expressions in the wrapper. If the test is positive for all the elements, the validation is considered successful.

We devise two models to describe the quality of a table wrapper – a *tuple similarity model (TSM)* and a *field similarity model (FSM)*. By TSM, we measure the similarity of tuples by their null fields as ψ . By FSM, we measure the similarity of tuple fields by their hyper-tags as ω . The ψ and ω are then combined with the data size of the table to have an overall evaluation S_t for a table wrapper.

Each model is described in a pattern concept. A *pattern* is nothing but a sequence of symbols. We are considering symbols from the alphabet of ASCII characters, and a pattern is represented by a string. For example, 'TNTTN' and 'TTTN' are two patterns. Two patterns can be *aligned* for matching characters. For example, patterns 'TNTTN'

and 'TTTN' can be aligned as:

$$T \quad N \quad T \quad T \quad N$$

$$T \quad - \quad T \quad T \quad N$$

, where character '-' represents a *gap*. Each aligned position is in one of the four states: match(M), insert(I), delete(D) and replace(R), and each state is associated with a predefined cost (called the *indel cost*). The

sum of indel costs at each alignment position is the cost of the alignment. Given a set of indel costs for the indel states (called a *cost matrix*), a dynamic programming algorithm can align two patterns for the minimal cost in time complexity of $O(nm)$, where n and m are lengths of the two patterns. A more detailed discussion of pattern alignment and the dynamic programming algorithm can be found in [29]. We now assume readers are familiar with these concepts.

We define *similarity* $\text{Sim}(a, b)$ of two patterns a and b as

$$\text{Sim}(a, b) = 1 - \text{Cost}(a, b)/2l, \quad (4.1)$$

where $\text{Cost}(a, b)$ is the minimal alignment cost with the cost matrix of $\{M=0, D=1, I=1, R=2\}$ and l is the length of the resulting alignment.

In the TSM model, each tuple field is described in two states: null(N) and text(T). The null state describes an empty field and the text state describes a non-empty field. A tuple is described in a *null pattern* consisting of a sequence of field states. The TSM ψ is computed from the formula:

$$\psi = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{Sim}(E_i, E_j)}{n(n-1)/2}, \quad (4.2)$$

where n is the number of tuples in the table and $E_k (k = 1 \dots n)$ is the null pattern of the k th tuple. From this formula, ψ is essentially the average similarity of null patterns of a table.

In the FSM model, items in the HTML representation of a field value are described in three states: start-tag(T), text(D) and end-tag(e). A field is described in a *tag pattern* consisting of the sequence of tag states. The FSM ω is computed from formula:

$$\omega_k = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{Sim}(F_{ki}, F_{kj})}{n(n-1)/2}, (k = 1 \dots m) \quad (4.3)$$

where m is the number of fields in a tuple, n is the number of tuples, and $F_{kl}(k = 1 \dots m, l = 1 \dots n)$ is the tag pattern of the k th field of the l th tuple.

From this formula, ω is essentially the average similarity of tag patterns of a table field.

Finally, we need to compute an overall score S_t to evaluate a given table. Let r be the null ratio of fields in the table (number of null fields divided by the number of total fields), and $d_k(k = 1 \dots m)$ be the sum of data lengths (text length in state D) of the k th field of all tuples. The value of S_t is computed as:

$$S_t = (mn - n + m)(1 - r)\psi^2 \sum_{k=1}^m d_k \omega_k^2, \quad (4.4)$$

where m is the number of columns of a table and n is the number of tuples of the table. This formula represents the information wrapped by a table wrapper. S_t captures the table size, data size and the empty fields of a table. The mn term reflects the size of the table and the $(mn - n + m)$ term favors columns over tuple numbers. In a table with a large number of empty fields, the $(1 - r)\psi^2$ term modifies the size of the table. The value of d_k reflects the text data size in a table. In a table with little text content, $d_k \omega_k^2$ reflects the modified data size.

A wrapper with a maximal S_t score is considered the best table wrapper. Our experimental results show a good performance of this formula for table evaluation.

4.3 Experiment of HRSR

4.3.1 *Experimental Results*

We ran our experiment on a Dell Dimension 8200 desktop computer with a Pentium 4 processor 2Ghz CPU and 1G RAM. Examples were chosen from major biological databases that return a table of results. Table 4.1 shows the number of attributes and columns that were correctly wrapped by our automatic wrapper program, namely PickUp. Attributes were chosen from tags that include non-blank text or special tags such as image tags. Experiments show that PickUp has no problem recognizing the table content. The wrapper induction time is very fast - within half minute for most Web pages we tested. The wrapper execution time is the average time to execute the generated wrapper against a test page. Since there is no wrapper learning procedure involved, the execution time is even faster and usually within one second. The generated wrappers were validated against 20 other pages from the same website. The validation results are promising. For most of the Web sites, the wrapper can wrap other pages with 100% correct rate. However, for the Hybridoma Data Bank (HDB) only 80% correct rates were obtained. This is because our wrapper expression is sensitive to certain structural changes along the path we have selected. For example, when a <table> tag is chosen in the wrapper path prefix, a new table inserted immediately before the corresponding table in the original document will cause

the path index to be invalid. Our path selection algorithm has been designed to choose robust path prefixes that are insensitive to most data and structure changes. However, in cases of failure, we are still able to reinduce a correct wrapper from the changed documents and the structure sensitivity will not compromise the merit of the fast wrapper execution. In a robust environment, we can use our automatic wrapper maintenance technique that is discussed in chapter V to get a 100% correct rate for the failed pages.

4.3.2 Comparison to Related Work

We compare our *Pickup* wrapper generation program to popular wrapper programs including XWrap Elite [45, 58], BYU tools [32], Lixto [9], RoadRunner [24], STALKER [6, 49] and WEIN [52, 50] in Table 4.2. Data for the table was collected from reported results of each program. Since each wrapper method has a different experimental purpose and uses different test data sets, it is hard to have a direct and quantitative comparison. As a result, we only compare them qualitatively considering a set of criteria that most affect the wrapper construction time. Criteria we use to compare them include ontology creation time, data labeling time, sample collection time and overall wrapper generation time.

Among these programs, only the BYU tool uses an ontology guiding approach. The advantage is once an ontology has been created, data can be wrapped according to the ontology quickly and with high precision. The disadvantage in using an ontology is that it is difficult to construct. It may take days to construct an ontology, which is still limited by the seen examples.

Table 4.1 Automatic wrapper generation experiment results

| Web site | Attri- butes | Rows | Test Pages | Induction Time (sec) | Execution Time (sec) | Valid- ation |
|-------------------------|-----------------|------|---------------|-------------------------|-------------------------|-----------------|
| NCBI protein search | 5 | 20 | 5 | 0.953 | 0.078 | 100% |
| NCBI genome search | 5 | 20 | 5 | 0.703 | 0.047 | 100% |
| NCBI book search | 10 | 26 | 5 | 6.297 | 0.110 | 100% |
| NCBI Locus Link | 6 | 50 | 5 | 15.594 | 0.438 | 100% |
| BioMedNet | 15 | 20 | 5 | 8.625 | 0.360 | 100% |
| Protein Data Bank(PDB) | 20 | 20 | 5 | 4.062 | 0.484 | 100% |
| SWISSPROT | 4 | 30 | 5 | 2.844 | 0.047 | 100% |
| Hybridoma Data Bank | 7 | 10 | 5 | 8.328 | 0.016 | 80% |
| Tumor Gene Database | 4 | 87 | 5 | 2.297 | 0.046 | 100% |
| Small RNA database | 4 | 20 | 5 | 0.375 | 0.016 | 100% |
| UM-BBD enzyme | 4 | 379 | 5 | 17.468 | 0.422 | 100% |
| Protein families (Pfam) | 2 | 20 | 5 | 0.594 | 0.016 | 100% |

STALKER and WEIN are similar in that they both take a wrapper induction approach and uses machine learning techniques to train wrappers from a set of examples. This approach requires labeling a reasonable number of examples first, which need to be done manually and is time consuming.

RoadRunner also follows a wrapper induction approach. However, it compares two samples at a time and can induce a wrapper automatically without any data labeling. As a result, it can generate wrappers quickly in a few seconds. However, RoadRunner can only be used when a set of examples similar in page layout is available.

All of the above programs need to collect a set of training examples. Lixto, XWrap and PickUp instead can generate wrappers from a single sample page. Lixto is still a semi-automatic wrapper generation tool that requires manually labeling interesting data from the sample page. It needs a considerable amount of time for a user to master the wrapper tools first and the wrapper labeling procedure is usually time consuming.

XWrap and PickUp are similar in that they both can automatically generate wrappers from a sample page and the wrapper generation procedure is highly automated. XWrap uses five heuristics to generate wrappers automatically. However, it still needs a user to inspect the generated wrappers and refine data manually from the results. On the contrary, PickUp generates a table wrapper fully automatically without any human interference. In case of wrapper generation for an arbitrary data item, PickUp only needs a mark in a web page and rest of the wrapper generation procedure is fully automated.

Table 4.2 Comparison of wrapper construction work

| Tools | Time | Samples Required | Labels Required? | Ontology Required? | Code Produced |
|-----------------|----------------------------------|-------------------------------------|---|--------------------------------------|--------------------------|
| Pickup | 0-20 sec | 1 | No | No | HTQL ex- pressions |
| XWrap Elite | 8-20 min | 1 | No | No | Java pro- grams |
| BYU | Days for ontology creation | Samples for ontology creation | No | Use appli- cation on- tologies | - |
| Lixto | 8 hours | 1 | Manual labeling | No | Elog programs |
| Road- Runner | 0-5 sec | ≥ 2 | No | No | Regular ex- pressions |
| STALKER | Several hours | 1-9 | Manual labeling for each sample page | No | STALKER rules |
| WEIN | Several hours | 2-44 | A hand-coded labeling wrapper for each source | No | Wrapper classes |

4.3.3 Examples

We now present two examples to demonstrate the capabilities and the effectiveness of the PickUp system. Figure 4.2 shows a CGAP page containing a table and figure 4.3 shows the table generated by a wrapper induced by PickUp from the page in figure 4.2. As shown in figure 4.2, the table structure is represented using table tags and is somewhat obvious and PickUp had no trouble correctly wrapping this table.

| COLUMN1 | COLUMN2 | COLUMN3 | COLUMN4 |
|-------------------------|-------------|--------------------------|-------------------------------------|
| 2819345 | Full Length | BC000292 | actin, gamma 1 |
| 2820489 | Full Length | BC001920 | actin, gamma 1 |
| 2962953 | Full Length | BC007442 | actin, gamma 1 |
| 3939552 | Full Length | BC009848 | actin, gamma 1 |
| 3940861 | Full Length | BC010999 | Similar to actin, gamma 1 |
| 4554944 | Full Length | BC012050 | Similar to actin, gamma 1 |
| 4842665 | Full Length | BC015695 | actin, gamma 1 |
| 4855463 | Full Length | BC015005 | actin, gamma 1 |
| 4856294 | Full Length | BC015779 | actin, gamma 1 |
| 4869628 | Full Length | BC018774 | actin, gamma 1 |
| 3533309 | Incomplete | BC004223 | |
| 3538275 | Incomplete | BC017450 | |
| 3897065 | Incomplete | BC009544 | |
| 4053240 | Incomplete | BC010417 | Similar to actin, beta, cytoplasmic |
| 4109821 | Incomplete | BC023548 | Similar to actin, beta |

Attributes=4, tuples=15, total=60

Figure 4.3 The structural table from figure 4.2 recreated by the wrapper

The next example (in figures 4.4 and 4.5) demonstrates that PickUp is capable of effectively identifying table structures even when the structure is not represented using HTML table tags. This example shows that PickUp is more versatile than XWrap, Island Wrapper and AutoWrapper in identifying and wrapping table structures.



Figure 4.4 A list of books from NCBI represented using loose table structures without table tags





| COLUMN1 | COLUMN2 | COLUMN3 | COLUMN4 | COLUMN5 | COLUMN6 |
|---|-----------|---------------------------------------|---|---|--|
|  | 695 items | Cancer Medicine. | Bast, Robert C. ; Kufe, Donald W. ; Pollock, Raphael E. ; Weichselbaum, Ralph R. ; Holland, James F. ; Frei, Emil, editors. | Canada: BC Decker Inc. ; c2000 | BC Decker Inc. |
|  | 577 items | Medical Microbiology. | Baron, Samuel, editor | Galveston: University of Texas Medical Branch. ; c1996. | University of Texas Medical Branch |
|  | 218 items | Molecular Cell Biology. | Lodish, Harvey; Berk, Arnold; Zipursky, S. Lawrence; Matsudaira, Paul; Baltimore, David; Darnell, James E. | New York: W H Freeman & Co. ; c2000. | W H Freeman & Co |
|  | 207 items | Molecular Biology of the Cell. | Alberts, Bruce; Bray, Dennis; Lewis, Julian; Raff, Martin; Roberts, Keith; Watson, James | New York and London: Garland Publishing. ; c1994 | Garland Publishing |

Figure 4.5 A faithful recreation of the books table in figure 4.4 by the wrapper generated by PickUp.

4.4 Composite Wrappers

Our over-arching goal for PickUp is to be able to generate a wrapper that can iterate over the rows of a table and navigate to the next page pointed to by one of the cells so that users do not need to simulate the iteration manually. This can be achieved in two principal ways. Hyperlinks can be identified automatically if all the links are of interest, or by letting the user mark a column on the generated wrapper output that contains a hyper link in order to avoid the generation of uninteresting wrappers. Once identified, the same technology as applied in the current page may be applied to identify tables in the subsequent pages. The iteration may be simulated by memorizing the column navigation and driving the generation of secondary documents using one wrapper inside a loop. However, if the hyper links point to documents having multiple types of documents, appropriate wrappers need be applied for each of the hyper links. In this case the composite wrapper definition must make provisions for customization and use a tree-like execution – recursively or iteratively. However, the idea of composite wrapper generation is still in its infancy and remains as an item for future research.

A *table wrapper* is referred to as a wrapper generated from a single page based on the wrapper generation algorithm we have discussed so far. We typically use W^d to denote a wrapper generated from a page d , and $W(d)$ to denote the wrapping result of a wrapper for page d . A *composite wrapper* is defined recursively from table wrappers. If W^d is a table wrapper generated from page d and $W_i (i = 1..m)$ are wrappers, then $W^d \{ \cup_{i=1..m} ([i]W_i) \}$ is a composite wrapper, which means wrapping a page with W^d and for each i th column

of the result tuples, navigating the hyperlink (if it exists) and wrapping the target page with W_i , where $i = 1..m$. Algorithm *Composite-Wrapper* generates a composite wrapper from a page d with a maximum depth l .

Procedure *Composite-Wrapper*

Input: document d , depth l

Output: a composite wrapper $W^{d,l}$

Begin

Generate a table wrapper W^d from document d ;

If $l = 1$ **then**

$W^{d,l} = W^d$;

return $W^{d,l}$;

For each column i of the result tuples $W^d(d)$

If column i has hyperlinks **then**

Navigate to a sample page d_i from the hyperlinks;

$W_i = \mathbf{Composite-Wrapper}(d_i, l - 1)$;

$W^{d,l} = W^d\{\cup_i([i]W_i)\}$ including each non-empty W_i ;

Return $W^{d,l}$

End

4.5 Summary

In this dissertation we explored a new technique for automatic wrapper generation for table structured data in semi-structured documents. The approach relies on several key ingredients – a formal model of path expression based characterization of tables, repeated pattern discovery and reconstruction of table structures from fragmented path expressions using a preference relationship. We have demonstrated through experiments and examples that our method is more effective and robust than the leading systems that generate wrappers for table structures in automatic ways.

It is possible to show that the wrappers generated by PickUp are more suitable for incremental maintenance. This is not true for most other systems. We claim that it is possible to detect changes in target documents and tweak the existing wrapper to adapt to the new document. This process may be facilitated by maintaining the set of candidate wrappers we generate during the wrapper induction phase.

The PickUp system will be available online soon. Our future research includes improving the heuristic to correctly identify a target structure without users having to mark it and to implement the idea of composite wrapper generation.

CHAPTER V

AUTOMATIC WRAPPER MAINTENANCE

Because wrappers are being extensively used in information management systems to extract Web resources, the maintenance of a large number of wrappers becomes an important issue. The maintenance task is difficult in that Web resources tend to change autonomously while little information can be used to detect the change. Continuous usage of the wrappers, on the contrary, requires the wrappers to function consistently and be transparent to changes from data sources. The absence of schema information in the Web source prevents wrappers from reestablishing the correlation between old and new data. Furthermore, the variation in vocabularies and data values further obstructs the successful maintenance of Web wrappers. When the number of sites to be integrated is substantially large, manual maintenance of their wrappers become tedious and error prone. This dissertation presents automatic wrapper maintenance techniques for Web data resolving the above difficulties in a systematic manner. Based on our infrastructure, a wrapper can detect and adapt to changes of data automatically and wrapper's rules are maintained incrementally. As a result, a large wrapper repository can be managed and maintained without effort. We envision our automatic wrapper technique to be a vital component for seman-

tic Web technology, where agents need to create and adapt wrappers to data sources in a dynamic and ad hoc manner with little human interference.

5.1 Introduction

Due to the vast amount of Web data accruing every day, heterogeneous Web database integration has attracted much attention for years. A wrapper-mediator architecture has been widely adopted in the integration systems. An important factor that affects the scalability of integration systems is in the wrapper construction part. This is because that wrapper is an irreplaceable communication component between an integration system and the semi-structured Web data, where little descriptive information about the data is present. Furthermore the continuous change of data in contents and structures invalidates much manual effort trying to maintain the schema information proliferated from the base data during the initial integration stage. As a result, most of the current Web databases integration systems inherently place a heavy burden both on the wrapper developers and the system keepers. In the semantic Web context, such a burden is undesirable since ad hoc integration of a large number of websites is critical for Web agents to survive. The purpose of this chapter is to show that both the construction and maintenance of wrappers can be fully automated. Consequently, an integration system can easily scale up while the validity of the Web wrappers is maintained.

Web databases integration can be found in every discipline nowadays. Both scientific and commercial institutes are moving to publish data through the Web. On one hand, the

publication of information from the Web increases the exposure opportunity of these organizations. On the other spectrum the vast amount of information on the Web quickly buries valuable individual organization information given the lack of effective data management systems. A Web databases integration system is a management system to facilitate the organization, integration and query of distributed online Web information. The idea comes from the success of traditional database systems for the organization of in-house file data. A Web databases system essentially manages the structural view of the semi-structured Web data and provides transparent access to the Internet data for upper layer information agents.

A wrapper is a set of data transformation rules that can be used to extract structured data from semi-structured Web documents. It realizes the Web data access requirement of a Web database system. The most challenging and interesting category of wrapper is the HTML data wrapper. An HTML wrapper is interesting in that HTML data are still a main data media in the Web. Although there is a trend in the XML standardization of the Web, the unparalleled simplicity and flexibility of HTML data still attracts most non-professional users. The lack of XML standardization makes the total replacement of existing HTML data difficult. The ready integration of existing scientific Web data such as the NCBI genome databases, the PDB protein database and thousands of others, is the main force driving our HTML wrapper research. HTML wrappers are challenging in that unlike XML data, where some metadata information and structural constraints are maintained, the HTML data has no such metadata information and structural constraints. Neverthe-

less, many graphical characteristics of HTML data, such as Web forms, Web tables, and so on, may provide invaluable message for the automatic discovery of information in the HTML data that has never been exploited. This chapter is focused on the wrapper technique for HTML data sources. However we believe our technique is general enough for XML data also.

Consider the NCBI nucleotide Web database. (Here a Web database refers to a collection of Web data published at a website, in contrast to the Web database integration system we have discussed so far.) Nucleotide information is searchable from the website with keywords and the results are formatted in HTML and displayed to users. The search of nucleotides related to keyword 'SARS' has an example HTML of Figure 5.1. We would like to access information, such as the Accession ID, the nucleotide description and the GI number of each result entry in a structural fashion as in Figure 5.2, where each tuple represents an entry of the search result. The transformation from the semi-structural format to the structural format can be done by a wrapper designed for the NCBI website. Once the wrapper is designed, further access and query of related Web pages, which may come from other keyword searches or a future document with the same keyword search can be straightforward.

Keeping a wrapper is important in contrast to retrieving and extracting Web pages at runtime in the following aspects. First, runtime-extracted data tend to be more unpredictable and unreliable than the rule-extracted data with wrappers. The predictability and reliability make it easy for an integration system to cope with dynamic Web data. Sec-

The screenshot shows the NCBI Nucleotide search interface. At the top, there is a search bar with 'Nucleotide' selected and 'SARS' entered. Below the search bar, there are options for 'Limits', 'Preview/Index', 'History', 'Clipboard', and 'Details'. The search results are displayed in a table with columns for 'Display', 'Summary', 'Show', 'Send to', and 'Page'. The results are numbered 1 through 4, each with a checkbox, a link to the accession number, and a brief description of the sequence.

| Display | Summary | Show | Send to | Page |
|--------------------------|---|------|---------|---------|
| Items 1-20 of 639 | | 20 | Text | 1 of 32 |
| <input type="checkbox"/> | 1: AY297028 SARS coronavirus ZJ01, complete genome gi 30910859 gb AY297028.1 [30910859] | | Links | |
| <input type="checkbox"/> | 2: AY286402 SARS coronavirus Taiwan JC-2003 RNA-directed RNA polymerase (pol) gene, partial cds gi 30909287 gb AY286402.1 [30909287] | | Links | |
| <input type="checkbox"/> | 3: AJ563470 Crassostrea gigas partial mRNA for Binding protein 2 like protein (fkbp2-1 gene) gi 30842643 emb AJ563470.1 CGI563470[30842643] | | Links | |
| <input type="checkbox"/> | 4: NC_004718 SARS coronavirus, complete genome gi 30271926 ref NC_004718.3 [30271926] | | Links | |

Figure 5.1 The NCBI nucleotide database search results

| | | | | |
|----|---------------------------|-----------------------|---|--|
| 1: | AY297028 | Links | SARS coronavirus ZJ01, complete genome | gi 30910859 gb AY297028.1 [30910859] |
| 2: | AY286402 | Links | SARS coronavirus Taiwan JC-2003 RNA-directed RNA polymerase (pol) gene, partial cds | gi 30909287 gb AY286402.1 [30909287] |
| 3: | AJ563470 | Links | Crassostrea gigas partial mRNA for Binding protein 2 like protein (fkbp2-1 gene) | gi 30842643 emb AJ563470.1 CGI563470[30842643] |
| 4: | NC_004718 | Links | SARS coronavirus, complete genome | gi 30271926 ref NC_004718.3 [30271926] |

Figure 5.2 A structural view of the NCBI nucleotide database

ond, a wrapper may be automatically generated with a special algorithm or manually encoded for a specific purpose. An application using wrappers does not need to consider the method used to construct a wrapper. Such independency and transparency is desirable for an integration system to adapt to heterogeneous Web data. Third, with a wrapper, general features of the source data can be learned, encoded and employed to recognize a future changed document, where a run-time extraction may fail to extract the correct data without such features. However, the above merits of a Web wrapper come with the expense of maintaining the wrappers continuously.

An integration database needs to keep a large set of wrappers for a large number of Web data sources. A problem with the current wrapper techniques is that the development of the wrapper rules is generally onerous and it is even more difficult to maintain the rules in a timely fashion. In the dynamic Web environment, the frequent changes of data content and page layouts of the data sources will quickly outstrip any manual effort to maintain a large wrapper set. Thus automatic maintenance of wrappers becomes an important issue faced by database researchers prior to resolving the data integration problem.

Automatic wrapper maintenance is difficult in the following respects. First, semi-structured data provides little hint on changes. The change may be detected when the data is wrapped and an error result is returned. This may be an empty result or a wrong set of data. Consequently, change detection is the first task for wrapper maintenance. Second, there is no schema information within the Web data and the correlation of old data content and new content is difficult to develop. In commercial Web sites, advertising contents

are frequently inserted into a Web page, which causes the interesting contents to move around in the Web data. The organization of the Web pages may also be changed on an irregular basis. However, the terminologies and values of the target information are changing over time, which makes the recognition of the target information difficult when the Web page changes. Third, to regenerate data extraction rules, the wrapper language should be powerful enough to re-induce itself. Automatic wrapper maintenance typically needs to utilize information captured in the old wrapper rules and regenerates substitution rules in adaption to the changed data. The generation of the new wrapper is carried out by the old wrapper during the wrapper execution time.

5.2 Automatic Wrapper Maintenance

Our automatic wrapper maintenance system (named WM) is an extension of the PickUp system [19, 20] developed by Chen, Jamil, and Wang, where techniques are proposed to detect repeated table structures and generate data extraction rules from HTML pages in a fully automatic manner. With the database integration background, we will focus our discussion of wrapper maintenance techniques on relational tables wrappers. Since a table wrapper can be seen as a superset of a single item wrapper (a table containing only one item) our technique can be applied to the maintenance of single item wrappers as well, although some special care may be needed in the single item case.

We are more interested in maintenance of wrappers in a fully automatic fashion, which means that no any human intervention is required. When a failure of a wrapper is detected,

which may result in an empty or wrong result data set, the wrapper repair and improvement algorithms are invoked to adapt to the change. During the maintenance stage, cheap adaptation algorithms are invoked in case of minor data change and expensive repair algorithms are only invoked after the cheap solutions fail. As a result, our maintenance technique is efficient. Another distinguishing feature of our maintenance technique is that it does not require past sample data to retrain the wrapper. Instead, it maintains wrappers incrementally, requiring only the existing wrapper and the current Web page and improving the wrapper when changes are detected. This prominent feature allows our wrappers to work autonomously and be independent of the system that uses the wrapper. The wrapper rules are encoded in an XML file, which allow the wrapper to be stored and transmitted on the Web conveniently.

Since our wrapper technique is domain independent, the change of information content in the Web page generally does not affect the effectiveness of the wrapper. However, if the structure of the Web page is substantially changed, our wrapper may stop functioning correctly. Therefore, the main task of wrapper maintenance is to automatically adjust the wrapper rules when the structural change of Web pages affect the extraction of target data. Two classes of changes are addressed in this chapter. First, the placement of target data is changed. This commonly occurs when irrelevant information is inserted or deleted from the source pages. For example, in Amazon.com, advertising information is frequently inserted or deleted in the source page, which may cause a wrapper to work improperly. Second, the record structure of the data of interest is changed. For example, the record

structure of some flight agent websites such as the Expedia.com website are frequently changed.

To detect the changes of data and verify the validation of a wrapper, we use a set of pattern construction and recognition techniques to compare different data. Briefly speaking, we construct a set of syntax patterns that work at different levels of granularity based on the sample data. When new data is wrapped, the corresponding syntactic patterns are compared so that the difference in granularities can be evaluated and the validity of the wrapper can be verified. These syntax patterns allow our maintenance system to be alert to changes in different syntax levels. Each pattern is associated with an assessment value of how well the data should fit with the pattern. By this method, the stringency of pattern rules can be adjusted. Although Lerman, Minton and Knoblock [56] also use a pattern recognition method for change detection, the pattern construction and comparison algorithms are different from ours. Our usage of various sequence alignment algorithms is expected to have a better assessment of the change. Our HTML level syntax pattern is a higher-level syntax pattern and is not present in [56].

The rest of this chapter will proceed as follows. We will first set up a formal model for a table wrapper and introduce the wrapper maintenance problem. Then we will present a brief review of techniques used for automatic wrapper generation, which will be revisited frequently by our WM system. We will define several patterns as verification criteria for table wrappers. The algorithm for wrapper verification will be presented followed by a description of the algorithms used in our WM system to adjust wrappers to different types

of data changes. An experiment that was conducted and the experimental results will be described. Finally we will draw our conclusions and identify future work.

5.3 Formal Model of Table Wrapper and Wrapper Maintenance

5.3.1 Model of Semi-structured Data

A document D can be seen as a collection of items including *begin-tag items*, *end-tag items* and *text items*. A *tag-name* is associated with each begin-tag and end-tag. A pair of begin-tag and end-tag items with the intervening items constitutes a *hyper-tag item*. The *reachability* relationship is defined as the relationship between the hyper-tag item and the items that are within the begin-tag and end-tag items of the hyper-tag item. The reachability relationships and the items in a document constitute an item-graph, where the items are nodes and the reachability relationships are edges. A *path* is a navigation sequence in the item-graph that uniquely identifies an item. Two items are *similar* to each other if they have paths with a same sequence of tag-names. Two items are *related* to each other if they have paths sharing the same prefix.

5.3.2 Model of Table Wrapper

In a document, a set of items that are similar and related to each other, and naturally form a table, is called a *structural table*. Intuitively, an HTML table is a structural table, the rows in the table are similar to each other, and the items in a row are related to each other. The definition of the structural table is a more general concept than the HTML table. For

example, in Figure 5.1 the result entries are not formatted with an HTML table. However, since the structure of each item is organized similarly, it still meets the requirements of a structural table. In the previous chapter, we have shown that a structural table can be wrapped automatically.

In this dissertation, a *table wrapper* refers to a wrapper that wraps a structural table. Formally, a table wrapper is a transformation function $W: D \rightarrow R$, where D is an HTML document, and R is a structural table. In the structural table R , each related item set forms a *tuple*, and each similar item set forms a *column*. An item in R is also called a *field*. In WM, a table wrapper W is constructed as a composition of p and t , where p is the set of rules that define the common path prefix identifying the structural table, and t is the set of rules that define the fields in the structural table.

5.3.3 Model of Type, Schema and Criterion

Each column of a structural table conforms to a schema defined by types. We follow and extend the type definition for semi-structured data by Arasu and Garcia-Molina [4]. They define a *type* recursively as:

1. A *Basic Type* represents a string of tokens (such as a word or an HTML tag).
2. If T_1, \dots, T_n are types, then their ordered list $\langle T_1, \dots, T_n \rangle$ is also a type representing a *tuple constructor* of order n .
3. If T is a type, then $\{T\}$ is also a type representing a *set constructor*.

We instead define a type as:

1. A *Basic Type* T represents a boolean function that takes as input a string of tokens and returns a boolean value reporting whether the token string is in type T .

2. If T_1, \dots, T_n are types, then their ordered list $\langle T_1, \dots, T_n \rangle$ is also a type representing a *tuple constructor* of order n . A tuple type reports true if and only if each of its element types reports a true value.
3. If T is a type, then $\{T\}$ is also a type representing a *set constructor*. A set type reports true if and only if each of its element types reports a true value.

We define a *schema* differently as a boolean expression of types. A table column conforming to a schema is said to be constrained by each of the types involved in the schema.

Example. Consider three example basic types $T_1 = isalpha$, $T_2 = isdigit$ and $T_3 = nolower$ and a schema $S = \langle T_1, \{T_2\} \rangle \wedge T_3$. Type T_1 represents any single alphabetic character, type T_2 represents any single digit character and type T_3 represents any string that has no lower-case characters. The set type $\{T_2\}$ represents a sequence of digital characters and the tuple type $\langle T_1, \{T_2\} \rangle$ represents a string consisting of an alphabetic followed by a sequence of digits. The schema S represents a string consisting of an alphabetic followed by a sequence of digits but not consisting any lower-case characters. A string 'A12345' is an instance of schema S .

Further, we define a *criterion* as a boolean expression of schemas. A structural table conforming to a criterion is said to be *validated* by the criterion. Commonly we allow the schema of a table wrapper to evolve incrementally over time while keep a criterion in constant for validation purpose.

5.3.4 Problem Formulation

A class of wrappers W is *maintainable* for a sequence of documents $D = \{d_1, \dots, d_n\}$ if we can find a wrapper $w_i \in W$ to wraps document d_i into a relation that can be validated by a criterion C for any $i \in \{1 \dots n\}$. If there is a function M that transforms the wrapper w_t to w_{t+1} for any $t \in \{1 \dots n\}$ then we call function M a *wrapper maintenance function*.

The problem of *automatic wrapper maintenance* is to find a class of maintainable wrappers W and a wrapper maintenance function M for a sequence of documents $D = \{d_1, \dots, d_n\}$, so that, for each $i \in \{1 \dots n\}$, function M generates a wrapper $w_i \in W$ from $w_{i-1} \in W$ and w_i wraps document d_i to a relation that can be validated by a criterion C starting from a known wrapper $w_0 \in W$.

The documents to be wrapped are typically unknown a priori and the number of documents may be infinite. Sometimes it is difficult to maintain wrappers for every input document. Therefore we allow a portion of documents to be *invalid* while only considering *valid* documents. Valid documents are those that can be maintained by the maintenance function continuously. At a given time t , the ratio of valid documents v and the number of documents processed t , called the *validity ratio*, measures the performance of the wrapper maintenance function. Another possible interpretation of a high validity ratio is that the input documents are homogeneous in nature.

The basic approach we are taking is to construct a set of basic and composite types for semi-structured data and develop an automatic wrapper maintenance function. A maintenance function is something like a reverse of a wrapper function – to derive a new wrapper

from the data. Usually this reverse process is much more complex and time-consuming than directly wrapping the data by a wrapper. As a result, we consider different maintenance functions with different costs and combine them for efficiency purpose.

Data type and validation criterion are important factors affecting the effectiveness and efficiency of a maintenance algorithm. Not only can they validate input data, they can guide the maintenance function to derive new wrappers.

An important step in the types and criteria construction process is type learning. In a structural table, each column includes items structurally similar to each other. We expect that item content will also be similar. Given this expectation, we have devised algorithms to learn the common feature, called a pattern, from each column and use this common feature to construct types and a criterion to verify and reconstruct the wrapper.

The relationship of wrapper maintenance components is shown in Figure 5.3. The bi-directional relationships between the wrapper and data and the criterion and data enables the renewal of new wrappers upon data changes.

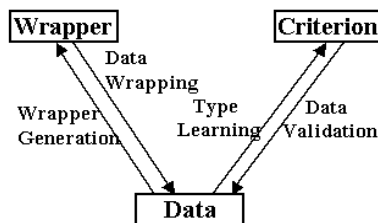


Figure 5.3 The relationship of automatic wrapper maintenance components

5.4 Automatic Table Wrapper Generation

Our automatic wrapper maintenance technique depends on the so-called automatic wrapper generation method. First, before a table wrapper exists, we need to generate a table wrapper automatically from a given HTML document. Second, once the source is changed and a new table position is detected, we need to regenerate the table wrapper. This section gives a brief review of the wrapper generation algorithm. The detailed algorithms can be found in [19, 20].

5.4.1 Target Structure Recognition

It is possible that a given document d includes several tables. Every such table structure (not necessarily represented using HTML table tags) will have repeated tag structures or sub-trees (captured in the form of paths). With the help of a variant of the SeqMiner tool [44], we discover all repeated patterns and their repeat count in the tag trees of d . Then we rank all the repeated patterns using a simple function R . The function R returns an integer value for every repeated pattern given the length of the repeat sequence and the frequency of repeat. Ranking of repeats (and thus identification of record structure) is somewhat tricky since unintended identification is possible. For this dissertation we assume that ranking of a pattern must be high if it has high frequency (many records) and long pattern length (many attributes or columns). So we use the function $R(n, m) = n * m$ where n is the length of the repeated pattern and m is the frequency of repeat. It is possible that several patterns will be assigned the same ranking using this formula. We choose the

top r ranked patterns, where r is a threshold one can change, as candidates that will be further evaluated by the next steps.

5.4.2 HTQL Path Expression Generation

Once the repeated tag pattern is determined, we need to generate the HTQL expressions for the purpose of wrapper generation. The leading tag of the repeated pattern is used to generate all possible path expressions in HTQL. A detailed algorithm to generate all possible paths can be found in [19]. The purpose of this module is to generate HTQL path expressions given a position s of an HTML page. The complete paths generation algorithm costs time $O((h - 1)^{h-1})$, where h is the maximum height of the tag-tree representation of the document. However, we can limit the height for the paths generation, and most valuable paths can be generated with a height limitation of 7, and can be generated in seconds. The generated paths will serve as prefixes for possible table wrappers.

5.4.3 Structural Relationship Recognition

Since each repeated pattern may be involved in a table wrapper, we need to automatically select a table that best represents the content of the HTML page. Such a table is called a maximal structural view (MSV) of the page. The purpose of this module is to identify a repeated pattern that leads to a table wrapper for MSV. We do this by generating tables from available candidate paths and evaluating candidate tables by an evaluation function. Specifically, each candidate path is placed into a table wrapper by selecting a prefix of

the path as the table prefix, and enumerating atomic and non-empty child nodes under the node represented by the prefix as the table suffixes. The resulting table wrappers are evaluated by the number of items wrapped by the wrapper, the similarity of table tuples and the similarity of table fields.

5.4.4 Performance Analysis of Table Wrappers Generation

The experiments with automatic wrapper generation are promising. For most of the Web sites with structural table content, for example, data retrieved from NCBI genome and protein databases, Protein Data Bank(PDB), BioMetNet and SWISSPROT, the table wrapper can wrap pages with 100% correctness. However, there are websites where the table wrapper generated from one page cannot correctly wrap other pages. For example, a table wrapper for weather report from www.weather.com can only correctly wrap 60% of other test pages and the wrapper for Hybridoma Data Bank (HDB) has a 80% success rate. This is because there are certain tables missing in some of the pages. We will resolve this problem in the rest of this chapter. With our maintenance technique, we will get a 100% correct rate for these failure pages.

5.5 Pattern Construction

We use two levels of patterns for the representation of data in an HTML page: HTML syntax pattern and text syntax pattern. The HTML syntax pattern describes the HTML tag

sequence in an HTML fragment. For example, the field of ‘AY297028’ in Figure 5.2 is an HTML fragment of:

```
<A href='query.fcgi?list_uids=30910859'> AY297028 </A>
```

This fragment can be described as an <A> tag, followed by a piece of text and an end-tag of . In WM, this description is represented in a hyper-pattern-string (HP) of “ADa”, where the ‘A’ is an HP-char stands for the <A> tag, the ‘D’ HP-char stands for a piece of text and the ‘a’ HP-char for the end-tag . We can see that fields of ‘AY286402’, ‘AJ563470’ and ‘NC_004718’ in Figure 5.2 can all be described in this HP. In contrast, the field of “SARS coronavirus ZJ01, complete genome” has a different HP of “D”. By representing such differences, the HP can be used to recognize different fields. The following table shows the HPs of each field in the first tuple in Figure 5.2.

Table 5.1 The hyper-pattern-strings (HPs) for the first tuple in Figure 5.2

| Field | HP |
|--|-----|
| 1: | BDb |
| AY297028 | ADa |
| Links | ADa |
| SARS coronavirus ZJ01, complete genome | D |
| gi 30910859 gb AY297028.1 [30910859] | D |

Each HTML hyper tag is assigned a corresponding HP-char. Tags that are considered significant for field recognition have special HP-chars, such as the ‘A’ and ‘B’ HP-chars in the above table, since we consider the <A> tag and the tag to be important in discrimination of HTML fragments. Other significant tags in the current WM include the <I>, <U>, and <Image> tags. A general HP-char ‘T’ is used to represent any tags that have no a stand-alone HP-char, such as the
 tag.

HP is a relatively high-level representation of data. It is sensitive to the organization of hyper tags and is effective for structure recognition of complex HTML text. However, for relatively plain text where there is little hyper-tag information, HP has its limitations. For example, in Table 5.5, HP cannot discriminate the ‘AY297028’ field from the ‘Links:’ field, both of which can be represented in HP as ‘ADa’. Similarly the fourth field and fifth field have the same HP of ‘D’ and cannot be recognized. A text-pattern-string (TP) is another class of pattern that works at the word level. A TP describes a sequence of syntax words in a sentence. The TP ignores any tag information in data, and classifies syntax words into word, number, keyword, data, currency, time, symbol, etc. Each class is represented in a character (TP-char). For example, a TP-char ‘W’ represents a word, TP-char ‘N’ represents a number and ‘K’ represents a keyword. Symbols such as ‘:’, ‘;’ and ‘|’ are kept in their original form. For example, the ‘AY297028’ field in Figure 5.2 is recognized as a keyword, whose TP is ‘K’, and the TP for the ‘Links:’ field is ‘W:’. The difference in TP provides a way to discriminate the two fields.

Table 5.2 The text-pattern-strings (TPs) for the first tuple in Figure 5.2

| Field | TP |
|--|---------------|
| 1: | N: |
| AY297028 | K |
| Links | W |
| SARS coronavirus ZJ01, complete genome | KWKWW |
| gi 30910859 gb AY297028.1 [30910859] | W N W K.N [N] |

For a column of fields with different pattern string representations, a single consensus pattern string is constructed to represent the table column. For example, the first column in Figure 5.2 can be described as $\{\text{HP}=\text{'BDb'}, \text{TP}=\text{'N:'}\}$ since HP and TP describe the first field of each tuple in the table. For fields with more variations, where each tuple may have a different HP or TP, a more complex algorithm is needed. For example, for the fourth column in Figure 5.2, the first tuple has a TP of 'KWK,WW,' the second tuple has a TP of 'KWWK-NKWKW...', and the third tuple has a TP that is different from the first two. To resolve the differences, a multiple sequence alignment (MSA) procedure can be used to construct a consensus pattern from all possible pattern strings. MSA is extensively used in the bio-information community for consensus DNA sequence construction in phylogeny analysis. Finding an optimal MSA is NP-hard. Sub-optimal algorithms are typically used in applications. WM adopts a progressive MSA algorithm. It has the computational complexity of $O(N^3L^2)$ time, where N is the number of patterns to be aligned and L is the

maximal length of the patterns. Since we will limit the length and number of the pattern for consensus pattern construction, the actual cost is bounded to a fixed amount of time.

Algorithm 1 describes the progressive MSA algorithm used for consensus pattern construction. The central part of the algorithm is the use of the dynamic programming algorithm to align a pair of patterns. The most similar patterns with the minimal cost are merged into a partial consensus pattern. The partial consensus pattern replaces the original patterns in the patterns set and progressively builds the final consensus pattern. An example consensus TP for the fourth column in Figure 5.2 is ‘WWWWKWWKWW’, where only the first ten TP-char of each TP is aligned.

Algorithm 1. *Consensus pattern construction*
Input: patterns $P[1..n]$
Output: consensus pattern S
Begin
 Initialize weights $W[1..n]$ with a value 1 for each pattern
While there are more than one patterns in P
 For each pair of pattern $P[i], P[j]$ in P
 Set alignment indel cost for dynamic algorithm:
 $Cost[match] = 0$
 $Cost[insert] = W[i]$
 $Cost[delete] = W[j]$
 $Cost[replace] = \min(W[i], W[j])$
 Use dynamic algorithm
 to compute the alignment cost of $P[i]$ and $P[j]$
 Find the pair $P[m_1]$ and $P[m_2]$ with a minimal cost
 Construct the partial consensus T of $P[m_1]$ and $P[m_2]$:
 Let s_1, s_2 be alignment strings of $P[m_1]$ and $P[m_2]$
 For each position k in s_1
 Set $T[k]$ as the non-indel char in
 $\{s_1[k], s_2[k]\}$ with a larger pattern weight.
 Assign $P[m_1]$ as the partial consensus pattern T
 Add weight $W[m_2]$ to $W[m_1]$
 Remove $P[m_2]$ from P
 Let S be the final pattern in P
End

5.6 Wrapper Verification

The purpose of wrapper verification is to evaluate the quality of a wrapper W for the wrapping of a given document D with a set of verification rules Q . The central part of the verification technique is the creation of verification rules and the method to evaluate the wrapper validity with the rules. Lerman, Minton and Knoblock [56] use rules of content patterns, average number of tuples-per-page, mean number of tokens in the example, mean token length and density of alphabetic, numeric, HTML-tag and punctuation types

to evaluate wrappers. The method used to evaluate the wrapper validity is the goodness of fit method [67], which can test whether two distributions are the same. However, since the variables are not independent, the method tends to overestimate the test statistic. As a result, their experiments included only the starting content pattern rule.

We use three categories of rules, the HP, TP and content lengths (CL) rules. A simple conjunction test of each rule is applied to verify a wrapper; that is, the wrapper is considered valid for a document only if all the rule conditions are met. The formula is:

$$V(Q, x) = V(Q_{HP}, x) \wedge V(Q_{TP}, x) \wedge V(Q_{CL}, x), \quad (5.1)$$

where x is a set of data for the verification, Q is the verification rules, Q_{HP} is the HP rule, Q_{TP} is the TP rule, Q_{CL} is the content length rule, and $Q = Q_{HP} \cup Q_{TP} \cup Q_{CL}$.

Q_{HP} is defined as a 4-tuple $Q_{HP} = \langle P_{HP}, \delta, \varepsilon, \omega \rangle$ where P_{HP} is the consensus HP build from examples, δ is the average alignment cost of example fields with the consensus HP, ε is the cost variation threshold for valid data, and the ω is the number of examples that support the rule. For example, the second field in Figure 5.2 has a Q_{HP} of $\langle \text{'ADa'}, 0.00, 20\%, 20 \rangle$, which means that the consensus HP for the fields is 'ADa', the average cost in the example is 0.00, the threshold of the cost variation threshold is 20% and the rule is built from 20 examples fields. Given a set of fields x whose consensus HP is $HP(x)$, the verification function is the following:

$$V(Q_{HP}, x) = \{\text{alignncost}(P_{HP}, HP(x)) < \delta + \varepsilon * \text{length}(P_{HP})\}, \quad (5.2)$$

where $Q_{HP} = \langle P_{HP}, \delta, \varepsilon, \omega \rangle$.

For example, in the above example of $Q_{HP} = \langle \text{'ADa'}, 0.00, 20\%, 20 \rangle$, if a wrapping result x has consensus $HP(x)$ of 'BDb', then the wrapper is invalid since

$$\begin{aligned} & V(Q_{HP}, HP(x)) \\ &= \{ \text{align-cost}(\text{'ADa'}, \text{'BDb'}) < 0.00 + 20\% * \text{length}(\text{'ADa'}) \} \\ &= \{ 2 < 20\% * 3 \} \\ &= \text{false} \end{aligned}$$

Similarly Q_{TP} is defined as a 4-tuple $Q_{TP} = \langle P_{TP}, \delta, \varepsilon, \omega \rangle$ and the verification function is defined as:

$$V(Q_{TP}, x) = \{ \text{aligncost}(P_{TP}, TP(x)) < \delta + \varepsilon * \text{length}(P_{TP}) \}, \quad (5.3)$$

where P_{TP} is the consensus TP of examples and $TP(x)$ is the consensus TP of fields x . For example, in the fourth field in Figure 5.2, if the TP rule is $Q_{TP} = \langle \text{'WWWWKWWKWW'}, 4.85, 20\%, 20 \rangle$, then the first field "SARS coronavirus ZJ01, complete genome" can be validated by

$$\begin{aligned} & V(Q_{TP}) \\ &= \{ \text{align-cost}(\text{'WWWWKWWKWW'}, \text{'KWKWW'}) < 4.85 + 20\% * 10 \} \\ &= \{ 5 < 4.85 + 2 \} \\ &= \text{true} \end{aligned}$$

Q_{CL} is defined as a 4-tuple of $\langle L_{max}, L_{min}, L_{avg}, \varepsilon, \omega \rangle$, where L_{max} is the maximal length item in the example, L_{min} is the minimal length, L_{avg} is the average length, ε is

the variation ratio and ω is the number of examples that support the rule. The verification function $V(Q_{CL}, x)$ is defined as:

$$V(Q_{CL}, x) = \{L(x) - L_{avg} < (1 + \varepsilon)(L_{max} - L_{min})\}. \quad (5.4)$$

Notice all of our verification functions are arbitrarily defined. Applications of the wrapper can choose their own wrapper verification function. The support ω of each rule is not used in our current verification functions. However, users can use the support information to gain confidence about the validation. The simple variation ratio ε provides an intuitive way for users to adjust the tolerance of variation rules. Wrapper rules are tested against a table field. For a document with multiple fields, if the number of fields that cannot be verified exceeds a certain threshold, the wrapper is judged to fail for the document. In our WM implementation, the threshold is set as 20%.

Rules are maintained in an XML file. Figure 5.4 shows an example of rules for the second and fourth column in Figure 5.2.

```
<WrapperField fieldName="COLUMN2">
  <LengthConsensus MaxLen="143" MinLen="142" AverageLen="142" Support="20"/>
  <HyperConsensus ConStr="ADa" Cost="0.000000" Support="20"/>
  <TextConsensus ConStr="K" Cost="0.000000" Support="20"/>
</WrapperField>
<WrapperField fieldName="COLUMN4">
  <LengthConsensus MaxLen="83" MinLen="33" AverageLen="51" Support="20"/>
  <HyperConsensus ConStr="D" Cost="0.000000" Support="20"/>
  <TextConsensus ConStr="WWWKWKWW" Cost="4.850000" Support="20"/>
</WrapperField>
```

Figure 5.4 Wrapper verification rules in XML

5.7 Wrapper Maintenance Algorithms

5.7.1 *Wrapper Adjustment with Candidate Paths*

Once a change is detected, the problem is to adjust the wrapper quickly to the change. With the PickUp automatic wrapper generation tool, a table wrapper is automatically composed with a path prefix p and a schema suffix t by analyzing the structure of the document. The path prefix p represents the best candidate path that can target the structural table for the table wrapper in a document. It is selected from a set of candidate paths with some evaluation function and is expected to be insensitive to most content and structural change in the document. For example, the table wrapper for Figure 5.2 can have candidate path prefixes ' $\langle DL \rangle$ ' and ' $\langle P \rangle 2. \langle DL \rangle$ '. The first prefix ' $\langle DL \rangle$ ' is insensitive to any tags other than a ' $\langle DL \rangle$ ' tag that may be inserted or deleted from the document, while the second prefix ' $\langle P \rangle 2. \langle DL \rangle$ ' is insensitive to any irrelevant $\langle DL \rangle$ tags that may be inserted or deleted before the second $\langle P \rangle$ tag. However, each of the prefixes has its limitation and will fail upon certain changes. For example, irrelevant $\langle DL \rangle$ tags that are inserted before the structural table may cause the first path prefix ' $\langle DL \rangle$ ' to fail and the removal of the a $\langle P \rangle$ tag in the document may cause the path prefix ' $\langle P \rangle 2. \langle DL \rangle$ ' to fail.

The most common situation where a table wrapper becomes invalid is when the placement of the structural table in a document changes, and the structural table remains unchanged. This situation is frequent when advertising information is inserted or removed

from the document on an irregular basis. One solution for such changes is to use candidate path prefixes that are insensitive to the change.

Since querying with a wrapper is cheaper than reconstructing a wrapper by orders of magnitude, this solution can avoid costly wrapper reconstruction procedures and quickly retarget the structural table. The wrapper rules will not be modified when a candidate path is validated. Thus occasional changes of a document will not affect the wrapper in general.

5.7.2 Re-targeting the Table

Not all movement of a structural table can be adjusted with the help of candidate paths. When there are significant portions of data inserted or deleted from the table, or the Web page on which the structural table resides incurs a major restructuring, candidate paths may not be able to adapt to the changes. For example, when the stock quote information is moved from the top of the page to the bottom of the page, candidate paths may fail to target the correct table. In this case, the schema prefix needs to be regenerated.

The first step for the regeneration of the schema prefix is to identify the position of the target table. The re-targeting task is difficult when the Web document is complex and multiple tables of different contents co-exist on the same page. Since our WM technique does not depend on a predefined application domain, and data in the Web page may change independently, it is impractical to understand the information in each structural table and then correlate the new structural tables with the old ones. We are therefore taking another approach - memorizing some 'signatures' in the original structural table. The idea is to

find the repeating patterns in the original structural table, and use the repeating pattern to recognize the target table in a new document.

For example, the first entry of the search result in Figure 5.1 , as shown in Figure 5.5, has a continuous sequence of tags “<dl><dt><table><tr><td><input> ...”, which is also present in the other entries. We expect this sequence also to be present in the new documents. With a local alignment of the signature sequence with the tags sequence of the new document, we can find out the most likely position in the new document that is similar to the signature sequence.

An algorithm to search for a similar subsequence from a long sequence is the local sequence alignment (LSA) algorithm [29]. Local alignment is typically used in gene sequence analysis to determine if a segment of gene sequence is a subsequence of another longer sequence. The most commonly used algorithm is a dynamic programming algorithm. However, it is different from the dynamic programming algorithm used to align two complete sequences (the global sequence alignment, GSA) [29]. A dynamic programming algorithm constructs a score matrix stepwise based on a given cost matrix. Each score in the matrix represents the best alignment cost of a pair of positions of the two sequences. The difference between the LSA and GSA is in determination of the overall score of the alignment and the trace back algorithm to get the alignment. GSA uses the score at the end of the alignment sequences as the overall alignment score and traces back from that end position. LSA, in contrast, searches for the best score in the score matrix and traces back from that position. The best score position is considered the most similar position of

the two sequences in our WM signature sequence finding. We will assume the reader is familiar with the dynamic programming algorithm, and the details of the algorithms will not be further discussed in this dissertation.

5.7.3 Regenerate Wrapper Fields

When the record structure of a structural table changes, the wrapper may fail to verify all fields of the table. When the majority of the fields are verified, we infer that the table has incurred minor changes in record structure and that the unverified fields have been moved inside the record structure. Therefore, it is the WM's task to find new wrapper rules for the restructured table fields. WM copes with this situation by regenerating all related fields from the record structure and matching the unverified fields with the newly generated fields. To re-generate all related fields, we simply build wrapper rules for every item that is enclosed with a tag within the first structural record tuple and test if the rules are valid for other tuples. The matching of old fields and new fields is done through the comparison of their consensus HPs and TPs. The formalism is listed below. With the formalism, the similarity of two fields is the product of their HP similarity, S_{HP} , and the TP similarity, S_{TP} . S_{HP} is defined as the ratio of the matched HP-chars to the average length of the two consensus HPs, and S_{TP} is the ratio of the matched TP-chars to the average length of the two consensus TPs.

$$S = S_{HP} * S_{TP},$$

$$S_{HP} = 1 - 2 * align-cost(P_{HP_1}, P_{HP_2}) / (L(P_{HP_1}) + L(P_{HP_2}))$$

$$S_{TP} = 1 - 2 * align-cost(P_{TP_1}, P_{TP_2}) / (L(P_{TP_1}) + L(P_{TP_2}))$$

The fields with a highest S value are considered the best matched fields. If the S score is higher than a threshold s (0.3 in our WM experiments), then the field wrapper rules and the verification rules are updated according to the new field. Once a target item is identified, the wrapper rules wrapping the item can be generated automatically with our automatic wrapper generation technique.

An advantage of wrapper repairing is that wrapper rules can be learned with imperfect samples. With more data being wrapped, a wrapper can encode more general information about the data source and become more robust. The benefit is that our wrapper generation algorithms requires only one Web page to learn the wrapper rules; this is important for the ad hoc integration of a large number of Web sources because the sample collection process is sometimes time-consuming. Another benefit is that our wrapper-learning algorithm does not require the samples to be distributed evenly in a domain. Characteristics of data are learned incrementally and occasional noise in the data can be adapted at runtime. This approach is practical considering the dynamic nature of Web data.

5.8 Experiment and Results

We developed a scheme to test the correctness of our WM technique that uses a wide variety of Web documents. The main purpose was to test the ability of WM to recognize a change, the effectiveness of table re-targeting, and the correctness of field regeneration.

The experimental data sources include NCBI nucleotide database search results, Amazon.com book search results, Weather.com weather reports, and Lycos.com stock quotes. These data source were selected for the following reasons: 1) they are popular experimental data for wrapper experiments; 2) interesting data is organized in a structural table format that is the main focus of our WM technique; 3) they come from different application domains; 4) the Web page and the table structure organizations are reasonably complex; and 5) results change with different keywords. Furthermore, each has some special features: the NCBI results are not organized in an HTML table; the Amazon results have a lot of related information encoded in a same page; the Weather results have a large number of recursive tables; and the Lycos results have fields that are similar. The automatically generated wrappers include the fields listed in Table 5.3.

Table 5.3 Data fields in the wrapper maintenance experiment

| Website | Example fields | Fields |
|---------|--|--------|
| NCBI | AccessionID, description, GI | 7 |
| Amazon | Book title, authors, cover type, image | 8 |
| Weather | Date, image, description, temperature | 6 |
| Lycos | Symbol, last price, change, volume | 17 |

Because there are relatively few major changes to the data sources, there will not be enough data for testing the correctness of our WM algorithms even if we spend a lot of

time monitoring and collecting all the changes. More importantly, it is difficult to prove the correctness of WM algorithms with a set of real changes since changes in one data source tend to be localized and may not reflect changes that may happen in other unstudied data sources. Therefore, we manually enumerated possible changes in a Web page and created artificial test sets based on the real example pages.

To conduct the following experiments, we set the field validation threshold to 80%, which means that if 80% of the fields are verified, the table wrapper is considered valid. On the contrary, if less than 80% of the fields can be verified, a set of wrapper maintenance algorithms will automatically be launched to improve the wrapper. If no improvement can verify 80% of the fields, then the wrapper is declared invalid for the new data. The modification of Web pages is done through the HTQL query language [19]. The experiments were conducted on a DELL Dimension 8200 computer with 500M memory and 2.0G cpu.

The first experiment tested the ability of WM to recognize a move of target tables and to re-target the table. For this purpose, we cut the target table from a source Web page, evenly selected n non-tag positions in the Web page, and inserted the table back into these positions. As a result, we had a test dataset including n changed documents for each test page. The original wrappers were then applied to these n changed documents. Table 5.4 shows the wrapping result. From these results, we can see that WM can recognize all the changes and can repair almost all the wrappers correctly.

The second experiment tested the ability of WM to recognize a change in a record structure and repair wrapper fields. For this purpose, we manually selected two columns

Table 5.4 Wrapper maintenance results for position changes

| Website | Change detected | Validity | Time(s) |
|---------|-----------------|----------|---------|
| NCBI | 100% | 100% | 0.45 |
| Amazon | 100% | 100% | 2.5 |
| Weather | 100% | 100% | 1.3 |
| Lycos | 100% | 99.2% | 10.0 |

in the structural table and swapped their positions in the Web page. In this way, we had a set of test documents different in the record structures by two fields. The original wrappers were applied to these changed documents, and the results are shown in Table 5.5. From the results, we can see that WM can recognize most of the changes. For the Weather data, all changes were detected and repaired. A portion of changes were not detected from the other datasets since the wrapper was still considered valid for the data. Most of the changed data can be wrapped after the maintenance procedure. Weather and Lycos wrappers were perfectly maintained for all the changes. Amazon and NCBI data had a lower maintenance rate. Still, over 85.7% of the changed documents remained valid after maintenance.

The third experiment combined effects of position movement and record structure changes. For this purpose, the table was cut out from the source page and two of the table columns were swapped by positions. The changed table was then inserted into the n non-tag positions that were evenly selected from the Web page. The resulting pages were

Table 5.5 Wrapper maintenance results for record structure changes

| Website | Change detected | Validity | Time(s) |
|---------|-----------------|----------|---------|
| NCBI | 85.7% | 95.2% | 0.78 |
| Amazon | 75% | 85.7% | 0.56 |
| Weather | 100% | 100% | 0.57 |
| Lycos | 86.1% | 100% | 1.2 |

different from the original page both in position and in record structure. The original wrappers were applied to these changed documents and the results are shown in Table 5.6. The maintenance results are very similar to those of the previous experiment; all the Weather data and Lycos data were determined to be valid after maintenance, and over 88% of the Amazon and NCBI data were valid after maintenance.

The maintenance time is commonly a few seconds. A position change typically costs more time to maintain than only a record structure change. This is because the retargeting table operation is much more costly than an adjustment with candidate paths or a regeneration of wrapper fields. The difference is significant for tables of homogeneous fields such as the Lycos data and minor for table fields of different structures such as the NCBI data. This demonstrates that the combination of different verification criteria is effective for efficient wrapper maintenance.

Table 5.6 Wrapper maintenance results for combined table position and record structure changes

| Website | Change detected | Validity | Time(s) |
|---------|-----------------|----------|---------|
| NCBI | 100% | 95.2% | 0.42 |
| Amazon | 100% | 88.8% | 1.0 |
| Weather | 100% | 100% | 1.54 |
| Lycos | 100% | 100% | 10.4 |

5.9 Conclusion and Future Work

We identify and develop a framework for the problem of automatic wrapper maintenance. This problem occurs in any large scale heterogeneous database integration system and has received relatively little attention. Our maintenance model as a combination of wrapper, criterion and data is novel and effective in maintaining a class of wrappers for a continuous set of semi-structured data. The resulting WM system does not need to remember past examples, and can wrap data with maintainable wrappers in a few seconds. We have developed a comprehensive method to verify the effectiveness of wrapper maintenance algorithms. We showed in experiments that multiple criteria can improve the efficiency of wrapper maintenance. We have employed a variety of sequence mining algorithms and demonstrated their effectiveness in semi-structured data analysis. As future work, we will apply our maintenance methodology for other wrapper classes, for example, single item wrappers.

```

<dl>
<dt>
  <table>
  <tr><td>
    <input name=uid type=checkbox value=30910859>
    <b>1: </b>
    <a href=query.fcgi?uids=30910859>AY297028</a>
  </td>
  <td align="right">
    <SPAN>
    <a CLASS="dblinks" href="javascript:Set()">Links</a>
    </SPAN>
  </td></tr>
  </table>
</dt>
<dd>
  SARS coronavirus ZJ01, complete genome <br>
  gi|30910859|gb|AY297028.1|[30910859]<br>
  <br>
</dd>
</dl>

```

Figure 5.5 An NCBI record entry in HTML

CHAPTER VI

THE *METEOROID* AD HOC INTEGRATION SYSTEM

As scientific data, tools, and services continue to populate the Web, scientific analyses must rely on data and tool resources scattered over the Web. These Web data are mainly designed for human navigational purposes. In order to exploit these Web data for more complex scientific computing tasks, it is important to make existing distributed Web resources not only accessible but also manageable to scientific people and applications. However, current Web data management approaches are brittle and unreliable due to the high degree of data heterogeneity, leading to an extremely high cost for scientists to collect and compute distributed data. We present an *ad hoc integration* approach to solve this problem. Ad hoc integration allows scientists to identify interesting data from the evolving Web pages independently, allows online heterogeneous data resources to be collected and manipulated conveniently, and allows distributed data computing to be designed and scheduled properly. Ad hoc integration provides a convenient and robust database environment for scientists to conduct Web data analysis tasks without resorting to complex computer infrastructures or consulting experts frequently. We have developed a *Meteoroid* ad hoc integration management system that meets these requirements. Meteoroid makes

it possible for individual scientists to easily utilize available Web resources in a database environment for daily data analysis purposes.

6.1 Introduction

As scientific data, tools and services continue to populate the Internet, scientific data analyses must rely more and more on online-available, distributed and heterogeneous data from the Web. These distributed Web data may contain invaluable information curated by domain experts, analyzed by specially designed software and tools, and sustained continuously by collaborating teams. However, it is still difficult for scientists who must excavate data from Web pages constantly, streamline data accesses to multiple sources, and combine data from distributed sources. This is because Web data usually lack fixed schemas and are typically changing constantly in both information content and data formats. Streamlining multiple data accesses typically requires transforming data from one format to another and non-trivial post-processing of collected data. Doing these tasks manually is not only time consuming but also error prone, posing a challenge for researchers in the field to fully exploit the wealth of information available on the Web.

On the other end of the spectrum, for practical reasons, biological scientists have been accustomed to simple and convenient personal storage systems and data formats such as Access databases, spreadsheets, plain text files and HTML files to record biological experimental data. Such data is exchanged frequently between labs, leading to an unmanageable number of data sources and formats. An increasing interest in computing in-house data

with online biological tools such as sequence alignment tools, BLAST searches, structure analysis tools, etc., makes this problem even more important. Biologists need a way to effectively collect, combine, and utilize the explosive data, tools, and services consistently for daily data analysis purposes.

Database management systems (DBMSs) have long been successful in providing a consistent data management platform for non-trivial data processing and for diverse data. Providing a reasonable data management framework for individual scientists to effectively integrate and query distributed information sources on demand has become another challenge for information and databases integration researchers. We identify the following challenges:

- Web data sources are mostly semi-structured and are evolving quickly, making it difficult for novice users to quickly identify desired data and record data accesses;
- A consistent approach to management of both traditional relational data sources and Web data sources is lacking. Web data sources have unique features such as semi-structured format, orientation to being used for navigation, dynamic generation, etc.;
- A full-fledged data integration framework able to succinctly express Web data, robustly maintain Web data, and unconventionally schedule, process, and mediate ad hoc integrated Web data has not been established.

As evidenced by prior research such as TSIMMIS [18], HEMMER [1], Information Manifold [57], Ariadne [49] and a number of other projects, a database approach to resolve data heterogeneity issues for scientific data interoperability appears to be the best approach. However, existing systems have not addressed the issues mentioned above. The following problems still exist:

1. Wrapper development is still challenging for users of existing systems. A *wrapper* is a basic component in a wrapper-mediator-based system to transform heterogeneous

data into a unified data model. Existing wrapper development methods either depend on an expensive training procedure with a considerable number of data samples or they require a wrapper expert to develop complex wrapper rules, posing a great burden to novice users. Moreover, the fast-evolving nature of Web sources quickly invalidates most manual or semi-manual efforts.

2. Existing systems treat Web sources as standard external views but fail to exploit Web-oriented properties Web-oriented such as navigational data exploration, pipelining data feeding, and session-dependent data retrieval. A scientific user also tends to add additional data dependency rules for data processing. A flexible framework to manage Web data sources to meet the variety of requirements is lacking.
3. Traditional query processing techniques rely heavily on join operators for query optimization purposes. These are mostly expensive operators and become awkward when dealing with Web sources with a large number of join attributes and a large number of participating views.

The inability to address these problems leads to a doubt about the reality and practicability of a general database approach to incorporating existing Web data. Most researchers believe a total reconstruction of the current Web structure may make the data integration work easier [13, 81]. In contrast to this circumventing solution, the approach in this dissertation provides an *ad hoc integration system* to tackle this problem directly.

An ad hoc integration system is a Web data management system that provides a traditional database management system environment for novice users to integrate and query distributed data. An overall goal of an ad hoc integration system is to integrate a large number of distributed Web sources and in-house databases to facilitate daily scientific data analysis works. An ad hoc integration system needs to close the boundary between end-users' data integration requirements and the heterogeneous data environment. As a result, it needs to develop a homogeneous data model to describe heterogeneous data, provide effective mechanisms for users to define and connect distributed data in real time,

manage integration queries that may incur unpredictable data behaviors, and meet complex data scheduling requirements. Under an ad hoc integration system, the Web becomes an extensible part of in-house databases.

6.2 A Motivating Example

Accompanying the complete sequencing of whole genomes of several species, a larger scale genetic information interpretation is underway. Prediction of biological functions, for example, now has more options such as single gene similarity searches, biological pathway comparisons, and genetic network references. Various biological data repositories have developed databases and Web tools facilitating genetic data analysis. For example, the BLAST tool at NCBI [83] allows searching of similar gene sequences to identify similar gene functions. KEGG (Kyoto Encyclopedia of Genes and Genomes) [12] developed integrated pathway/genome databases allowing prediction of metabolic pathways from genome sequences. LocusLink at NCBI [59] provides a single query interface connecting curated sequences with descriptive information including official nomenclature, sequence accessions, map locations, and related websites. It is, however, painstaking for biologists to actively utilize these distributed Web resources for even simple data analysis tasks given that more and more genomic data is available.

Consider a simple gene expression prediction scenario where a scientist wants to utilize the KEGG pathway database to predict from a set of gene IDs the relative degree that each

gene is contributing to a certain gene function. Since KEGG cannot recognize a gene ID to search for participating pathways, the user needs to go through a data discovery procedure:

1. Go to the LocusLink database to retrieve the description of each gene corresponding to the gene ID;
2. Use a special tool DBGET/gene [26] at the KEGG site to convert the gene description into an entry name recognizable by KEGG database.
3. Paste the entry name into a DBGET/LinkDB [27] interface to retrieve the participating pathways.
4. Connect each pathway to an XML/HTML file describing the relationships and reactions among genetic objects.
5. Analyze the pathway files to detect activating gene products.
6. The number of pathways each gene activates or inhibits reflects the relative degree of the gene affecting a gene function.

The above procedure needs to access various distributed data sources including:

A: The set of gene IDs for analysis in a local database;

B: The LocusLink database at the NCBI website returning a gene description recprd given a gene ID;

C: DBGET/gene at the KEGG site returning a set of entry names given a gene description;

D: DBGET/LinkDB at the KEGG site returning a set of pathway links from an entry name;

E: Pathway description files at KEGG in XML/HTML format.

In this example, dataset *A* is a structured database, datasets *B*, *C*, and *D* are Web tools, and dataset *E* is a set of semi-structured data files. In order to analyze the function of a

gene (for example, to identify a human cancer-activating gene), a biologist needs to go through the above data discovery procedure for each relevant gene from a human genome and apply certain filtering conditions at each discovery step such as limiting the entry name to start with an 'hsa:' string (representing human gene entries).

The above manual data discovery procedure may be frustrating when the number of genes to be analyzed is large. Unfortunately, it has become routine for many biologists to deal with such problems with even more complex operations. It would be ideal for the heterogeneous data sources to be managed under a uniform framework, queries to be recorded and submitted in a declarative way, and the complexity behind the physical data operations to be hidden from biologists. From the database point of view, the above query can be expressed simply in an algebraic expression as:

$$\pi_{\text{gene-id,pathway,type}} \sigma_{\text{entry like 'hsa: \%'} (A \bowtie B \bowtie C \bowtie D \bowtie \sigma_{\text{type='activate'}} E) \quad (6.1)$$

It is the purpose of this research to provide a foundation for direct application of high-level data manipulation operations – algebra operations – in the heterogeneous Web data environment. By using this approach, the Web becomes a synthetic extension of traditional database systems. Automated tools and visual interfaces provided with sound database management support can be easily developed.

6.3 Ad Hoc Data Integration Solution

We have developed an ad hoc data integration system, *Meteoroid* (MEthodology for ad hoc inTEgration of Online distributed heteROgeneous Internet Data), for biologists to

integrate experimental data with online resources. A navigation-oriented Web interface was designed to allow users to 'pick up' interesting data sources and attributes from the Web by employing our automated *PickUp* wrapper technique. 'Picking up' a piece of information usually means a user clicks on or selects a data item. For example, in order to retrieve gene descriptions from LocusLink, a user can 'pick up' the search form from the LocusLink website, enter a search term, submit the search, and 'pick up' the gene description from the results page. A user's behavior in this sequence of operations is captured by the Meteoroid system and is transformed into a declarative language called a data definition language (DDL). The generated DDL expression memorizes the user's navigation pattern and can be used to automatically extract data of interest from the same site for unexplored pages. Thus, it provides a means to record data collection procedures electronically.

Meteoroid allows a user to pick up a piece of data, a table of data or a Web form from a Web page. The 'picked up' information can then be accessed like a table (called *virtual table*) in a local database. Meteoroid maintains virtual tables automatically even when substantial changes have occurred in the original Web pages. The user accesses virtual tables through SQL queries or through a visual interface wizard Meteoroid provides. This means post processing for Web data can be easily done by applying conditions in an SQL query or by invoking data transformation functions in the query.

Connecting and streamlining data access from different sites is extremely easy and can be done with traditional table joins. A user usually does not need to be concerned about the

order in which multiple tables are joined. Meteoroid will schedule access to data sources according to the properties coming with each participating table. Input-output behavior of each data source will be utilized and streamlined according to the join conditions in a query.

To allow more complex data operations, such as scheduling access to Web sources with constraints and merging data values from different sources with mediation rules, Meteoroid employs a multi-layer table and view architecture. The multi-layer architecture facilitates flexible fusion and mediation of data at different level. Data mediation and query scheduling constraints are encoded in a *virtual view* and can be maintained incrementally. A virtual view can be seen as a cluster of related data sources for a scientific investigation. Virtual views are encoded in XML files and can be stored and transmitted easily.

Web data processing is slow, and complex scientific data mediation may consume even more time. It is desirable that available results be shown to users early during the execution of a query. Meteoroid devises a novel two-phase pipeline scheduling technique to expedite the integration of results for users. A user can make use of this information to terminate an unwanted query in an early stage.

Finally, Meteoroid provides a visual interface to facilitate ad hoc data integration. Using the visual interface, a user can define Web data sources in an ad hoc fashion with a few clicks, design integration queries from a wizard interface, and monitor query results in progress.

6.4 Declarative Support for Web Data

Declarative support for Web data is a unique contribution of this research. Other recent systems such as DiscoveryLink [42], which is based on Garlic [41] and DB2 [17] technologies, also claim to provide declarative definitions of wrappers for data sources. However, their declarative data definition can occur only after the heavy burden of wrapper development is completed, which usually requires several days as well as special programming skills. In contrast, our declarative language handles wrapper creation in the background, accompanying our fully automatic wrapper generation and maintenance technique.

Web data in Meteoroid are defined as remote user-defined functions (RUDFs) and managed as parameterized views. For form-based Web pages, parameters are the form inputs. For static Web pages, no input parameter is required. RUDF always returns a table of data as output. (Note that a single data item can also be treated as a table with one field.) This rendering conforms to the table function concept in the new SQL:2003 standard [31].

Four elements are required for the data definition language (DDL) in Meteoroid to define an RUDF: data location, input parameters, output fields, and a candidate HTQL (Hyper Text Query Language) expression. The data location specifies the location of a data source in the Universal Resource Location (URL) format. Input parameters define the inputs in a Web form that need to be bound with actual data at run time. Default values for input can be defined in the DDL. Output fields define a projection and transformation of attributes from an internal wrapper. The candidate HTQL expression is responsible

for the creation of an internal wrapper for data extraction. The reason for an additional internal wrapper is that the internal wrapper is transformed and maintained by the system automatically. It is typically more adaptable to changes occurring in the source data than a handcrafted wrapper. However, the HTQL expression itself can be seen as a simple-form wrapper when robustness is not a big concern and can be traded for speed.

Consider the following DDL to create a RUDF named `KEGG_LinkDB` for pathway finding from `DBGET/LinkDB` at the KEGG website. The location of the data source is ‘`http:// www.genome.ad.jp/ dbget-bin/ www_linkdb`’, and it uses the fourth form in the Web page. There are two input parameters: the *keywords* and the *targetdb*, with default parameter values of ‘`hsa:126`’ and ‘`path`’, respectively. A candidate HTQL expression is ‘`<PRE>.<A>`’. Two output fields are generated: `path_url` and `path_id`. Notice the two output fields are transformed from the same HTQL output – one takes the URL address from a hyperlink and the other takes the text (as a path ID).

```

REDEFINE FUNCTION KEGG_LinkDB
  HREF [http://www.genome.ad.jp/dbget-bin/www_linkdb]
  FORM 4
  PARAMETER
    keywords VARCHAR(255) DEFAULT 'hsa:126',
    targetdb VARCHAR(255) DEFAULT 'path',
  GIVING [<PRE>.<A>] FIELDS
    [%1:href &url] AS path_url,
    [%1:tx] AS path_id;

```

An advantage of the declarative DDL for an RUDF is the succinct syntax. The DDL itself is enough for the system to know where to fetch data, how to wrap the resulting data, and what the input-output behavior of the data source is. No additional wrapper

training task is required. The second advantage is the convenience in defining the wrapper. A user does not need to develop a very complex and robust wrapper. The Meteoroid system will generate a robust wrapper from the candidate HTQL expression and maintain it properly. The third advantage is its add-on data transformation power. Data projection and transformation in the definition of the output fields provides a flexible channel for external programs to manipulate Web data while allowing the internal wrapper to be kept intact. Finally, the declarative DDL makes ad hoc integration of a large number of Web sources straightforward. We will see from Section 6.7 that users of the Meteoroid system do not need to write the DDL by hand. Instead, a few clicks on the Web pages are enough for an assistant program to compose the DDL automatically.

Once an RUDF is defined, it can be queried as a regular table in the database as well as joined with other tables, including other RUDFs. Data binding is performed automatically on an RUDF when it is joined with other tables or there are constant value assignments in the query expression. Input fields without binding will use the default values defined in the DDL. Meteoroid will schedule the execution order of distributed Web queries that generate and consume data from each other. Another method for using an RUDF is to actually use it as a function. The input parameters can be set to constant values, other table fields, or default values. The functional view allows a manually defined execution order for multiple RUDFs. We must point out that since an RUDF takes a relation as input and generates a relation as output, it satisfies the closure property in the relational data

model. Since we do not limit the source to which an input/output field can be bound, it also satisfies the compositionality property.

6.5 Multi-layer Table and View Logical Design

Meteoroid compiles the declarative DDLs into a multi-layer logical design of tables and views which is maintained in a data ontology file. A Meteoroid data ontology describes the source schema and the data mediation rules of a Meteoroid object (table, view, data source, etc.) in XML format. Meteoroid adopts a four-layer logical design – the data source, physical table, virtual table, and virtual view layers. The physical table layer includes actual tables managed by remote data sources. Remote data sources may provide interfaces such as Web interfaces or ODBC interfaces for access to the physical tables, but the physical tables are typically autonomous for an integration system. As a result, Meteoroid will not manage the physical layer directly. Meteoroid maintains the other three layers of data definitions in a data ontology file, including data source definitions, virtual table definitions, and virtual view definitions. We will discuss the three layers of data definitions in more detail below.

6.5.1 Data Source Definition

Each data source has a source definition tag defining the capability of the data source and the method to connect to the data source. For example, a relational data source has query and update capabilities, while an online HTML file has only query capability. A

relational data source can be connected with an ODBC description, and an HTML file can be connected with a URL description.

Figure 6.1 describes the definition of the DBGET/LinkDB data source, which is a Web data source that can be accessed from a Web form at the URL 'http://www.genome.ad.jp/dbget-bin/www_linkdb'.

```

<DataSource Name='KEGG_LinkDB' Type='Form'>
  <Form
    Type='Url'
    URL='http://www.genome.ad.jp/dbget-bin/www_linkdb'
    FormIndex='4'
  />
</DataSource>

```

Figure 6.1 Definition of the LocusLink data source

6.5.2 *Virtual Table Definition*

A virtual table definition describes an exported relation from a data source. A data source can have multiple exported virtual tables. For example, a relational database may have multiple tables to be exported, and an HTML page may be interpreted as multiple views. A virtual table definition describes a method to transform the source data into an exported table, the names and types of the table fields, and the capability of the fields.

With a virtual table definition, a standard SQL query can be translated into a source query without difficulty. The virtual table provides a uniform data access interface for heterogeneous data stored at different locations in different media.

Figure 6.2 shows an example definition of a DBGET/LinkDB virtual table including the fields 'keywords', 'targetdb', and 'path', where the fields 'keywords' and 'targetdb' are input parameters and the field 'path' is an output field.

```

<Table Name='KEGG_LinkDB' DataSource='KEGG_LinkDB' Form='4'
  Htql='<PRE>.<A>'>
<Fields>
  <Field Name='keywords' Type='varchar' Length='50' >
    <SourceField Type='IN' Name='keywords' Value='hsa:126' />
  </Field>
  <Field Name='targetdb' Type='varchar' Length='50' >
    <SourceField Type='IN' Name='targetdb' />
  </Field>
  <Field Name='path' Type='memo' Length='0' >
    <SourceField Type='OUT' Htql='%1' />
  </Field>
</Fields>
</Table>

```

Figure 6.2 An example of the GenBank virtual table definition

6.5.3 Virtual View Definition

A virtual view defines a mediation strategy for data from multiple data sources. It includes specifications of the connections among data sources, the unified schema, and the mediation rules. A connection among data sources is defined in a grouping of virtual table fields.

Fields grouped together form a united field. Conflicts may occur for a united field during the data integration stage. The conflicts are resolved by the mediation rules. A mediation rule may specify a group function such as max or min to compute the united field value, an evaluation function to select a best value, or a dependency rule to check dependency with other fields.

Queries to a virtual view are translated into source queries during execution. The runtime query translation ensures that the data is fresh. The capability of each data source is inferred from the virtual table definition.

Figure 6.3 shows a virtual view connecting virtual tables of KEGG_gene and KEGG_LinkDB. The new *entry* field in the virtual view is a merge of the *entry* field in the table KEGG_gene and the *keywords* field in the table KEGG_LinkDB.

6.5.4 Advantage of Multi-layer Logical Design

The four-layer logical design introduces a set of unique advantages. First, the logical independence of data definitions in each layer is more adaptable to the autonomicity property of remote data sources. Physical tables in distributed data sources typically evolve quickly. Virtual tables, however, reflect a relatively stable local view of physical tables. Further, virtual tables can be dynamically aggregated into virtual views, making the connection of disjointed data sources flexible. Second, separating virtual table design and virtual view design allows a convenient interpolation of interoperability rules in the virtual view to resolve data heterogeneity tasks without violating the definition of each source table. A vir-

```

<View Name='path_view'>
  <Fields>
    <Field Name='keywords' Type='varchar' Length='50'>
      <FieldSource DataSource='KEGG_gene' TableName='KEGG_gene'
        FieldName='keywords' />
    </Field>
    <Field Name='entry' Type='varchar' Length='50'>
      <FieldSource DataSource='KEGG_gene' TableName='KEGG_gene'
        FieldName='entry' />
      <FieldSource DataSource='KEGG_LinkDB' TableName='KEGG_LinkDB'
        FieldName='keywords' />
    </Field>
    <Field Name='path' Type='memo' Length='0'>
      <FieldSource DataSource='KEGG_LinkDB' TableName='KEGG_LinkDB'
        FieldName='path' />
    </Field>
  </Fields>
</View>

```

Figure 6.3 A virtual view definition connecting DBGET/gene and DBGET/LinkDB Web tools

tual view reflects a specific data integration solution by users. A virtual view can be stored in data ontology files permanently or constructed and destroyed on the fly during query execution time. A permanent virtual view allows more complex data interoperability rules to be encoded with the data ontology and be maintained incrementally. On the other hand, one-time queries involving multiple data sources can be translated into an intermediate virtual view to resolve data heterogeneity and be destroyed after the query is completed. Third, multi-layer logical design allows more advanced data management strategies to be carried out at different logical levels. In addition to the automatic wrapper maintenance technique at the virtual table layer and the data interoperability rule management at the virtual view layer, another powerful flow control mechanism is implemented at the virtual view layer. Since a virtual view reflects an aggregation of tools and data, scheduling the execution order of data queries is important for scientific data analysis. The flow control mechanism allows scientists to organize the execution order of source queries according to a specific scientific purpose. In our earlier example, a query to DBGET/LinkDB will be submitted only when the entry name has been retrieved and satisfies the condition of being prefixed with 'hsa:'. As another example, to execute a query of a BLAST search of NCBI, the retrieval of a BLAST result must be repeatedly submitted until the search is completed at the server site (The BLAST server will return a BLAST ID only when the query is not completed). Providing these functionalities makes Meteoroid more capable of complex scientific data analysis tasks.

6.6 Scheduling-oriented Query Processing

As we mentioned previously, query scheduling is an important task for scientific data integration. Meteoroid has designed a novel mechanism to handle the query scheduling problem. Specifically, Meteoroid introduces the flow control mechanism in query processing. This section will describe the Meteoroid query scheduling solution. First, we discuss how to interpret dependencies among data sources. Then we describe the modules – the query rewriting, query transformation and query scheduling modules – for query scheduling in more detail. Finally, we discuss the mediation of results with interoperation rules.

6.6.1 Dependencies Among Data

There are two categories of dependencies among data in Meteoroid. One is the *internal table dependency*, and the other is the *external table dependency*. Here we use a table to mean either a virtual table or a virtual view, which itself is an integration of multiple tables. Internal table dependency means that some fields of a table depend on other fields of the table. Internal key dependency is one kind of internal table dependency. Fields not in the key depend on the key fields. Input-output dependency is another kind of internal table dependency which is unique to RUDF-based tables. We call the input fields the *input key* of the table. Fields not in the input key depend on the input key fields. We use

$$T : I \vdash O$$

to denote an internal dependency of table T where attributes in set O depend on attributes in set I . For example, our motivating example has the following input-output dependencies:

$B : \text{geneid} \vdash \text{description};$
 $C : \text{description} \vdash \text{entry};$
 $D : \text{entry} \vdash \text{path, pathway, pathurl}.$

External table dependency describes the relationship where fields of one table depend on fields from another table. For example, in our running example, the entry name in table D depends on the entry name in table C . External dependency occurs in a query or a virtual view. For a virtual view, the dependency rules have been encoded in the data ontology as a part of the view definition. For a query, external dependencies have to be checked and set up at run time. If a table does not externally depend on other tables in a view, then we say the table is *grounded*. Obviously, a query of data from a table that is not grounded has to wait until the dependent attributes have been bound. We use

$$T_1 : I \vdash T_2 : O$$

to denote an external dependency between tables T_1 and table T_2 , where attributes in set O of T_2 externally depend on attributes in set I of T_1 . In our example, the following external dependencies hold:

$A : \text{geneid} \vdash B : \text{geneid};$
 $B : \text{description} \vdash C : \text{description};$
 $C : \text{entry} \vdash D : \text{entry};$
 $D : \text{pathurl} \vdash E : \text{url}.$

Internal and external table dependencies specify the constraints we need to consider in the execution of queries involving multiple tables. It is a basic concept in our further discussion of query processing.

6.6.2 Query Rewriting

Once Meteoroid receives a query, the first step is to analyze the query to see if there are any data dependencies among multiple tables. If there is only one table involved, the query is delivered directly to the query transformation module. Otherwise, a temporary virtual view is created to encode table dependency rules, and the query is rewritten into a query upon the virtual view. Both the temporary virtual view and the rewritten query are then delivered to the query transformation module.

The idea behind query rewriting is simple. It checks join conditions in the query. A new field in the virtual view is created for each set of join attributes. For each pair of join attributes $\langle \alpha, \beta \rangle$, where α and β are fields in tables T_1 and T_2 respectively, if α is a key field of T_1 but β is not a key field in table T_2 , then an external dependency rule is set up between α and β :

$$T_1 : \alpha \vdash T_2 : \beta.$$

Join conditions are removed from the original query, and fields appearing in the rest of the query are rewritten into corresponding fields in the new view. At this point, query rewriting is completed.

During the query rewriting step, only attributes relevant to the query are encoded into the virtual view. Thus, unnecessary data retrieval can be saved in the remaining query processing steps. This savings can be remarkable when some attributes require additional Web access.

6.6.3 Query Transformation

The task of query transformation is to transform a query into an internal query graph, set up an integration environment and generate candidate scheduling plans. The query graph consists of a condition expression tree, a set of output expression trees, and a list of source tables with their optimization conditions. The integration environment includes the table dependency rules and data mediation rules read from the data ontology and necessary cache tables created for intermediate query results. A candidate scheduling plan is a linear-directed graph consisting of a sequence of source tables where each table depends only on tables preceding it. Candidate scheduling plans are derived from table dependency rules. Meteoroid derives a candidate plan for each grounded table. We will assume tables are not circularly dependent on each other. Thus, the table dependencies form a partial-order relationship among tables, and each plan can be derived with a standard sorting algorithm.

6.6.4 Query Scheduling

Meteoroid employs a pipelining strategy for query scheduling. Pipeline scheduling has a set of advantages over traditional query scheduling in the Web environment. Since data

navigation from the Web is typically slow, a complete join of two Web data sources may require a lot of time. Waiting for the complete join of a large number of Web sources is usually unacceptable. On the other hand, users doing distributed Web queries are more interested in quickly seeing partial results first, and then waiting for the complete result set or abandoning the query altogether based on the examined partial results. Furthermore, pipeline scheduling can avoid overburdening a particular remote Web source. Traditional query scheduling joins tables sequentially. Each table join may incur an extensive data visit on the based table. A table from a Web data source may be clogged by a sudden and explosive Web access. Pipelining will make Web access evenly distributed over the whole query execution time and allows access to different Web sources to be interleaved. Thus no remote Web source will be suddenly overwhelmed.

Meteoroid creates a novel two-phase pipelining technique to schedule distributed queries and resolve data heterogeneity. The first phase is the *key-pipelining*, and the second phase is the *data-pipelining*. Key-pipelining uniquely identifies the set of keys of the results data, while data-pipelining retrieves data and resolves heterogeneities. Separation of key retrieval and data retrieval is a particular consideration for Web data integration. First, keys constitute a much smaller set of data than the data of the whole view. Retrieving only the keys first will eliminate unnecessary data retrieval. Second, data with the same key may be duplicated at different sites with different data representations. Special data mediation can be conducted separately at the data-pipelining stages. Data with the same key can be retrieved and mediated together to ensure that partial results are correct and integrated.

Key-pipelining is scheduled in batch mode starting from a grounded table. Meteoroid chooses a candidate plan and a set of tuples (*seed tuples*) from the starting table. Following each plan, Meteoroid derives all of the keys from the source tables in a pipelined fashion, where each table binds input parameters from data of preceding tables. Each seed tuple may derive multiple key tuples, which we call the *key factor* of a plan. A plan with a large key factor may result in a slow response of available results. Therefore, Meteoroid will try, for each plan, to select a plan with a minimal key factor. We do the plan selection dynamically because in an ad hoc integration scenario few statistics are available for remote data sources. The scheduling procedure can be captured by the following *key-pipelining* algorithm.

Algorithm *key-pipelining*

Input: A set of candidate plans P , a constant k ;

Output: Key tuples K ;

Begin

For each candidate plan p in P

 Fetch a seed tuple from the grounded table of p ;

 Derive key tuples from the seed tuple;

 Compute key-factor α as the number of unique key tuples newly derived;

 Choose the plan p' with a minimal key-factor α ;

While the grounded table in p' is not end;

 Fetch k seed tuples from the grounded table;

 Derive key tuples from the seed tuples;

End

Data-pipelining is similar to key-pipelining. The difference is that instead of starting from a seed tuple, data-pipelining starts from a key tuple. This happens when a non-key attribute needs to be retrieved. Meteoroid first develops a data-retrieval plan for each non-key field in the view. A data-retrieval plan is based on the dependencies in a virtual view

connecting the key fields and the target field. A shortest path is computed, which reflects the shortest sequence of Web navigations needed in order to retrieve the data attribute starting from a set of key values. Dijkstra's algorithm [23] is used to compute the shortest path. Following a data retrieval plan for one table at a time, Meteoroid retrieves data from the tables in a pipeline, where data from preceding tables are bound to the input fields of the next table.

Data-pipelining will not increase the number of result tuples, although each view field may have multiple data values from multiple source tables. In cases where multiple and conflicting results are retrieved from different sources, a result mediation procedure is needed. We will discuss this issue next.

6.6.5 Results Mediation

Scientific data analysis requires a combination of data from different sources. This may be due to the fact that the data from a single source is incomplete or a more complex data transformation procedure is expected. In our running example, only the KEGG database is queried for pathway searching. However, KEGG collects only a subset of pathways. Other pathway databases such as EcoCyc [48] may be included in our query for a larger scale pathway analysis. When the pathway information is not present in the KEGG database, we can use the pathway information from EcoCyc instead and vice versa. This is an example of *incomplete information*. Since the pathways in KEGG and EcoCyc are developed and curated by different groups of people using different methods, conflicting results may

occur when they generate different pathways. If we believe only one of them is correct, we have *incorrect information* from the other. Otherwise, if we believe both of them will contribute in part to a combined result, we have *imperfect information* from each of them. A method of solving these information problems from the database level apparently is desirable for complex scientific data analyses.

Meteoroid allows data mediation rules to be encoded with a virtual view. A non-key field in a view combining fields from multiple sources will be mediated by the rule. The rule decides whether to discard any information or to transform the information into a combined format. For instance, we may want to count available activating pathways from all sources instead of seeing any of them.

6.7 Visual Interfaces for Ad Hoc Integration

The visual interface is one of the most important factors in designing a scientific data integration system. In an ad hoc integration scenario it is even more crucial. A direct impact of a convenient visual interface is that scientists are more likely to use the technology when it is user friendly. From the computer science point of view, how user interfaces can be designed reflects the level of maturity in the system architecture organization and the soundness of the underlying technology. Our interfaces will show that ad hoc integration is practical for real world data integration problems. The basis of a declarative language used for data description and manipulation proves that Meteoroid technology is platform independent and can interoperate with existing programs well.

Meteoroid visual interfaces demonstrate three functionalities. First, we show that Web data can be defined conveniently and in a few clicks by employing our *PickUp* technique. Second, we show that integration of distributed Web data is simple – just drawing the connection between matching fields. Third, we show that distributed queries are scheduled properly, and pipelined results help users evaluate their queries in the early stages of execution. These three functionalities demonstrate the declarativeness, compositionality, and pipeline scheduling properties of our ad hoc integration approach, respectively.

6.7.1 Picking Up Web Data From Scratch

The first step toward Web data integration is to define Web data. Meteoroid provides a navigation-oriented interface to allow users to explore the Web as usual. A user can also submit Web forms and retrieve the dynamically generated results. At a page of interest, a user can pick up the the form input (which may be a text box, a listbox, a checkbox, etc.) with a click on the ‘Pickup Input’ button (Figure 6.4). Alternatively, a user can pick up a piece of information or an entire table from the page with a click on the ‘Pickup Result’ or ‘Pickup Table’ button (Figure 6.4).

In the background, Meteoroid employs the *PickUp* technique to automatically generate a wrapper for each piece of information the user has picked up. Upon a request to ‘Create RUDF,’ Meteoroid composes an RUDF table DDL according to the information the user has picked up. The table, with automatically labeled field names, is displayed to the user graphically (Figure 6.5). Sample data extracted from the previous pages are also shown.

The screenshot shows the Meteoroid-WebFusion interface with the following components:

- Search Bar:** LocusLink search with query '15004'.
- Table of Results:**

| LocusID | Org | Symbol | Description | Position | Links |
|---------|-----|----------|--|-------------|-------------|
| 15004 | H2 | Pb | Mistocompatibility 2, P region beta locus | 17 18.51 cM | [P] [G] [V] |
| 3116 | Hs | HLA-DPB2 | major histocompatibility complex, class II, DP beta 2 (pseudogene) | 6p21.3 | [P] [G] [V] |
- Annotations:**
 - Pickup input from Web pages:** Points to the 'Pickup Value' button.
 - Apply RUDF wrapper technique:** Points to the 'Pickup Input' button.
 - Pickup results from Web pages:** Points to the 'Pickup Table' button.
 - Apply table wrapper technique:** Points to the 'Pickup Table' button.

Figure 6.4 Picking up a table of Web data in a click

The user can confirm the RUDF table creation, modify the field names, or perform a set of standard transformations over the extracted fields.

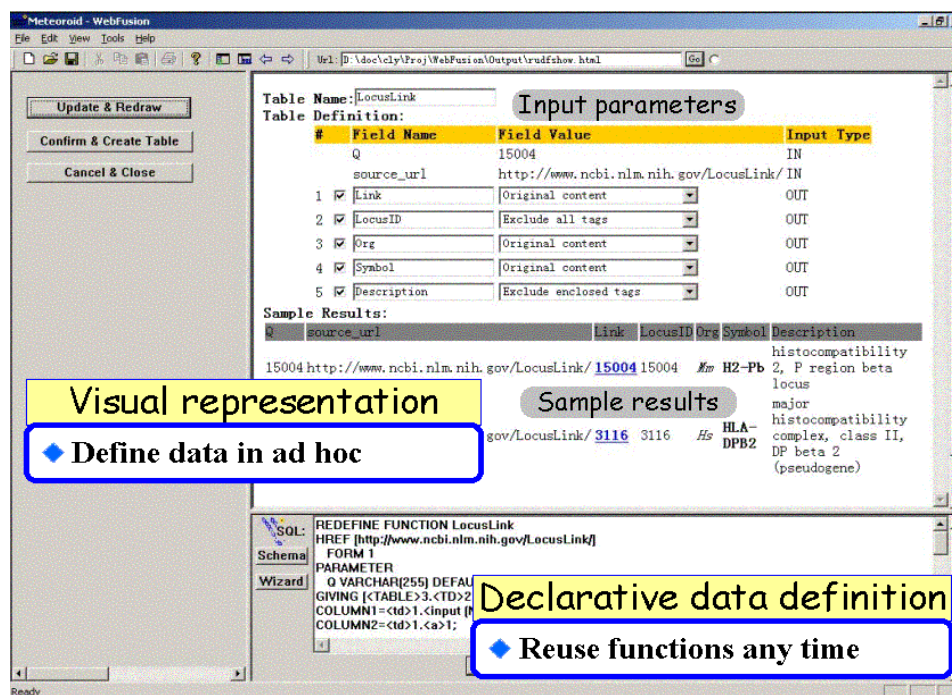


Figure 6.5 Customizing RUDF table creation

For example, to define the LocusLink data in our running example, a user can navigate to the LocusLink website ('http://www.ncbi.nlm.nih.gov/LocusLink/'), enter the search term '15004' (and click the 'Pickup Input' button to remember the input), and submit the search. LocusLink will return two records to the user (as shown in Figure 6.4). The user marks the first record and clicks on the 'Pickup Table' button to identify the results data of interest. Then the user clicks the 'Create RUDF' button. The results page is displayed as a LocusLink table as shown in Figure 6.5. The user names the table 'LocusLink' and names

the columns 'Link', 'LocusID', 'Org', 'Symbol', and 'Description', respectively. Further, the user can specify the 'LocusID' field as excluding any tags and the 'Description' field as excluding the enclosed tags. Then the user confirms the creation of the LocusLink table.

The whole data definition procedure can be captured in a declarative RUDF DDL expression shown in an SQL box at the bottom right side of the screen. The DDL expression can be recorded, stored, and transferred electronically for future reference. The DDL can be executed at other Meteoroid systems at any time without repeating the above manual operations. As a result, it provides a means for researchers to record data analysis procedures.

6.7.2 Fusing *Distributed Web Sources*

Once the definition of data sources is completed, we can query distributed data just as we query an in-house database. The fusion of distributed data is no more complex than querying multiple tables locally. Researchers with some knowledge of SQL can directly submit SQL queries to Meteoroid. Otherwise, we provide a wizard interface to help design the fusion of distributed data.

The wizard interface works incrementally based on each pair of table connections. In our running example, the DBGET/gene site (defined as a KEGG_gene table) acquires the gene description from the LocusLink site (defined as the LocusLink table) and the DBGET/LinkDB site (defined as the KEGG_LinkDB table) draws the entry name from the DBGET/gene site. We can link these connections through the wizard interface as shown

in Figure 6.6. We can further specify a filtering condition for each table. For example, we may specify the entry name with the prefix 'hsa:' as a filter condition.

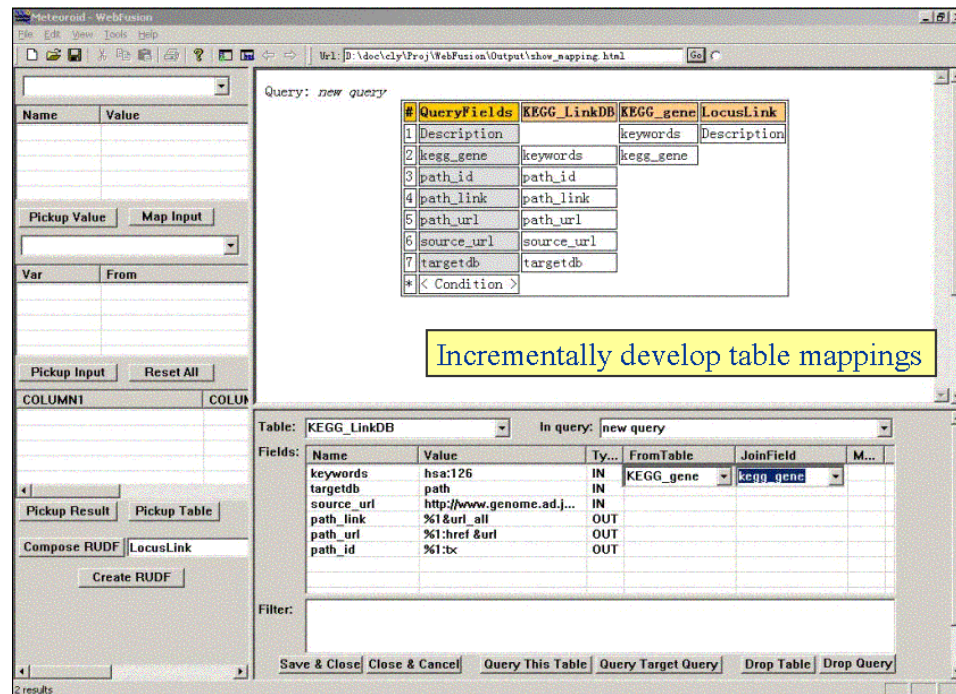


Figure 6.6 Connecting tables incrementally

The wizard interface returns a multiple-table SQL query expression in the SQL box (Figure 6.7). Again this expression can be recorded and stored for future reference.

6.7.3 Monitoring Query Results

As Meteoroid schedules queries in a pipeline fashion, a query in progress can be monitored in real time. Figure 6.8 shows the in progress results for our running example. Queries can be terminated or resumed as needed.

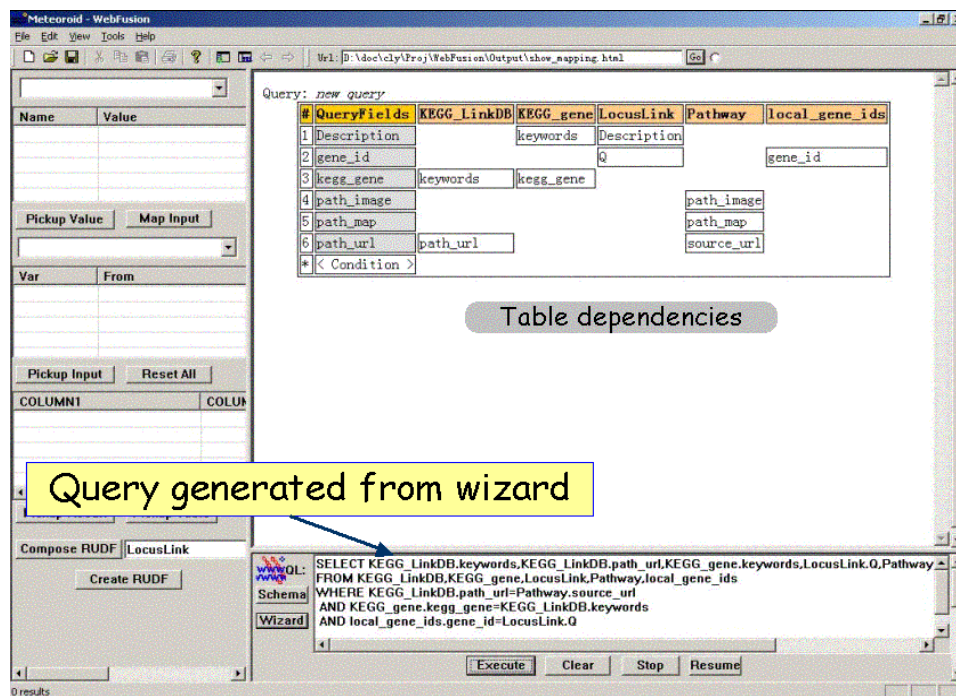


Figure 6.7 Generating SQL queries from wizard

Meteoroid can execute queries in a batch. Experimental procedures can be recorded declaratively in a batch and be submitted to Meteoroid as a whole.

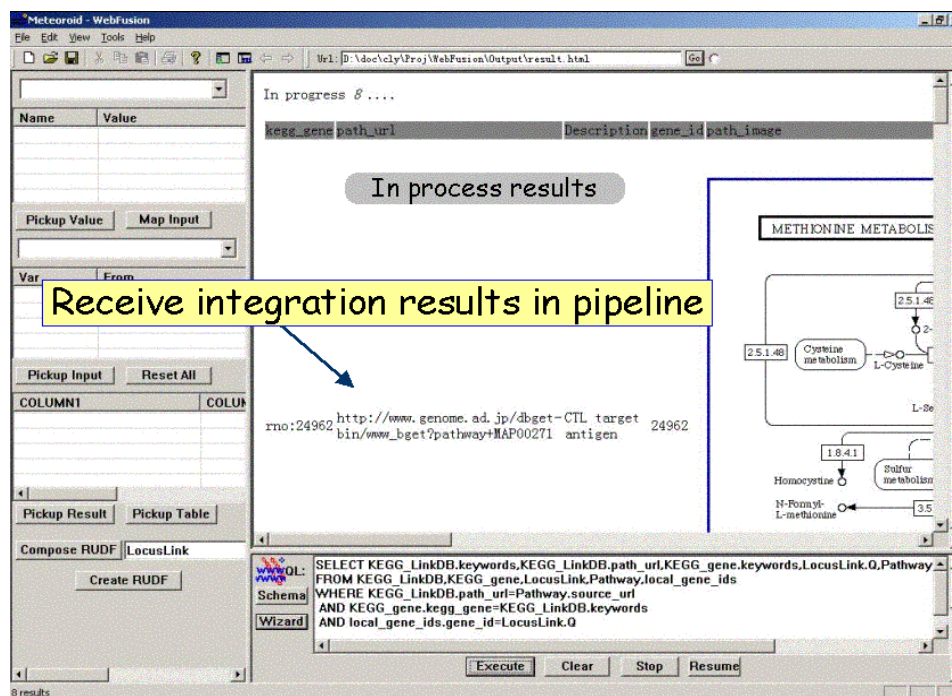


Figure 6.8 Monitoring in progress query results

6.8 Conclusion

We have described an ad hoc integration methodology to extend a database management scheme in the Web environment. Distributed and dynamic Web data has unique properties that are different from the data in a traditional database such as a semi-structured organization, a navigational orientation, the lack of a schema and statistics, etc. Existing data integration approaches show limitations in both data modeling and query processing, leav-

ing a great burden for scientific researchers to utilize distributed Web resources for data analyses in a timely manner. An ad hoc integration approach allows scientists to easily integrate a large number of distributed data sources for daily scientific data investigation purposes.

A novel two-phase pipeline scheduling technique was devised for distributed querying and integration of heterogeneous online data sources. This technique ensures that integrated queries to a large number of distributed Web resources will be scheduled and streamlined properly without overwhelming remote servers, while available results will be expedited to users as soon as possible.

An ad hoc integration system, the Meteoroid system, was shown to resolve an example problem from life science data research. A declarative language to integrate and query Web data was provided. The declarative language enables biologists to define Web-oriented data discovery electronically and repeat experiments easily. A visual interface was provided to help biologists integrate and query remote data in a few clicks.

CHAPTER VII

CONCLUSION

The Web has entered our life as a major information source. In addition to the information dissemination role, the Web has been used by scientific communities to link distributed databases and share scientific tools. Researchers now are able to access public data all over the world and conduct scientific computing tasks online. The ever increasing influence of a number of major biological databases such as NCBI, EMBL and SwissProt makes biologists more and more dependent on online resources for data analysis. Research has been devoted to the development of new technologies to facilitate automated Web computing tasks.

A relational database framework for Web data management has been recognized as a valid approach and provides a number of unique advantages. First, by turning Web operations into relational algebra expressions, complex Web computing tasks can be scheduled and optimized by computer systems to increase efficiency. Second, a database management layer provides better data independency for Web computing applications. Third, relational database systems have been widely accepted by field researchers so there is no need to retrain researchers.

Existing research fails in different ways to achieve a practical database management framework for scientific Web computing. First, manual and time-consuming wrapper training prohibits scientific users from being able to integrate Web data in a timely manner. Second, brittle and manually maintained wrappers hinder an integration database from scaling up to larger data sets and adapting to changes. Third, traditional database schema management and query planning techniques are limited in their ability to deal with a large degree of data heterogeneity in Web data.

This dissertation argues that an *ad hoc integration* methodology to extend a database management concept to the Web environment is plausible for scientific Web computing tasks. First, ad hoc integration eliminates the complex wrapper training requirement for Web data integration. Instead, a declarative and convenient data definition language will produce a fully automated wrapper generation module. Second, ad hoc integration allows scientists to integrate Web resources of interest for scientific data investigation purposes in a timely manner and adjust parameters for dedicated queries. Third, ad hoc integration allows scientists to define Web-intensive data investigation procedures electronically and repeat experiments easily for the ever accruing scientific data.

The contributions of this dissertation can be summarized as follows:

- Ad hoc integration fully automates the syntactic integration of Web data under a relational framework. It resolves the wrapper development bottleneck found in other heterogeneous database systems.
- This dissertation research develops a set of data modeling techniques to effectively and robustly transform Web data into relational entities. First, form-based Web data are modeled as remote functions. Second, table-based Web data are modeled as tables. By using these two constructs, Web data becomes compatible with the tradi-

tional relational data model and can be queried homogeneously with in-house relational database systems.

- This dissertation research devised an automatic maintenance mechanism to adjust an ad hoc integration system to the changing Web resources. The automatic maintenance mechanism frees users from having to constantly monitor the changing Web sources and allows the integration system to be easily scaled up.
- This dissertation research facilitates the data integration purpose through a novel two-phase pipeline scheduling technique to query and mediate heterogeneous Web data. This technique ensures that integrated queries to a large number of distributed Web resources will be scheduled and streamlined properly without overwhelming remote servers, while mediated results will be expedited to users as soon as possible.
- This dissertation research has demonstrated that declarative integration of Web data is possible. A declarative language enables biologists to define data investigation procedures electronically and repeat experiments easily. A visual GUI has been developed to further assist users to compose distributed queries.

Ad hoc integration technology will prevent scientists from having to manually navigate through an increasing number of scientific databases. We have demonstrated the use of the ad hoc integration system for a biological pathway finding problem. We have not conducted experiments to verify how much time a biologist can save using the ad hoc integration system. However, reasonable estimations indicate that the savings can be dramatic. Suppose a user needs to compute 1000 tuples of data using four Web databases sequentially. Suppose each Web database navigation costs a unit of time. Navigating the 1000 tuples manually would require $1000 \times 4 = 4000$ units of time. On the other hand, if the user uses ad hoc integration, he or she needs only to navigate 4 pages manually with a sample tuple (with a cost of 4 units of time); the rest of the 999 tuples can be computed automatically by the system with an SQL query. There will be some overhead for learning the system, picking up data from each of the 4 pages, and formulating the SQL query using

the visual interfaces. However, such overhead occurs only one time for the sample tuple and becomes negligible when the amount of data to be computed is large.

Another advantage of an ad hoc integration approach is the ability for a scientific community to share data exploration procedures. Since scientific Web databases have become increasingly complex and the data in each database are updated daily, manually verifying past experiments is time-consuming and unaffordable. Sharing of experimental procedures allows experiments to be verified later when the data has changed and allows peer researchers to utilize prior experiments for larger-scale data analysis.

Ad hoc integration provides a foundation for further integration of existing Web data more intelligently – such as reaching a stage by the Semantic Web vision [78] where software agents automatically go to the Web and plan routine jobs for humans – but without a total reconstruction of the current Web infrastructure.

REFERENCES

- [1] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian, "Query Caching and Optimization in Distributed Mediator Systems," *Proceedings ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, Jun 1996, pp. 137–148, ACM Press.
- [2] B. Adelberg, "NoDoSE - A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents," *Proceedings ACM SIGMOD International Conference on Management of Data*, 1998, pp. 283–294.
- [3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res.*, vol. 25, 1997, pp. 3389–3402.
- [4] A. Arasu and H. Garcia-Molina, "Extracting structural data from web pages," *Proceedings ACM SIGMOD International Conference on Management of Data*, San Diego, California, Jun 2003, pp. 337–348.
- [5] Y. Arens, C. A. Knoblock, and W.-M. Shen, "Query reformulation for dynamic information integration," *Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration*, vol. 6, no. 2/3, 1996, pp. 99–130.
- [6] N. Ashish and C. Knoblock, "Semi-automatic Wrapper Generation for Internet Information Sources," *CoopIS*, 1997, pp. 160–169.
- [7] P. Atzeni and G. Mecca, "Cut & Paste," *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997, pp. 144–153, ACM Press.
- [8] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens, "TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources," *6th International Conference on Intelligent Systems for Molecular Biology*, Montreal, Canada, 1998, pp. 25–34, AAAI Press, Menlo Park.
- [9] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual Web Information Extraction with Lixto," *Proceedings of 27th International Conference on Very Large Data Bases, VLDB 2001*, Roma, Italy, Sep 2001, pp. 119–128, Morgan Kaufmann.

- [10] R. J. Bayardo, B. Bohrer, R. S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, "The InfoSleuth Project," *Proceedings ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 1997, pp. 543–545, ACM Press.
- [11] P. Bonnet, J. E. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," *Proceedings of the Second International Conference on Mobile Data Management*, Jan 2001.
- [12] H. Bono, S. Goto, W. Fujibuchi, H. Ogata, and M. Kanehisa, "Systematic Prediction of Orthologous Units of Genes in the Complete Genomes," *Genome Informatics*, vol. 9, 1998, pp. 32–40.
- [13] O. Boucelma, S. Castano, C. Goble, V. Josifovski, Z. Lacroix, and B. Ludäscher, "Report on the EDBT'02 Panel on Scientific Data Integration," *ACM SIGMOD Record*, vol. 31, no. 4, 2002, pp. 107–112.
- [14] A. Bouguettaya, B. Benatallah, L. Hendra, J. Beard, K. Smith, and M. Ouzzani, "World Wide Database - Integrating the Web, CORBA, and Databases," *Proceedings ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, Jun 1999, pp. 594–596, ACM Press.
- [15] P. Buneman, M. F. Fernandez, and D. Suciu, "UnQL: a query language and algebra for semistructured data based on structural recursion," *VLDB Journal: Very Large Data Bases*, vol. 9, no. 1, 2000, pp. 76–110.
- [16] R. G. G. Cattell, "ODMG-93: A Standard for Object-Oriented DBMSs," *Proceedings ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 1994, p. 480, ACM Press.
- [17] D. D. Chamberlin, *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann, San Francisco, CA, 1998.
- [18] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources," *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, Tokyo, Japan, Oct 1994, pp. 7–18.
- [19] L. Chen and H. M. Jamil, "On Using Remote User Defined Functions as Wrappers for Biological Database Interoperability," *International Journal on Cooperative Information Systems*, vol. 12, no. 2, 2003, pp. 161–195, Special Issue on Biological Databases.

- [20] L. Chen, H. M. Jamil, and N. Wang, “1st International Workshop on Biological Data Management - BIDM 03,” *Automatic Wrapper Generation for Semi-Structured Biological Data Based on Table Structure Identification*, Prague, Czech Republic, 2003.
- [21] Z. Chen and P. Seshadri, “An Algebraic Compression Framework for Query Results,” *Proceedings of the International Conference on Data Engineering*, Santiago, Chile, Mar 2000.
- [22] G. O. Consortium, “Gene Ontology Consortium,” <http://www.geneontology.org/>, accessed on Mar 20, 2004.
- [23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, The MIT Press, Cambridge, Massachusetts, 2001.
- [24] V. Crescenzi, G. Mecca, and P. Merialdo, “RoadRunner: Towards Automatic Data Extraction from Large Web Sites,” *Proceedings of 27th International Conference on Very Large Data Bases, VLDB 2001*, Roma, Italy, Sep 2001, pp. 109–118, Morgan Kaufmann.
- [25] S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and J. C. J. Stoeckert, “K2/Kleisli and GUS: Experiments in integrated access to genomic data sources,” *IBM Systems Journal*, vol. 42, no. 2, 2001, pp. 512–531.
- [26] DBGET/gene, “Search GENES database using DBGET,” http://www.genome.ad.jp/htbin/www_bfind?genes, accessed on Apr 3, 2004.
- [27] DBGET/LinkDB, “Search LinkDB database using DBGET,” http://www.genome.ad.jp/dbget-bin/www_linkdb, accessed on Apr 3, 2004.
- [28] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, “XML-QL: A Query Language for XML,” *W3C Note*, Aug 1998, <http://www.w3.org/TR/NOTE-xml-ql>.
- [29] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, Cambridge University Press, 1998.
- [30] L. Eikvil, “Information extraction from the world wide web: a survey,” *Technical Report 945*. 1999, Norwegian Computing Center.
- [31] A. Eisenberg, K. Kulkarni, J. Melton, J.-E. Michels, and F. Zemke, “SQL:2003 Has Been Published,” *ACM SIGMOD Record*, vol. 33, no. 1, Mar 2004.
- [32] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith, “Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages,” *Journal of Data & Knowledge Engineering*, vol. 31, no. 3, Nov 1999, pp. 227–251.

- [33] D. W. Embley, Y. S. Jiang, and Y.-K. Ng, "Record-Boundary Discovery in Web Documents," *Proceedings ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, 1999, pp. 467–478.
- [34] T. W. Finin, R. Fritzon, D. McKay, and R. McEntire, "KQML As An Agent Communication Language," *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, Maryland, Nov 1994, pp. 456–463, ACM.
- [35] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu, "Query Optimization in the Presence of Limited Access Patterns," *Proceedings ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, Jun 1999, pp. 311–322, ACM Press.
- [36] X. Gao and L. Sterling, "AutoWrapper: automatic wrapper generation for multiple online services," *Asia Pacic Web Conference '99*, Hong Kong, 1999.
- [37] M. Godfrey, T. Mayr, P. Seshadri, and T. v. Eicken, "Secure and Portable Database Extensibility," *Proceedings ACM SIGMOD International Conference on Management of Data*, L. M. Haas and A. Tiwary, eds., Seattle, Washington, Jun 1998, pp. 390–401, ACM Press.
- [38] P. M. Gray, G. J. L. Kemp, C. J. Rawlings, N. P. Brown, C. Sander, J. M. Thornton, C. M. Orengo, S. J. Wodak, and J. Richelle, "Macromolecular structure information and databases," *TIBS*, vol. 21, 1996, pp. 251–256.
- [39] G. Grieser, K. P. Jantke, S. Lange, and B. Thomas, "A Unifying Approach to HTML Wrapper Representation and Learning," *Discovery Science, DS 2000*, vol. 1967, Dec 2000, pp. 50–64.
- [40] J.-R. Gruser, L. Raschid, M. E. Vidal, and L. Bright, "Wrapper Generation for Web Accessible Data Sources," *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, New York City, New York, Aug 1998, pp. 14–23, IEEE Computer Society.
- [41] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang, "Optimizing Queries Across Diverse Data Sources," *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB'97*, Athens, Greece, Aug 1997, pp. 276–285.
- [42] L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope, "DiscoveryLink: A system for integrated access to life sciences data sources," *IBM Systems Journal*, vol. 40, no. 2, 2001, pp. 489–511.
- [43] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. M. Breunig, and V. Vassalos, "Template-Based Wrappers in the TSIMMIS System," *Proceedings ACM*

SIGMOD International Conference on Management of Data, Tucson, Arizona, May 1997, pp. 532–535, ACM Press.

- [44] J. Han, H. M. Jamil, Y. Lu, L. Chen, Y. Liao, and J. Pei, “DNA-Miner: A System Prototype for Mining DNA Sequences,” *Proceedings ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, May 2001.
- [45] W. Han, “Wrapper Application Generation for Semantic Web: An XWRAP Approach,” *Ph.D. dissertation, College of Computing, Georgia Institute of Technology*, 2003.
- [46] H. M. Jamil, “Achieving Interoperability of Genome Databases Through Intelligent Web Mediators,” *Proc. IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, Washington, DC, Nov 2000.
- [47] V. Josifovski, P. M. Schwarz, L. M. Haas, and E. Lin, “Data Replication for Mobile Computers,” *Proceedings ACM SIGMOD International Conference on Management of Data*, 2002, pp. 524 – 532.
- [48] P. Karp, M. Riley, M. Saier, I. Paulsen, J. Collado-Vides, S. Paley, A. Pellegrini-Toole, C. Bonavides, and S. Gama-Castro, “The EcoCyc Database,” *Nucleic Acids Research*, vol. 30, no. 1, 2002, pp. 56–58.
- [49] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada, “The ARIADNE Approach to Web-based Information Integration,” *International Journal of Cooperative Information Systems (IJCIS)*, vol. 10, no. 1-2, 2001, pp. 145–169.
- [50] N. Kushmerick, “Wrapper induction: Efficiency and expressiveness,” *Artificial Intelligence*, vol. 118, no. 1-2, 2000, pp. 15–68.
- [51] N. Kushmerick, “Wrapper verification,” *World Wide Web*, vol. 3, no. 2, 2000, pp. 79–94.
- [52] N. Kushmerick, “Wrapper Induction for Information Extraction,” *Ph.D. dissertation, Dept of Computer Science & Engineering, Univ of Washington*, 2003.
- [53] C. T. Kwok and D. S. Weld, “Planning to Gather Information,” *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, vol. 1, Aug 1996, pp. 32–39.
- [54] Z. Lacroix, “Biological Data Integration: Wrapping Data and Tools,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 6, no. 2, Jun 2002, pp. 123–128.

- [55] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira, "A Brief Survey of Web Data Extraction Tools," *ACM SIGMOD Record*, vol. 31, no. 1, 2002, pp. 84–93.
- [56] K. Lerman, S. Minton, and C. A. Knoblock, "Wrapper maintenance: A machine learning approach," *To appear in the Journal of Artificial Intelligence Research*, 2003.
- [57] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions," *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, Mumbai (Bombay), India, Sep 1996, pp. 251–262.
- [58] L. Liu, C. Pu, and W. Han, "XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources," *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, San Diego CA, 2000.
- [59] LocusLink, "LocusLink at NCBI," <http://www.ncbi.nlm.nih.gov/LocusLink/>, accessed on Feb 2004.
- [60] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni, "The Araneus Web-Base Management System," *Proceedings ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, Jun 1998, pp. 544–546, ACM Press.
- [61] A. O. Mendelzon and T. Milo, "Formal Models of Web Queries," *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997, pp. 134–143, ACM Press.
- [62] I. Muslea, "Active learning with multiple views," *Ph.D. dissertation, University of Southern California*, 2002.
- [63] I. Muslea, S. Minton, and C. Knoblock, "Hierarchical Wrapper Induction for Semistructured Information Sources," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 1/2, 2001, pp. 93–114.
- [64] I. Muslea, S. Minton, and C. Knoblock, "Active + Semi-Supervised Learning = Robust Multi-View Learning," *Proceedings of the 19th International Conference on Machine Learning (ICML 2002)*, 2002, pp. 435–442.
- [65] NCBI, "National Center for Biotechnology," <http://www.ncbi.nlm.nih.gov/>, access in Mar 2004.
- [66] A. Pan, J. Raposo, M. Álvarez, P. Montoto, V. Orjales, J. Hidalgo, and A. V. n. Lucía Ardao, Anastasio Molano, "The Denodo Data Integration Platform," *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

- [67] A. Papoulis, *Probability and Statistics*, Prentice Hall, Englewood Cliffs, NJ, 1990.
- [68] N. W. Paton, R. Stevens, P. Baker, C. A. Goble, S. Bechhofer, and A. Brass, "Query Processing in the TAMBIS Bioinformatics Source Integration System," *11th international conference on scientific and statistical database management*, Z. M. Ozsoyoglu, G. Ozsoyoglu, and W. Hou, eds., Cleveland, Ohio, Jul 1999, pp. 138–147.
- [69] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences of the USA*, 1988, vol. 85, pp. 2444–2448.
- [70] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom, "Querying Semi-structured Heterogeneous Information," *International Conference on Deductive and Object-Oriented Databases (DOOD)*, Singapore, Dec 1995, pp. 319–344.
- [71] A. Rajaraman, Y. Sagiv, and J. D. Ullman, "Answering Queries Using Templates with Binding Patterns," *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, California, May 1995, pp. 105–112, ACM Press.
- [72] J. Robie, J. Lapp, and D. Schach, "XML Query Language (XQL)," *WWW The Query Language Workshop (QL)*, Cambridge, MA, Dec 1998, <http://www.w3.org/TandS/QL/QL98/pp/xql.html>.
- [73] Sacch3D, "Sacch3D: Structural Information for Yeast Proteins," <http://genome-www.stanford.edu/Sacch3D/>, access on Feb 1, 2002.
- [74] A. Sahuguet and F. Azavant, "Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F," *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, M. P. Atkinson, M. E. Orłowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, eds., Edinburgh, Scotland, UK, Sep 1999, pp. 738–741, Morgan Kaufmann.
- [75] A. Sahuguet and F. Azavant, "Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F," *Proceedings of 25th International Conference on Very Large Data Bases, VLDB'99*, Edinburgh, Scotland, UK, Sep 1999, pp. 738–741, Morgan Kaufmann.
- [76] P. Seshadri, "Enhanced Abstract Data Types in Object-Relational Databases," *VLDB Journal*, vol. 7, no. 3, 1998, pp. 130–140.
- [77] A. P. Sheth and J. A. Larson, "BioKleisli: A Digital Library for Biomedical Researchers," *International Journal on Digital Libraries*, vol. 1, no. 1, 1997, pp. 36–53.

- [78] A. P. Sheth and R. Meersman, "Amicalola Report: Database and Information System Research Challenges and Opportunities in Semantic Web and Enterprises," *ACM SIGMOD Record*, vol. 31, 2002, pp. 98–106.
- [79] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Molecular Biology*, vol. 147, 1981, pp. 195–197.
- [80] R. Stevens, C. Goble, N. W. Paton, S. Bechhofer, G. Ng, P. Baker, and A. Brass, "Complex Query Formulation Over Diverse Information Sources in TAMBIS," *Bioinformatics: Managing Scientific Data*, May 2003.
- [81] B. Sturgeon, D. McCourt, J. Cowper, F. Palmer, S. MaClean, and W. Dubitzky, "Can the Grip Help to Solve the Data Integration Problems Molecular Biology?," *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and Grid(CCGRID'03)*, Tokyo, Japan, May 2003, pp. 594–600.
- [82] J. D. Thompson, T. J. Gibson, F. Plewniak, F. Jeanmougin, and D. G. Higgins, "The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools," *Nucl. Acids. Res.*, vol. 25, no. 24, 1997, pp. 4876–4882.
- [83] D. L. Wheeler, D. M. Church, S. Federhen, A. E. Lash, T. L. Madden, J. U. Pontius, G. D. Schuler, L. M. Schriml, E. Sequeira, T. A. Tatusova, and L. Wagner, "Database resources of the National Center for Biotechnology," *Nucleic Acid Research*, vol. 31, no. 1, 2003, pp. 28–33.
- [84] L. Wong, "Kleisli, its Exchange Format, Supporting Tools, and an application in Protein Interaction Extraction," *IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, Arlington, Virginia, Nov 2000, pp. 21–28.
- [85] R. Yerneni, C. Li, J. D. Ullman, and H. Garcia-Molina, "Optimizing Large Join Queries in Mediation Systems," *Proceedings of the 7th International Conference on Database Theory - ICDT '99*, Jerusalem, Israel, Jan 1999, vol. 1540 of *Lecture Notes in Computer Science*, pp. 348–364, Springer.