

Intelligent Integration of Information Design and Implementation of a Web Wrapper

by

Kenneth C. Lau

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements of the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 1999

© Copyright 1999 Kenneth C. Lau. All rights reserved.

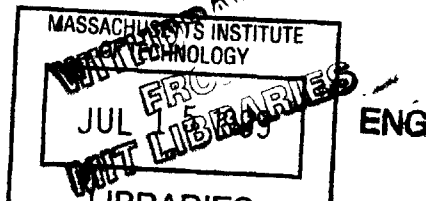
The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author [Signature] Department of Electrical Engineering and Computer Science
May 19, 1999

Certified by [Signature] Prof. Stuart Madnick
Co-Thesis Supervisor

Certified by [Signature] Dr. Michael Siegel
Co-Thesis Supervisor

Accepted by [Signature] Arthur C. Smith
Chairman, Department Committee on Graduate Thesis



Intelligent Integration of Information Design and Implementation of a Web Wrapper

by

Kenneth C. Lau

Submitted to the Department of Electrical Engineering and Computer Science
In partial fulfillment of the requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 1999

Abstract

This thesis describes the design and the implementation of a wrapper engine to query semi-structured remote data sources for the Context Interchange group. Research is focused on the continued development of the query execution engine, and the design and implementation of a planner to structure and optimize user queries. Major problems dealt with include the automatic extraction of data, manipulation of information using relational operations, search for plans to order sub-query executions, circumvention of network delays, and composition of results to be returned to the receiver of the mediated query.

Co-Thesis Supervisor: Stuart Madnick
John Norris Maguire Professor of Information Technology
Sloan School of Management

Co-Thesis Supervisor: Michael Siegel
Principal Research Scientist
Sloan School of Management

Table of Content

Table of Content	3
Figures and Tables	5
Acknowledgement	6
Chapter 1	7
Introduction.....	7
1.1 The COIN Framework	7
1.1.1 Client Processes	8
1.1.2 Server Processes.....	8
1.1.3 Mediator Processes	9
1.2 Wrapper Generator.....	9
1.3 Goal of the Thesis	9
1.4 Motivation.....	10
1.4 Organization of Thesis.....	14
Chapter 2.....	15
Literature Review.....	15
2.1 Context Mediation	15
2.1.1 The Tight-Coupling Strategy	15
2.1.2 The Loose-Coupling Strategy	16
2.1.3 Context Interchange Strategy.....	16
2.2 Wrapper Technology.....	17
2.2.1 COIN Architecture.....	18
2.2.2 Wrapper Generation – University of Maryland	19
2.2.3 Wrapper Technology – University of Alberta	20
2.2.4 WebL – A Programming Language on the Web	22
2.2.5 Medved’s Quote Tracker	23
2.2.6 Alpha Microsystems StockVue 99.....	23
Chapter 3.....	25
Wrapper Engine	25
Chapter 4.....	27
Query Execution	27
4.1 Relational Algebra	27
4.1.1 Select and Project.....	27
4.1.3 Set Operations: Union, Intersection, Set-Difference, Cross-Product	27
4.1.4 Other Operators.....	28
4.1.5 Relational Completeness.....	28
4.2 Execution Tree	28
4.3 Relational Operators	30
4.3.1 Select.....	30
4.3.2 Project	30
4.3.3 Join.....	30
4.3.4 Union.....	31
4.4 Physical Operators	31

4.4.1 Submit	31
4.4.2 Join-Submit	32
4.4.3 Regular-Expression	32
Chapter 5	33
Planner	33
5.1 Specification File	33
5.1.1 Header	34
5.1.2 Body	34
5.2 Ordering Sources	35
5.3 Planning	36
Chapter 6	38
Implementation Details	38
6.1 Implementation Language	38
6.2 Modules	38
6.2.1 Network Access Module	39
6.2.2 Boolean Operations Module	40
6.2.3 Data Module	41
6.2.4 Pattern Matching Module	42
6.2.5 Relational Operations Module	42
6.2.6 SQL Parser Module	43
6.2.7 Specification File Parser Module	43
6.2.8 Planner Module	43
6.2.9 Interface Module	44
6.3 Scheduling Threads	45
6.4 Concurrent Execution	46
6.5 Dynamic Optimization	47
Chapter 7	49
Conclusion and Future Research	49
7.1 System Evaluation	49
7.2 Future Research and Concluding Thoughts	49
References	51
Appendix A: User's Manual	53
Appendix B: Sample Specification Files	54
Appendix C: Query Trace	56

Figures and Tables

Figure 1.1: COIN Framework.....	8
Figure 1.2: Query 1 Results	12
Figure 1.3: USA Today - Trading Prices for IBM; Fastquote - Headlines for IBM.....	12
Figure 1.4: Query 2 Results	13
Figure 1.5: Yahoo - Trading Price, Time for Intel.....	13
Figure 2.1: COIN Wrapper Architecture	18
Figure 2.2: Maryland Architecture	20
Figure 2.3: University of Alberta Architecture.....	21
Table 2.1: Summary of Findings	24
Figure 3.1: Wrapper Architecture	25
Figure 4.1: Execution Tree	29
Figure 6.1: Module Dependency.....	39
Figure 6.2: Accessdata Classes Inheritance Tree.....	40
Figure 6.3: Bool_op Classes Inheritance Tree.....	41
Figure 6.4: Data Classes Inheritance Tree	42
Figure 6.5: Relational_op Classes Inheritance Tree	43
Figure 6.6: Interface of the Wrapper Engine	44
Figure 6.7: Scheduler	45
Figure 6.8: Concurrency Model.....	46

Acknowledgement

I would like thank my thesis supervisors, Professor Stuart Madnick and Dr. Michael Siegel, for giving me the opportunity to work in the Context Interchange group. The group provides an excellent environment to conduct research. Particular thanks to Aykut Firat and Tom Lee for all their help and encouragement.

Thank you to my many friends at MIT for making my life sometimes difficult, but always interesting and enjoyable. Finally, I would like to thank my parents and my brother for their love and their support.

Chapter 1

Introduction

In the past several years, advances in computer networking and telecommunications have led to the explosive growth in the number of information sources that are being connected. This connectivity has given way to a proliferation of data stored in various databases in a distributed fashion. With this growing abundance of data, the problem of retrieving information and interpreting information becomes an important challenge.

The ability to exchange information physically has taken giant steps forward with the explosive growth in computer networking, but the ability to exchange information in a meaningful manner has lagged significantly behind. The meaning of information is often dependent on a particular context - a context which embodies a number of underlying assumptions. For example, "07-04-98" might mean 4th of July in the US, but in Europe it means April the 7th. Different context of data lead to possible confusion and conflicts when the information from heterogeneous sources is brought together. This problem is referred to as the need for semantic interoperability among distributed data source; any data integration effort must be capable of reconciling possible semantic conflicts [4].

1.1 The COIN Framework

The Context Interchange (COIN) project seeks to mediate the problem of interoperability using a context mediator [4]. This mediator sits between the users of data and the sources. The meaning and underlying assumptions about the data set are explicitly represented as data contexts. When the need to bring together data of different context arises, it is the job of the context mediator to determine semantic conflicts between contexts and apply any transformations necessary to exchange the data in a meaningful way.

The mediator uses the Structured Query Language (SQL) as a means of issuing queries to data sources. Mediation is the process of rewriting queries posed in the receiver's or the client's context into a new set of queries where all potential conflicts are explicitly solved. Each query issued by the receiver is translated to a source context and then passed on to the data source. The data returned is similarly translated to the receiver's context and presented to the receiver. This process allows users to extract disparate data from different sources and produce consistent results. Figure 1.1 shows an overview of the COIN framework [3].

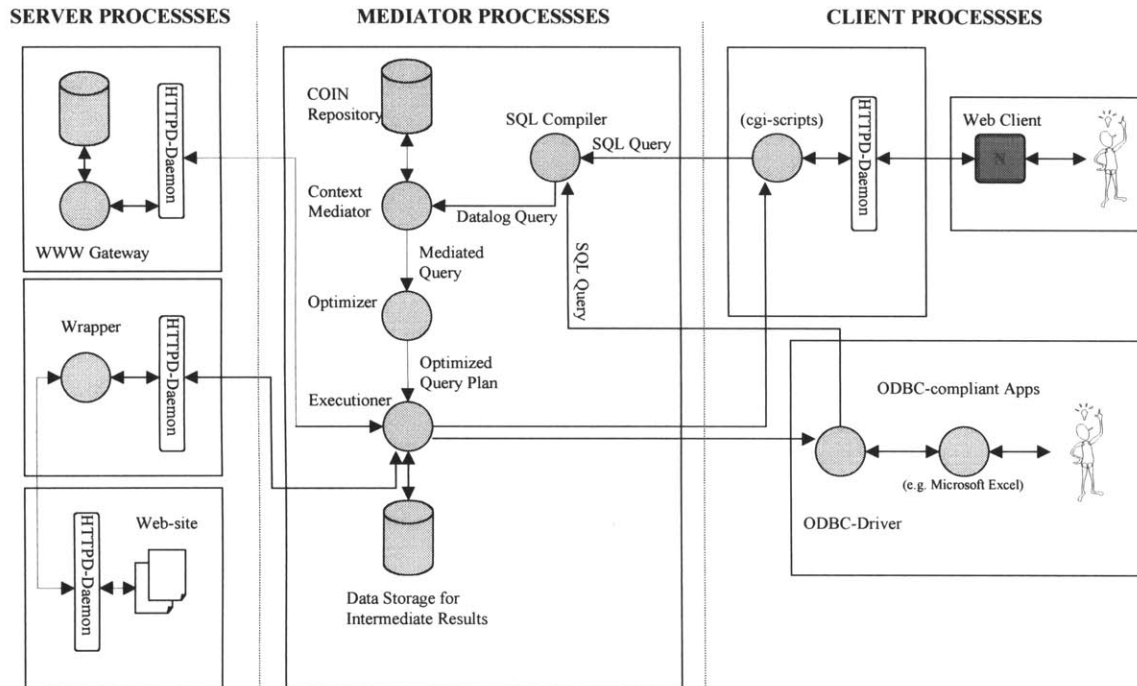


Figure 1.1: COIN Framework

1.1.1 Client Processes

Client processes provide the interaction with receivers and route all database requests to the Context Mediator [3]. An example of a client process is the multi-database browser, which provides a point-and-click interface for formulating queries to multiple sources and for displaying the answers obtained. Specifically, any application program that posts queries to one or more sources can be considered a client process. This can include all the programs (e.g. spread sheet software programs like Excel or Access) that can communicate using the ODBC bridge to send SQL queries and receive results.

1.1.2 Server Processes

Server processes refer to *database gateways* and *wrappers*. Database gateways provide physical connectivity to databases on a network. The goal is to insulate the Mediator Process from the idiosyncrasies of different database management systems by providing a uniform protocol for database access as well as canonical query language (and data

model) for formulating the queries. Wrappers, on the other hand, provide richer functionality by allowing semi-structured documents on the World Wide Web to be queried as if they were relational databases [3]. This is accomplished by defining an *export schema* for each of these web sites and describing how attribute-values can be extracted from a web site using pattern matching.

1.1.3 Mediator Processes

Mediator processes refer to the system components that collectively provide the mediation services. These include SQL-to-datalog compiler, context mediator, and query planner/optimizer and multi-database executioner. SQL-to-datalog compiler translates a SQL query into its corresponding datalog format. Context mediator rewrites the user-provided query into a mediated query with all the conflicts resolved. The planner/optimizer produces a query evaluation plan based on the mediated query. The multi-database executioner executes the query plan generated by the planner. It dispatches sub-queries to the server processes, collates the intermediary results, and returns the final answer to the client processes.

1.2 Wrapper Generator

Client applications interact with data sources and route all database requests to the context mediator by issuing SQL queries. An example of a client process is the multi-database browser, which provides a point-and-click interface for formulating queries to multiple sources and for displaying the answers obtained. When the data sources do not comply with the relational database model, the SQL query is translated to the native query format of the source. This gives the client a homogeneous way of querying sources which might not be relational databases, but semi-structured documents on the World Wide Web, for example.

The wrapper generator is an engine that allows for the incorporation of the data originating from Web sites [8]. The Wrapper automatically performs all the steps necessary to obtain data values from the site. It issues commands directly to data servers, thus mimicking the interaction between a human user and the World Wide Web site.

1.3 Goal of the Thesis

The goal of the thesis is to seek and identify the major design decisions taken for the implementation of a wrapper generator. The wrapper generator takes a list of accessible sources, executes the queries to those sources, and combines the result into a table to be presented to the receiver. My contributions are as follow:

- (i) research on current wrapper generation technology;
- (ii) continued development of the query execution engine (which was originally designed and partially built last year to be a robust, and easy to migrate execution model);

- (iii) design and implementation of a planner/optimizer that is responsible for planning the order of execution of the user's SQL query;
- (iv) design and implementation of a friendly interface to interact with the various components of the wrapper engine; and
- (v) documentation on the current status of the wrapper technology within the Context Interchange Group.

1.4 Motivation

Consider the following example as a motivational scenario - a businessman who regularly updates his stock portfolio. Current and historical information on the stock movement can be tracked and visualized in a spreadsheet. To keep this information up to date, the businessman will need to collect the relevant information of the stock on a daily basis and then update the spreadsheet. The New York Stock Exchange (NYSE) provides a very useful web site with pages for each of the companies trading on the exchange. These pages hold information such as the last stock price and the company ticker. Yahoo stock pages show the price, percentage and point changes, the volume sold for the trading period. Bloomberg pages provide news headlines for each companies. USA Today pages provide the high and low of the stock price during the trading day. This is a simple, hypothetical scenario. There are web sites that provide much of this information, but the scenario serves to illustrate the motivation for this research effort.

Although the information is very easily accessible, the only way to get to this information is to manually browse and click to load up the required pages. For a single company, this may not be very difficult but when dealing with a large portfolio, the manual gathering of the data becomes a bit more tedious. An even graver problem occurs if the search set is not predefined to certain companies. Below, two web-intensive data-extraction situations will be described and the procedures, which can be used to automate this extraction, will be highlighted.

The first example, the businessman would like to get the names of companies, their tickers, last selling price, high and low trading price during the trading day, and company headlines from a portfolio of stocks. This information is provided by Fastquote and USA Today pages. This example is quite manageable, however, and the process can be automated, as it is a monotonous everyday activity. The query (representing the example) and the results returned by the wrapper engine are illustrated in Figure 1.2. For comparison, the screen shots of the actual web sites from which the information is collected are shown in Figure 1.3.

The second example, the businessman would like to get the ticker, the last trading price and the time the quote is obtained from a portfolio providing that the recent stock price is less than \$100. This example is quite similar to the first one but the additional clause illustrates the use of relational operations. Information on the ticker, the last trading price, and the time is provided by Yahoo pages. Relational operations are then performed to return the appropriate results (trading price less than \$100) to the receiver. The query

(representing the example) and the results returned by the wrapper engine are illustrated in Figure 1.4. For comparison, the screen shots of the actual web sites from which the information is collected are shown in Figure 1.5.

Solving the problems above requires an intermediary agent that will allow the user to formulate a structured request. The agent will automatically extract the data from the required sources to manipulate the data to produce the desired results and then present this result to the user [1]. The automatic nature of such a system will definitely be limited by the varying structures of stored data e.g. if one requires data from both a web site and an Excel spreadsheet to produce the result of the request. The problems are even clearer when dealing with two different web sites, where data may be stored in completely different formats.

The sites can easily be seen as being tables (providing certain specific attributes), but automating the extraction remains a problem due to the varying structures of the sites. Wrapping the sources can hide these structural differences. Using an object that knows how to access a source and extract the information can then provide information on that particular source. It's clear that the main information needed here is the list of attributes provided along with the method of extraction; in this system this information is stored in a source specification file. The make up of the specification file is explained in detail in a later chapter.

Query 1

```

SELECT Ticker, LastTrade, High, Low, Headline
FROM usatoday, fastquote
WHERE Ticker in ["AAPL", "IBM"]
    
```

Results

Ticker	LastTrade	High	Low	Headline
IBM	209.1875	211.0000	203.4375	IBM Design C
IBM	209.1875	211.0000	203.4375	New IBM S/G
IBM	209.1875	211.0000	203.4375	WALL ST WE
IBM	209.1875	211.0000	203.4375	RealNetworks
IBM	209.1875	211.0000	203.4375	IBM To Unvel
IBM	209.1875	211.0000	203.4375	IBM mail
IBM	209.1875	211.0000	203.4375	IBM may testi
IBM	209.1875	211.0000	203.4375	U.S. names 1
IBM	209.1875	211.0000	203.4375	SmartPortfoli
AAPL	46.0000	47.1250	44.0000	SmartPortfoli
AAPL	46.0000	47.1250	44.0000	DIARY - U.S.
AAPL	46.0000	47.1250	44.0000	Singapore-Ba
AAPL	46.0000	47.1250	44.0000	ValleyJobs.co

Figure 1.2: Query 1 Results

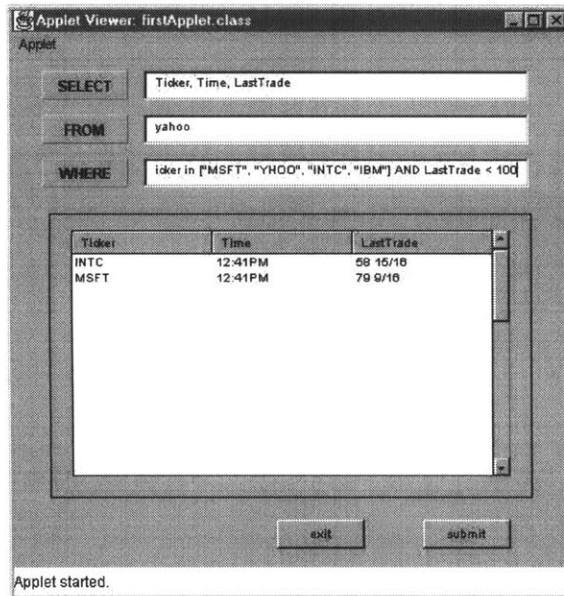
Screen Shots of Actual Web Pages

Figure 1.3: USA Today - Trading Prices for IBM; Fastquote - Headlines for IBM

Query 2

```
SELECT Ticker, Time, LastTrade
FROM yahoo
WHERE Ticker in ["MSFT", "YHOO", "INTC", "IBM"]
AND LastTrade < 100
```

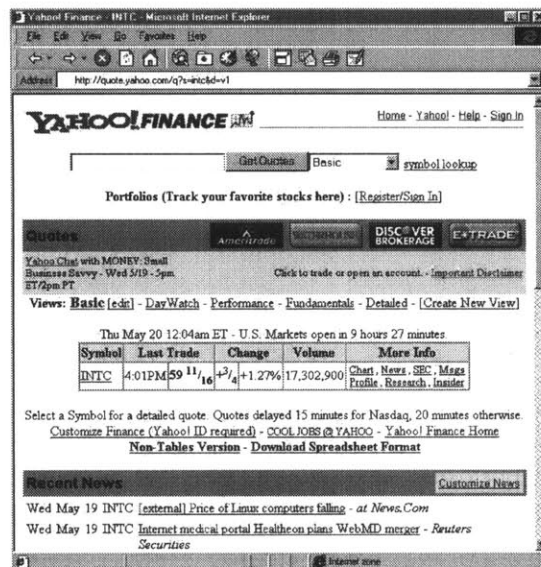
Results



Ticker	Time	LastTrade
INTC	12:41PM	58 15/16
MSFT	12:41PM	79 9/16

Figure 1.4: Query 2 Results

Screen Shot of Actual Web Page



Symbol	Last Trade	Change	Volume	More Info
INTC	4:01PM 59 11/16	+1 27%	17,302,900	Chart, News, SEC, Mega Profile, Research, Insider

Figure 1.5: Yahoo - Trading Price, Time for Intel

1.4 Organization of Thesis

This thesis is divided into seven chapters and several appendices.

- Chapter 1 gives an overview of the research conducted at the Context Interchange Group at the Massachusetts Institute of Technology, as well as provides a brief motivational example to illustrate the purpose and the goal of this research effort.
- Chapter 2 is a thorough literature review on the current technology, both in academia and in the marketplace, of querying semi-structured data from the World Wide Web.
- Chapter 3 briefly summarizes the wrapper engine and the architecture of its design.
- Chapter 4 provides a detail summary of the query execution engine as designed and implemented.
- Chapter 5 looks into the design of the planner and the optimizer that was implemented.
- Chapter 6 provides all the implementation details of the engine and the decisions made to satisfy the various design criteria.
- Chapter 7 is the conclusion of the thesis, giving some insights into how the system can be improved, directions for future research and the limitation of the current technology.
- The list of references and appendices at the end provide various technical annotations such as a user's manual of the engine application, examples of specification files, and system trace during engine execution.

Chapter 2

Literature Review

The goal of this chapter is to provide a thorough literature review consisting of a brief but comprehensive investigation of the approaches towards context mediation, and an extensive analysis on web wrapper technology. The review on context mediation serves as an initial motivation for the research. Given the distributed nature of information storage within the World Wide Web, semantic interoperability is unavoidable. However, the decentralized system of information storage is still superior to the alternative of a central database - hence there is a need for context mediation. With the initial motivation, our focus is narrowed on web wrapper technology. A web wrapper is an engine that allows for the incorporation of the data originating from public World Wide Web sites. The wrapper automatically performs all the necessary steps to obtain data values from different distributed data sources, which in essence mimics the interaction that would normally take place between a human user and a public web site. This literature review and the analysis on academic and marketplace web wrapper technology in this section lay the foundation for the rest of this research effort.

2.1 Context Mediation

Within the last decade, there has been a proliferation of proposals and research prototypes aimed at achieving interoperability among autonomous and heterogeneous databases. Primarily, these proposals differ from one another in either the choice of the underlying data model for conflict resolution or the subscription to a tight-coupling or a loose-coupling integration strategy [6]. With respect to the choice of the underlying data model, semantically rich data models have gained greater popularity over traditional systems. The distinction between tight-coupling and loose-coupling systems, on the other hand, can be characterized by:

- who is responsible for identifying what conflicts exists and how they can be circumvented; and
- when the conflicts are resolved.

2.1.1 The Tight-Coupling Strategy

In the case of tightly-coupled systems, the detection of conflicts is performed by a system administrator and the actual resolution is accomplished by defining, a priori, one or more views, which define the shared schemas for the system. A shared schema insulates the receiver from underlying data heterogeneity. Queries formulated against a shared schema can be transformed to sub-queries, which are submitted to component sources.

Conflicts in underlying sources are encapsulated via the introduction of a supertype, which has methods or functions, which are defined with reference to its subtypes [6]. For instance, consider the following conflict for student grades reported by two databases: the first database represents student grades using letter grades, and the second represents the same as points in the range of 0 to 100. In a tightly-coupled systems, this integration can be accomplished by introducing a supertype which integrates the two student types and allow all attributes of the subtypes to be inherited. Hence, if the attribute Name is common to Student1 and Student2, the invocation of method Name on Student will be automatically translated to the invocation of Name on one of the subtypes. Semantic conflicts are circumvented by providing the necessary conversion functions to effect the translation.

2.1.2 The Loose-Coupling Strategy

Systems constructed using the loose-coupling approach, on the other hand, subscribe to the belief that the creation and maintenance of shared schemas is infeasible for any nontrivial number of sources. Hence, instead of resolving all conflicts a priori, conflict detection and resolution are undertaken by receivers themselves, who need only interact with a limited subset of the sources at any one time [6]. To facilitate this task, research has focused on the invention of data manipulation languages, which are sufficiently expressive so that queries on multiple sources can be interleaved with operations for effecting data transformations.

2.1.3 Context Interchange Strategy

In reality, most integration systems fall between the continuum, and few would venture to the extremes. The context interchange strategy combines the best features of existing loose-coupling and tight-coupling approaches to semantic interoperability among autonomous and heterogeneous systems. It allows the complexity of the system to be harnessed in small chunks, by enabling sources and receivers to remain loosely-coupled to one another, and by sustaining an infrastructure for data integration.

The context interchange strategy seeks to address the problem of semantic interoperability by using both a data model and a logical language [13]. The data model and language are used to define the domain model, which is a collection of semantic elements within the receiver and the data source, and the context associated with them. The data model contains the definitions for the types of information units that constitute a common vocabulary for capturing the semantics of data in disparate systems. Contexts, associated with both information sources and receivers, are collections of statements defining how data should be interpreted and how potential conflicts should be resolved.

The modular design of the strategy, both in components and protocol, keeps the infrastructure scalable, extensible, and accessible [13]. Scalability means that the complexity of creating and administering the mediation services does not increase exponentially with the number of participating sources and receivers. Extensibility refers to the ability to incorporate changes into the system in a graceful manner. Accessibility refers to how a user in terms of its ease-of-use perceives the system and the flexibility in supporting a variety of queries.

2.2 Wrapper Technology

Typically, queries have access to a data source to get the desired information. The data source used can be accessed by anyone on the World Wide Web. In order to access web sources, a technology is developed that lets users treat web sites as relational data sources. The users then issue SQL queries just as they would to any relation in a relational database, thus combining multiple sources and creating queries as the one above. This technology is called web wrapping and an implementation for this technology is called a wrapper engine. Using a web wrapper engine, application developers can very rapidly wrap a structured or semi-structured web site and export the schema for the users to query against.

This section provides an overview of current web wrapper technology, both in the marketplace and in the academia. The section begins with a description of the wrapper technology used by the COIN project. Besides the COIN architecture developed at MIT, there are many other different technologies employed for data wrapper from the World Wide Web. Traditionally, these are first developed from academic institutions, including the ones developed at the University of Maryland and the University of Alberta. As the growth of the World Wide Web continues and the need for commercial data wrapping manifests, commercial organizations begin to develop and market different wrapper engines, mostly derivations from ideas developed within the academia. These marketplace wrapper engines usually involve wrapping quotes or other business data; several such engines are included in the following sections as a review.

2.2.1 COIN Architecture

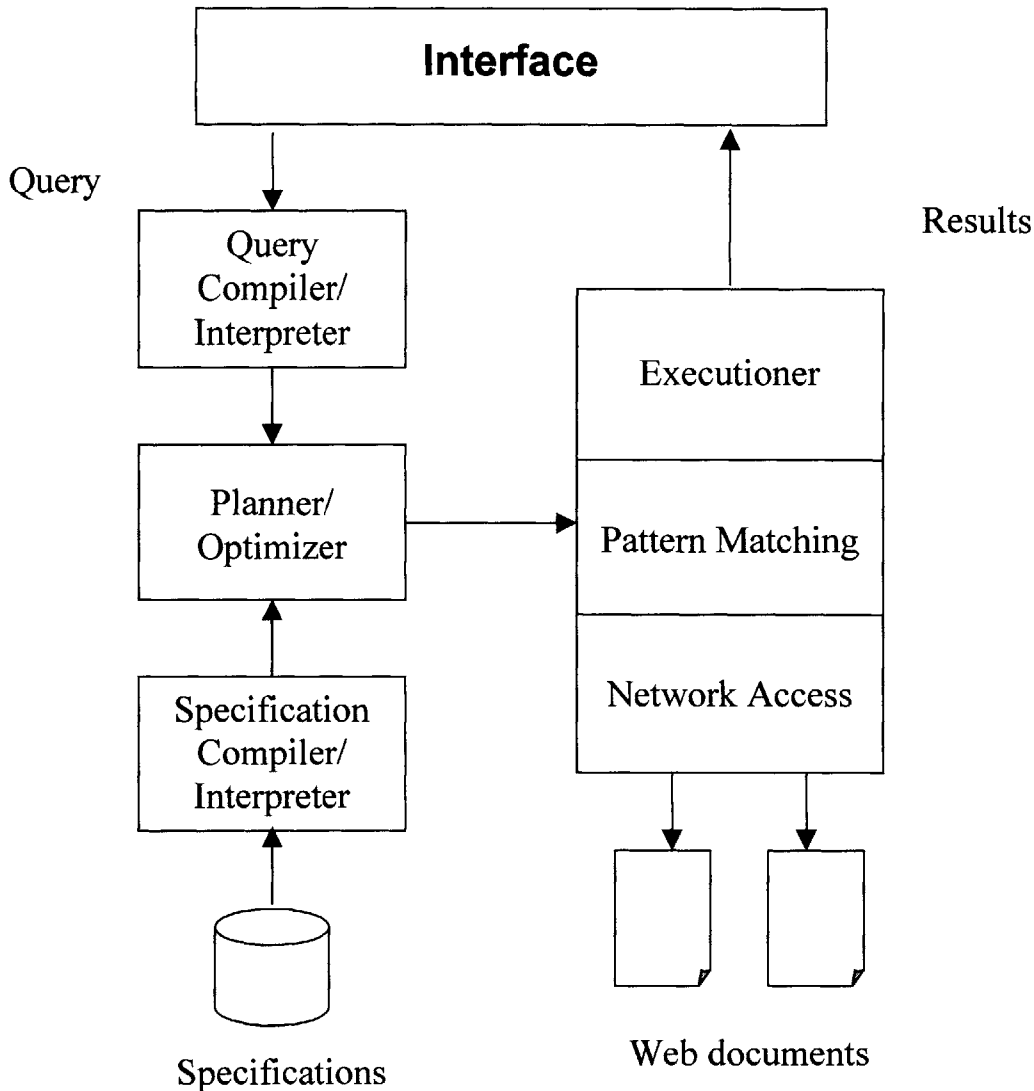


Figure 2.1: COIN Wrapper Architecture

The above figure shows the architecture of the COIN web wrapper. The system takes the SQL query as input. It parses the query along with the specifications for the given web site. A query plan is then constituted. The query plan contains a detailed list of web sites to send http requests, the order of those requests and the list of documents that will be fetched from those web sites. The executioner then executes the plan. Once the pages are fetched, the executioner then extracts the required information from the pages and presents the collated results to the user.

In order to wrap a site, you need to create a specification file. This file is plain text file and contains information like the exported schema, the URL of the web site to access and

a regular expression that will be used to extract the actual information from the web page. A simple specification file is included below:

```
#HEADER
#RELATION=quotes
#HREF=GET http://qs.cnnfn.com
#EXPORT= quotes.Cname quotes.Last
#ENDHEADER
#BODY
#PAGE
#HREF=POST http://qs.cnnfn.com/cgi-bin/stockquote?symbols=
    ##quotes.Cname##
#CONTENT=last:&nbsp;
    </font><FONTSIZE=+1><b>##quotes.Last##</FONT></TD>
#ENDPAGE
#ENDBODY
```

The specification has two parts, Header and Body. The Header part specifies information about the name of the relation and the exported schema. In the above case, the schema exported has two attributes, Cname, the name of the company, and Last, the latest quote. The Body portion of the file specifies how to actually access the page (as defined in the HREF field) and what regular expression to use (as defined in the CONTENT field).

Once the specification file is written and placed where the web wrapper can read it, the system is ready for use. Users can start making queries against the new relation that are just created.

2.2.2 Wrapper Generation – University of Maryland

The architecture of the University of Maryland model allows query translation and answer extraction from a single document. It typically requires that an exact address of the web site representing the HTML document containing the answers (or a script, which can provide the answers) must be produced [7].

The architecture differentiates between a simple extractor and a complex extractor. A simple extractor extracts the relevant data from the corresponding HTML document and produces answer objects. In many cases, however, this simple scenario is inadequate. For example, when answers of the same type must be extracted from multiple documents, a complex extractor would iteratively call other simple extractors to extract objects from each document. In other cases, a complex extractor is needed to use the output of another extract, in constructing another URL, or in constructing an answer object. Finally, when the output format of a document may not always be known a priori, as is common for many Web accessible sources, then a complex extractor would conditionally call other extractors.

With the Maryland architecture, specification files are unnecessary because the exact address of the site containing the information requested must be provided in order to wrap data. A URL as well as a pattern matching expression are included as part of the

query; the results returned are in a tabular format, similar to the results returned under the COIN architecture. Under the Maryland architecture, query planning and optimization are also unnecessary, resulting in an architecture that is simple and easy to implement. With simplicity, the Maryland architecture sacrifices performance; because an exact address must be provided in order to wrap information, this wrapper technology proves to be less general and less useful than the COIN technology. Figure 2.2 below shows the architecture of the Maryland wrapper.

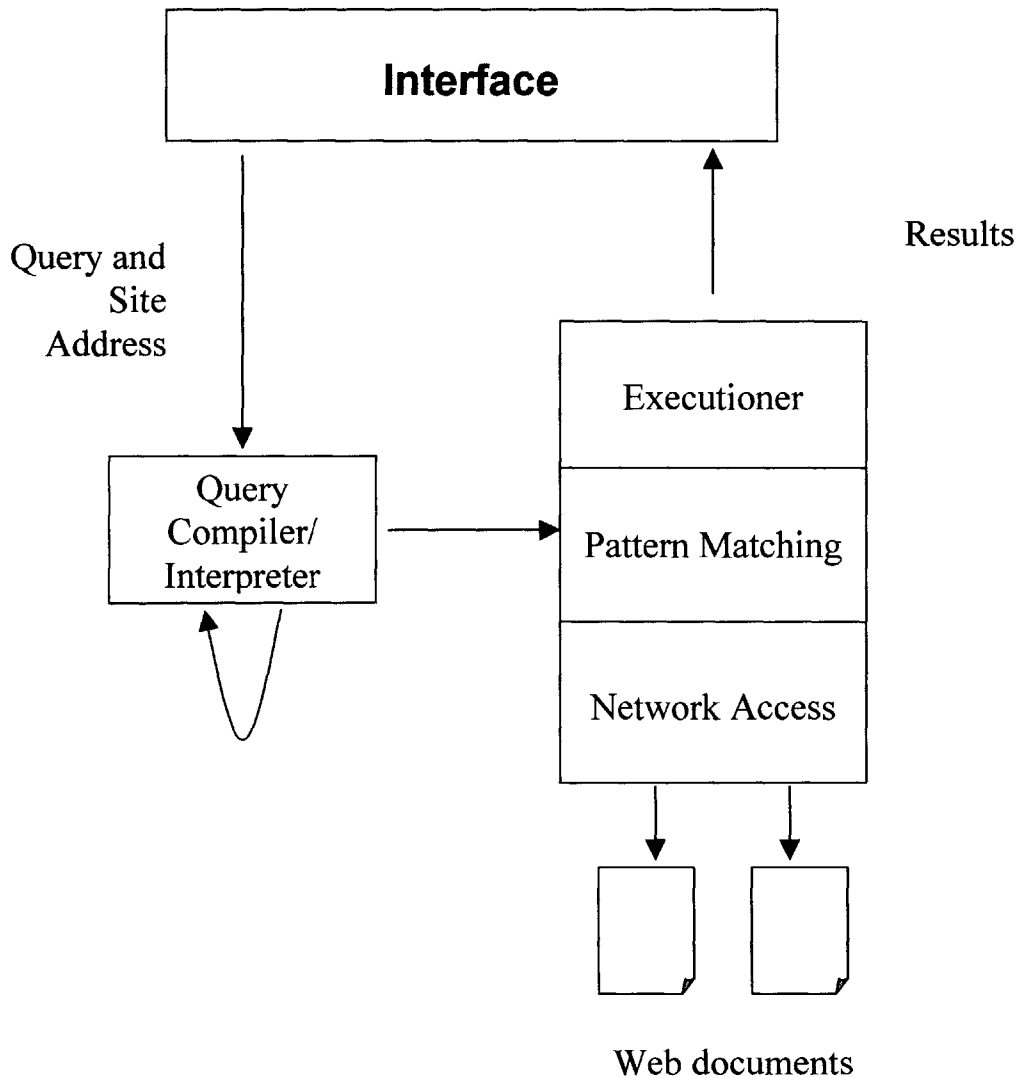


Figure 2.2: Maryland Architecture

2.2.3 Wrapper Technology – University of Alberta

The prototype of the wrapper technology developed by Lee at the University of Alberta in 1996 operates as a standard web-type application. There are HTML pages, scripts, and an underlying oracle database, which helps to process requests. The interface is created through the reading of a static HTML file (e.g. such as a form) or through the creation of

a dynamic web page through a perl script (CGI-compliant) [9]. Thus, the actual look and feel of the system is created through the linking of scripts and files.

Oracle is used as the underlying DBMS. It has two main functions: first relations are kept to store the data on the producer sources and consumer interfaces. The producer's need to store two tables: One table for the general description of a repository, and a second table for more synonyms of each keyword. The interfaces have two tables: (1) storage of the interface description and (2) storage of interface keyword synonyms.

Under this architecture, the relational database is not the entire World Wide Web but rather a pre-specified portion of it – in this case the Yahoo site. Simple querying of Yahoo is done through the `simple.query.cgi` script. It translates the form data into Yahoo URL calls, executes the URL calls in parallel, gets the return data, translates the data into Oracle tables, and returns an HTML table result. Thus, URL from users or specification files are not necessary in this implementation; instead they are replaced by Oracle tables stored at the source for both the producer (the source) and the consumer (Yahoo site). Figure 2.3 shows the wrapper architecture from the University of Alberta.

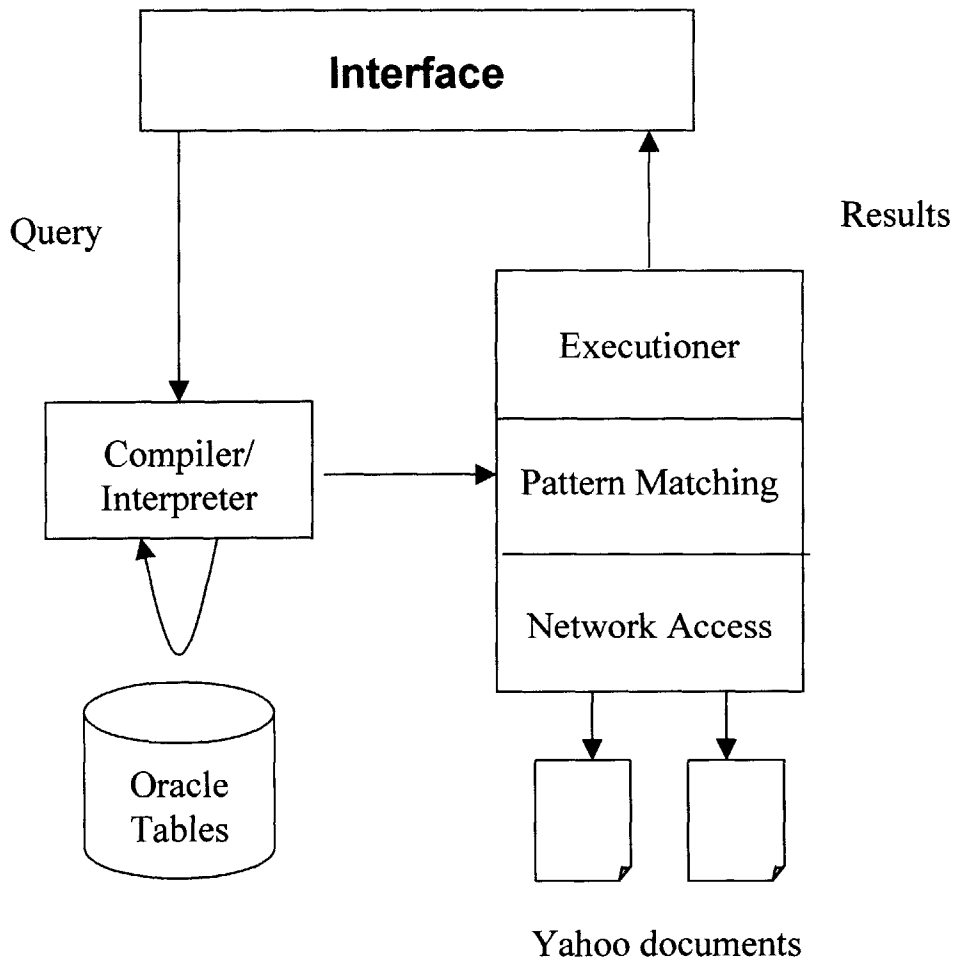


Figure 2.3: University of Alberta Architecture

2.2.4 WebL – A Programming Language on the Web

WebL is a web scripting language for processing documents on the World Wide Web. It is different from all the previous technologies because WebL is not an engine or an application to wrap information. Rather, it is a programming language used as a foundation where wrapper engines can be easily built. The language allows programmers an interface to extract information from specified web sites using Java and regular Perl expressions [11].

WebL is well suited for retrieving documents from the web, extracting information from the retrieved documents, and manipulating the contents of documents. In contrast to other general purpose programming languages, WebL is specifically designed for automating tasks on the web. Not only does the WebL language have a built-in knowledge of web protocols like HTTP and FTP, but it also knows how to process documents in plain text, HTML and XML format [11]. The flexible handling of structured text markup as found in HTML and XML documents is an important feature of the language. In addition, WebL also supports features that simplify handling of communication failures, the exploitation of replicated documents on multiple web services for reliability, and performing multiple tasks in parallel. WebL also provides traditional imperative programming language features like objects, modules, closures, control structures, etc.

To give a better idea of how WebL can be applied for web task automation, and what makes WebL different from other languages, it is instructive to discuss the computational model that underlies the language. In addition to conventional features you would expect from most languages, the WebL computation model is based on two new concepts, namely service combinators and markup algebra [11].

Service combinator is a formalism that can provide more reliable access to web resources and services. Very succinctly, service combinator is an exception handling mechanism that is powerful enough to encode robust behavior when communication failures occur. This concept is especially important for performing any reliable computation on the unreliable web structures. It often happens that web services are unavailable, suddenly fail or become unacceptably slow. These are very serious complications for computations that depend so much on the web infrastructure. Although service combinators cannot make a web-based computation completely failure-proof, it does add a certain amount of robustness to programming on the web.

Markup algebra is a formalism for extracting information from structured text documents and the manipulation of those documents [11]. It consists of functions to extract elements and patterns from web documents, operators to manipulate what has been extracted in this manner, and functions to change a page, for example to insert or delete parts. The functions and operators all work on the high-level concept of a parsed web page, and there is little need to do lower level string manipulation.

2.2.5 Medved's Quote Tracker

Medved's Quote Tracker is a commercial application that automatically tracks information regarding stock quotes, index changes and a variety of other data of publicly traded companies. The application represents a good way to collect a lot of information about a security without having to visit many different web sites.

The engine technology is similar to the COIN technology – users can choose to wrap information from a variety of different data sources such as E-Trade, Yahoo, and Realtime Quotes. The main drawbacks include

- inability to wrap information from non-specified sites
- unable to wrap information which are not inherently specified by the application

2.2.6 Alpha Microsystems StockVue 99

Alpha Microsystems' StockVue 99 is a Web-based application designed to automatically track stocks and mutual funds, using a variety of on-line resources. Knowledge workers, ranging from corporate finance officers to investors and traders, will find the application integrates many tools and functions they have used separately on a daily basis into a single, self-managing application. StockVue is able to track the following:

- stock quotes, including current value, current day's opening price, change, volume
- company news
- company SEC filings
- research data from Zacks Research
- number of shares owned and purchase price
- investment valuations
- portfolio performances

StockVue is able to export data to a variety of applications, including text files, word processors, spreadsheets, Quicken databases, Web pages, fax machines, e-mail addresses, even alphanumeric pagers. StockVue organizes information into multiple portfolios, with file folder views. It features a moving stock ticker and is self-updating through the built-in update manager.

2.2.7 Summary of Findings

<i>Architecture</i>	<i>Pros</i>	<i>Cons</i>
COIN Architecture	<ul style="list-style-type: none">• allow for access to the entire universe of semi-structured data sources on the web• simplicity in design• SQL-like queries are accepted and processed• ODBC compatible	<ul style="list-style-type: none">• reliance on specification files

<i>University of Maryland Architecture</i>	<ul style="list-style-type: none"> • simplicity in design • no need for specification file, or any other pattern matching mechanisms 	<ul style="list-style-type: none"> • performance is sacrificed • exact address and 'location' of data have to be provided
<i>University of Alberta Architecture</i>	<ul style="list-style-type: none"> • concurrent execution • no need to specify address and pattern mechanisms 	<ul style="list-style-type: none"> • limited to Yahoo pages • CGI scripts to do pattern matching defined in advance
<i>WebL</i>	<ul style="list-style-type: none"> • allow for access to the entire universe of semi-structured data sources on the web • can process HTTP, FTP, and XML pages 	<ul style="list-style-type: none"> • complicated 'programming language'
<i>Medved's Quote Tracker</i>	<ul style="list-style-type: none"> • rapid speed of execution 	<ul style="list-style-type: none"> • can only wrap information from inherently defined sites
<i>Alpha Microsystems Stockvue 99</i>	<ul style="list-style-type: none"> • rapid speed of execution • allows for data to be exported to a wide variety of user applications 	<ul style="list-style-type: none"> • can only wrap information from inherently defined sites
<i>Current Design and Implementation</i>	<ul style="list-style-type: none"> • allow for access to the entire universe of semi-structured data sources on the web • concurrent execution • relational operations can be performed • SQL-like queries are accepted and processed • rapid speed of execution 	<ul style="list-style-type: none"> • reliance on specification files

Table 2.1: Summary of Findings

Chapter 3

Wrapper Engine

The wrapper engine is divided into various parts: query and specification file compilers, planner and optimizer, and query execution. The focus of the thesis is the design and implementation of a query engine. In addition, the SQL and specification file compilers, the planner and optimizer, and an interface are also built to facilitate the construction and the use of the system. The design is modular in order to be easily manipulated and updated in future implementations.

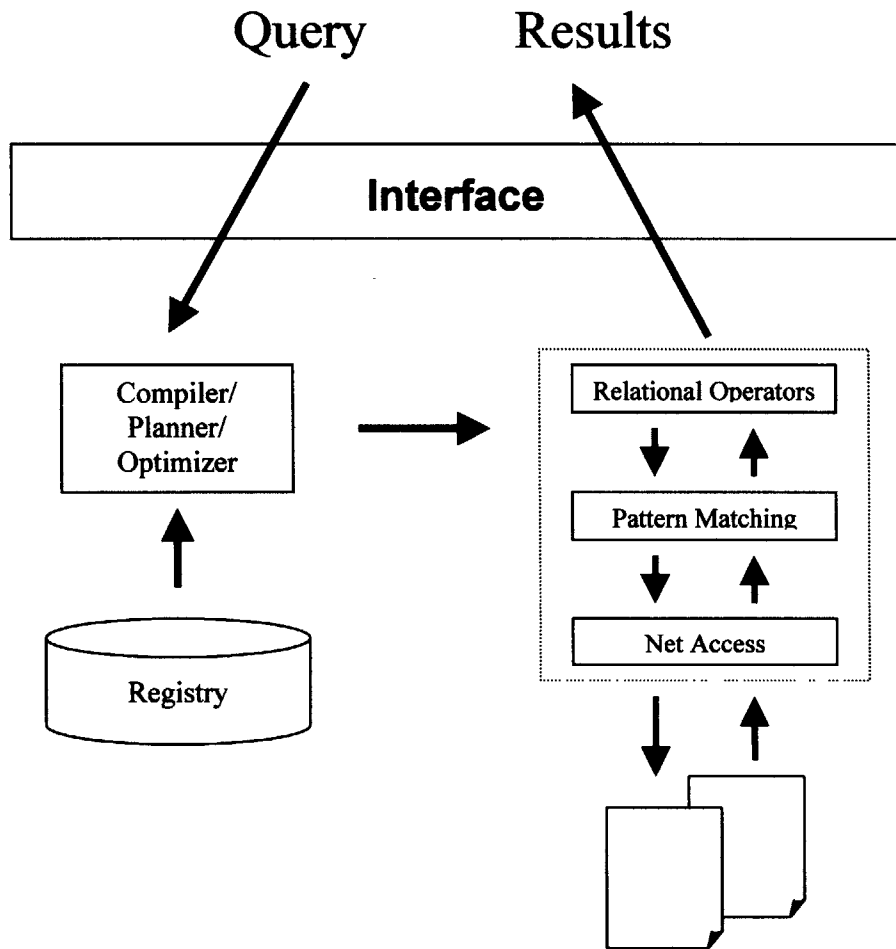


Figure 3.1: Wrapper Architecture

The following steps provide a brief overview into the dynamics of the wrapper engine. These broad steps are taken by the engine to wrap information every time the input is entered and the user requests an execution:

1. The SQL query from the user is parsed into a tree.
2. The sources of the query, taken from the input SQL, determine which specification files and Perl regular expressions to use. All the relevant specification files are parsed into trees.
3. According to the parse SQL tree, a plan is formulated to fetch information and to conduct relational operations.
4. Data are fetched concurrently from multiple web sites, with the results returned in a table format.
5. Relational operations are performed on the returned data, represented as tables; the final output is displayed to the user.

In the following chapters, the current design and implementation as well as the mechanisms behind the query execution engine and the planner/optimizer will be described in detail.

Chapter 4

Query Execution

The main focus of this thesis centers on the creation of a framework that will allow the effective and efficient execution of a SQL query, which accesses multiple data sources via the Internet. The engine will be designed to support read-only queries.

4.1 Relational Algebra

Relational algebra is a formal query language associated with relational database model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts relation instances as arguments and returns a relation instance as the result. This property makes it easy to compose operators to form a complex query. There are five basic operators of the algebra - select, project, union, cross-product and difference. All other operators can be defined in terms of the basic operators [12].

4.1.1 Select and Project

Relational algebra provides operators to select rows from a relation and to project columns. These operations allow us to manipulate data in a single relation. The select operator specifies the tuples to retain through a selection condition. In general, the selection condition is a boolean combination of terms that have the form *attribute op constant* or *attribute1 op attribute2*, where *op* is one of the comparison operators. The schema of the result of a selection is the schema of the input relation instance. The project operator allows us to extract columns from a relation. The schema of the result of a projection is determined by the fields that are projected.

4.1.3 Set Operations: Union, Intersection, Set-Difference, Cross-Product

The following standard operations on sets are also available in relational algebra: union, intersection, set-difference, and cross-product

- Union: $R \cup S$ returns a relation instance containing all tuples that occurs in either relation instance R or relation instance S (or both). R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.

- Intersection: $R \cap S$ returns a relation instance containing all tuples that occur in both R and S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R. Intersection is in fact redundant; $R \cap S$ can be defined as $R - (R - S)$.
- Set-difference: $R - S$ returns a relation instance containing all tuples that occurs in R but not in S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.
- Cross-product: $R \times S$ returns a relation instance whose schema contains all the fields of R followed by all the fields of S. The result of $R \times S$ contains one tuple $\langle r, s \rangle$ for each pair of tuples $r \in R, s \in S$.

4.1.4 Other Operators

All other operators can be defined as an expression using the five basic operators. For example, join can be defined as a cross-product followed by selections and projections. The five basic operators serve as the building block for all relational algebra expressions.

4.1.5 Relational Completeness

If a query language can express all the queries that can be expressed in relational algebra, it is said to be relationally complete [12]. A practical query language is expected to be relationally complete. In addition, commercial query languages (including SQL) typically support features that allow users to express some queries that cannot be expressed in relational algebra. In other words, even if our query execution engine is relationally complete, it still does not ensure full compatibility with SQL or any other commercial query languages.

4.2 Execution Tree

Our query execution engine has a set of instructions to operate on. These instructions include relational operators such as select, project, join, and union, and physical operators such as submit, regular-expression and scan. Execution trees are made up of a tree of operators with their required parameters. The specific parameters would differ depending on which operator is being used. The tree is compiled into a directed graph where a single node in the graph represents each operator [1]. Figure 4.1 shows the execution tree for query 1, examples of relational and physical operators that may be used. The connections within the graph represent data streams, which allow data to flow in one direction. A single data table is produced by each relational or physical operation.

Query 1

```
SELECT    Ticker, LastTrade, High, Low, Headline
FROM      usatoday, fastquote
WHERE     Ticker in ["AAPL", "IBM"]
```

Execution Tree:

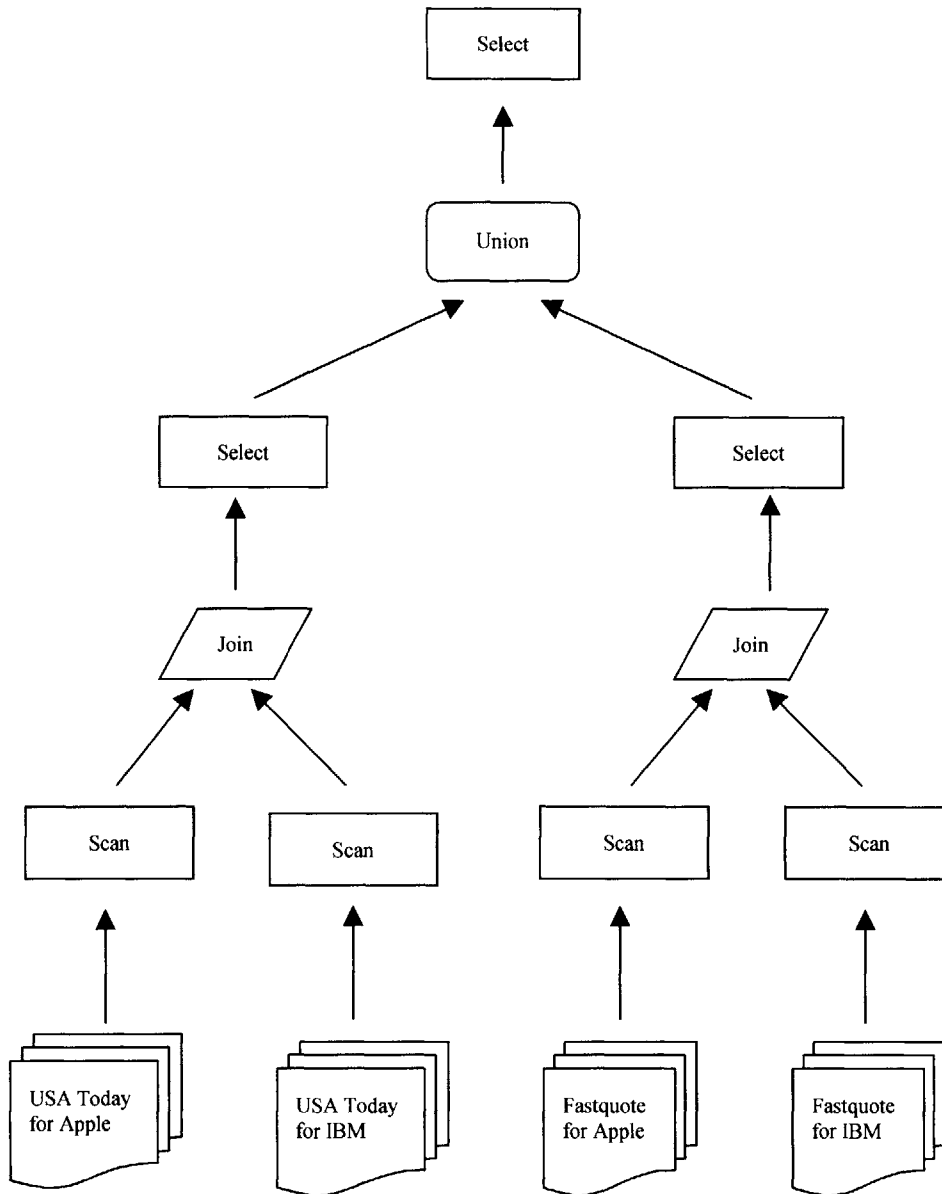


Figure 4.1: Execution Tree for Query 1

In this query, USA Today pages, which provide the last trading price, the day's high and the day's low trading prices, are accessed concurrently for Apple and IBM. Simultaneously, Fastquote pages, which provide the headlines, are accessed concurrently for the same two companies. The appropriate join, select and union operations are applied in the correct order, as shown in the execution tree above, to tabulate the results to the receiver.

4.3 Relational Operators

Relational operators are the set of standard operators on collections of structured data [1]. Select, project, join and union provide the basics of relational operations. Other complex operations can be built with these four elementary operators, to allow for the evaluation of arithmetic expressions and the evaluation of boolean comparisons. The implementation of the system provides for the extension of the base operators.

4.3.1 Select

The class constructor for this operator is of the following form:

Select(SelectionList, Conditions, Subtree)

SelectionList: This is the list of attributes to be selected from the relation. These attributes will be involved in the join conditions of the query and those in the final projection list of the query.

Conditions: A set of boolean operations applied to the attributes of the relation. Each attribute is replaced by an index into the relation.

Subtree: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

This select node is used to apply conditions to intermediate results. Only the set of tuples that pass the condition will be propagated up the tree.

4.3.2 Project

The class constructor for this operator is of the following form:

Project(ProjectionList, Subtree)

ProjectionList: This is very similar to the selection list of select; it is a list of attributes to be projected from the relation.

Subtree: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

Project is used to extract attributes from a given *Subtree*. The results will be propagated up the tree.

4.3.3 Join

The class constructor for this operator is of the following form:

Join(SelectionList, Conditions, Subtree1, Subtree2)

SelectionList: This is very similar to that of select, but the attributes are indices from both the incoming relations.

Conditions: A set of join conditions on the relations. If no conditions are given then the result of the join will be a full cross product of the two relations.

Subtree1: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

Subtree2: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

For each pair of tuples in the two relations gotten from *Subtree1* and *Subtree2*, the operator will check to determine whether the join condition will hold. If it does hold, then concatenating the two tuples forms a new tuple, which is then put onto the result stream for the join operator.

4.3.4 Union

The class constructor for this operator is of the following form:

Union(Subtree1, Subtree2)

Subtree1: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

Subtree2: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

This is used to get the union of the results obtained by executing the two subtrees.

4.4 Physical Operators

Physical operators are those that perform functions other than relational operations. These include submit, join-submit and regular-expression. Submit and join-submit are two access operators which send requests for data and stores the result of the request as sets of data tuples. Regular-expression is the pattern matching operator which extracts the relevant attributes from a stream of output. The resulting tuples are propagated up the tree.

4.4.1 Submit

The class constructor for this operator is of the following form:

Submit(Sourceaddress, Sourcetype)

Sourceaddress: A string representation of the absolute address from where the required data has to be extracted.

Sourcetype: The type of data source being accessed.

This is the first of the two access operators. Both access operators send requests for data, located at sourceaddress, to the system scheduler. The response from the scheduler is a handle to a data stream, which holds the result of the request - from this point on, all data

within the system needs to be transferred as sets of tuples passing through the buffered data streams between operators.

4.4.2 Join-Submit

The class constructor for this operator is of the following form:

Join-Submit(Sourceaddress, Sourcetype, Subtree)

Sourceaddress: A parameterized address that has to be completed by one or more attribute values.

Sourcetype: This is same as in the submit operator.

Subtree: This is the data structure for the subtree, which gives the set of tuples that are used in this operator.

This operator is used when the address of a data source depends on the value of a particular attribute, e.g. the stock information of IBM can be accessed from the following web address:

<http://qs.cnnfn.com/cgi-bin/stockquote?symbols=IBM>

The address includes the value of the Ticker attribute IBM and is only a partial address without this attribute. The attributes needed to construct the complete address are accessed from another relation that is gotten from the *Subtree*. For each tuple in the *Subtree* relation, a new address will be constructed and a request sent to the scheduler. Once the resulting stream returns it will be appended onto the appropriate tuple as a new attribute and the tuple is put on the buffered data stream.

4.4.3 Regular-Expression

The class constructor for this operator is of the following form:

Regular-expression(RegularExp, AttrList, Subtree)

RegularExp: A string parameter that holds the regular expression used in the pattern matching process to extract data.

AttrList: The AttrList gives a list of all the attributes that the system extracts from the input stream and the types of these attributes. The attributes are represented by the index into the regular expression.

Once a stream result from a source is passed on from one of the access operators, the regex operator will take this stream and extract the attributes stored in the stream. The input stream is taken from the last attribute of the incoming result from the *Subtree* execution. Using the pattern matching techniques with the *RegularExp*, this operator produces a list of attributes, which are all appended to the tuple from which the attributes used to complete the address for the request were taken.

Chapter 5

Planner

The planner represents the missing connection between the user supplied SQL query and the execution tree. This chapter outlines the design and implementation of the planner to facilitate the wrapper engine outlined in the preceding chapter.

5.1 Specification File

Before a source can be used by the system, information such as the type of source, the source address, means of data extraction, list of provided attributes and their types must put into a specification file.

Type of source

The type of source can vary from web pages to local files to relationship database. Each type of source requires a particular set of access procedures hence the necessity of having the source type as one of the parameters stored in the source specification file.

Source address

Source address varies as well according to the source type. The address is vital for it provides the location of the source.

Means of data extraction

In addition to source type, in the case of web pages the method of access is also necessary piece of information as web sites use a combination of POST and GET access methods in the HTTP protocol.

List of provided attributes and their types

The list of provided attributes and their types allow the planner knowledge of what kinds information to look for within the source. This list is matched up with the information requested in the user SQL query and, if matched, will retrieve the requested information from the source and deliver back to the user.

The specification files have a tagged-based syntax to make it simple to identify the various parts of the specification description both for the system administrators who will

be writing the files and to simplify the specification file compiler. The tagged components are neatly cased into the header and the body of the specification.

5.1.1 Header

The header contains the general information for each source. Most of the information is not actually used in the planning or the execution but is there for identification and registry purposes. The <HEADER> tags clearly identify the region of the file that represents the specification file header. The following is the header part of the specification file for Yahoo site.

```
<HEADER>
  <RELATION>yahoo</RELATION>
  <HREF>GET http://quote.yahoo.com</HREF>
  <SCHEMA> Ticker:string, Time:string,
  LastTrade:real, Changepts:real,
  Changepct:real, Volume:integer
</SCHEMA>
</HEADER>
```

Relation

The name of the relation is also the name of the system file in which the information is stored. The string between the <RELATION> tags always identifies the relation.

HREF

Another parameter is the *href* that holds the base address of the main source stored within the file. The string between the <HREF> tags identifies the *href* parameter.

Schema

The last and probably the most important parameter in the header is the schema. All the attributes that the source provides or requires are stored within the schema. Types are necessary to facilitate comparisons and certain types of operations that may be carried out by the execution engine. The string between the <SCHEMA> tags identifies the schema.

5.1.2 Body

The second section of the specification file is called the body and is marked by the <BODY> tags. This section is made up of all the information that is necessary to allow the planner to make an execution tree. The body is made up of a set of pages. Each page has a method, a URL and a pattern. Within the body section all attributes are referenced with the attribute name surrounded by two pairs of ## characters. The following is the body part of the specification file for Yahoo site.

```
<BODY>
<PAGE>
  <METHOD> GET </METHOD>
```

```

<URL>http://quote.yahoo.com/q?s=##Ticke
r##&d=v1&o=t</URL>
<PATTERN><a
href="/q?s=.*?&d=t">.*?</a>\s+##Time:(
.*?)##\s+<b>##LastTrade:(.*?)##</b>\s+##
#Changepts:(.*?)##\s{2,}##Changepct:(.*
?)##\s+##Volume:(.*?)## <small>
</PATTERN>
</PAGE>
</BODY>

```

Method

The method gives the method of remote access (POST or GET) according to the HTTP protocol. In the case of pages with POST methods, there may also be a content parameter. The string between the <METHOD> tags identifies the method.

URL

The URL gives the access location of the remote data source. This represents the exact address of web site that is accessed to obtain the given attributes. The string between the <URL> tags identifies the URL.

Pattern

The pattern is a regular expression that is used to extract the attributes from the remote data source. Each attribute that needs to be extracted is named, along with the regular expression with which it must be matched. The name of each attribute and its corresponding regular expression is surrounded by two pairs of ## characters for identification purpose. This regular expression is applied to page source. The string between the <PATTERN> tags identifies the pattern.

5.2 Ordering Sources

The capability and requirements of the sources sometimes implicitly determine the order in which the selected source can be accessed. Other times, ordering is necessary and an algorithm is applied during execution to determine the ordering and return the ordered list of sources, if such ordering can be found. The main idea is to treat all pages as individual sources with a list of attributes it requires and a list of attributes provided. Starting with the set of required attributes from the query, the system would construct the list of sources by checking for those that supplied the required attributes. Query conditions normally adds to the initial list of required attributes, but in the case of an assignment constraint, the condition statement then becomes a source. This source will be represented as a constant data source in the execution engine, with tuples being made up from the values used in the assignment. The ordering algorithm uses a beam search, depth first search with the major difference being the fact that at any one point in time there are n active nodes to be extended.

5.3 Planning

Once an appropriate list of data sources is produced, the plan generation module takes the initial set of required attributes, the query conditions and the resulting list of sources and uses them to construct the final execution tree. This is done by traversing the list of sources and creating the appropriate set of relational operators needed to extract the attributes required from that source and to use those attributes as they are needed within the entire query execution.

Query 1

```
SELECT    Ticker, LastTrade, High, Low,  Headline
FROM      usatoday, fastquote
WHERE     Ticker in ["AAPL", "IBM"]
```

Plan constructed:

```
select( true,true,true,true,false,true, , TRUE)
regex_fn( com.oroinc.text.regex.Perl5Pattern@2269c4 )
join-submit (
[http://fast.quote.com/fq/quotecom/headlines?mode=headlines
&symbols=,0, &mode=NewsHeadlines] )
select( true,true,false,false,true,true,false,false, ,
TRUE)
regex_fn( com.oroinc.text.regex.Perl5Pattern@226af0 )
join-submit (
[http://quote.bloomberg.com/usatoday/bquote.cgi?ticker=, 0,
] )
constanttable ( [AAPL, IBM] )
```

High Level Explanation:

The constanttable operator creates a data structure to hold the basic elements of the search. In our example, it creates the structure to hold the Ticker AAPL and IBM, as they form the basis of the search. Then the join-submit operator performs the actual extraction of the attributes, using the exact web address (USA Today) given and the regular expression provided by the regex_fn operator. The results from this concurrent execution are joined together in an appropriate fashion. The select operator selects the necessary attributes according to the request of the original query. Simultaneously, another join-submit and regex_fn accesses the Fastquote pages for headlines regarding AAPL and IBM. The final select operator will select the necessary attributes and formulate the result to the receiver.

Query 2

```
SELECT    Ticker, Time, LastTrade
FROM      yahoo
WHERE     Ticker in ["MSFT", "YHOO", "INTC", "IBM"]
AND      LastTrade < 100
```

Plan constructed:

```
select( true,true,true,false,false,false, , int 2 ::
INT_ATT:0/0 < 100)
regex_fn( com.oroinc.text.regex.Perl5Pattern@223c3c )
join-submit ( [http://quote.yahoo.com/q?s=, 0, &d=v1&o=t] )
constanttable ( [MSFT, YHOO, INTC, IBM] )
```

High Level Explanation:

The constanttable operator creates a data structure to hold the Ticker MSFT, YHOO, INTC and IBM, as they form the basis of the search. Then the join-submit operator performs the actual extraction of the attributes, using the exact web address given (Yahoo in this case) and the regular expression provided by the regex_fn operator. The results from this concurrent execution are joined together in an appropriate fashion. The select operator selects the necessary attributes according to the request of the original query. This select operator also performs the boolean operations to test whether the last trading price of each of the four companies lies below \$100. The companies with last trading prices below \$100 are returned to the receiver along with all the other information requested; the ones with last prices above \$100 are taken out.

Chapter 6

Implementation Details

6.1 Implementation Language

The entire system implementation was done in Java, with the objective being to develop a wrapper engine that could be integrated into the COIN system and several other architectures including client software on personal computers. There are numerous reasons to choose Java over other languages - it is platform independent, it supports interactive content on web pages, it is well suited to distributed networking applications because of its built-in networking support, and it directly supports the object-oriented concepts of encapsulation, inheritance, messages and methods, and data hiding. Most important of all, however, is the availability of threads; this is the main reason for choosing Java as the language of implementation. Threads allow for concurrent data accesses, which are the primary means of reducing bottlenecks that may develop when trying to import data not stored on the local disk. On a sequential execution, the latency of the application is the combined time of all remote accesses; in comparison, on a concurrent execution, the latency can be reduced to just the time of the longest network access. Programming in threads to facilitate concurrency could be done with another language but the processes that these languages launch will cause too much of a drain on the processor. Semantec Visual Cafe Version 3 is the chosen Java compiler because it offered the ability to compile Java code into Native Win32 DLLs and into Win32 executable.

6.2 Modules

The implementation of the wrapper engine includes a number of different modules for the convenient reuse of often required functionality. The purpose of this chapter is to introduce these modules. Within the Java implementation, these modules are written as different packages; in order to use any of them, a programmer must import the package and refer to the exported variables of the package. Figure 6.1 provides a brief overview of the interaction between the different modules.

Module Dependency

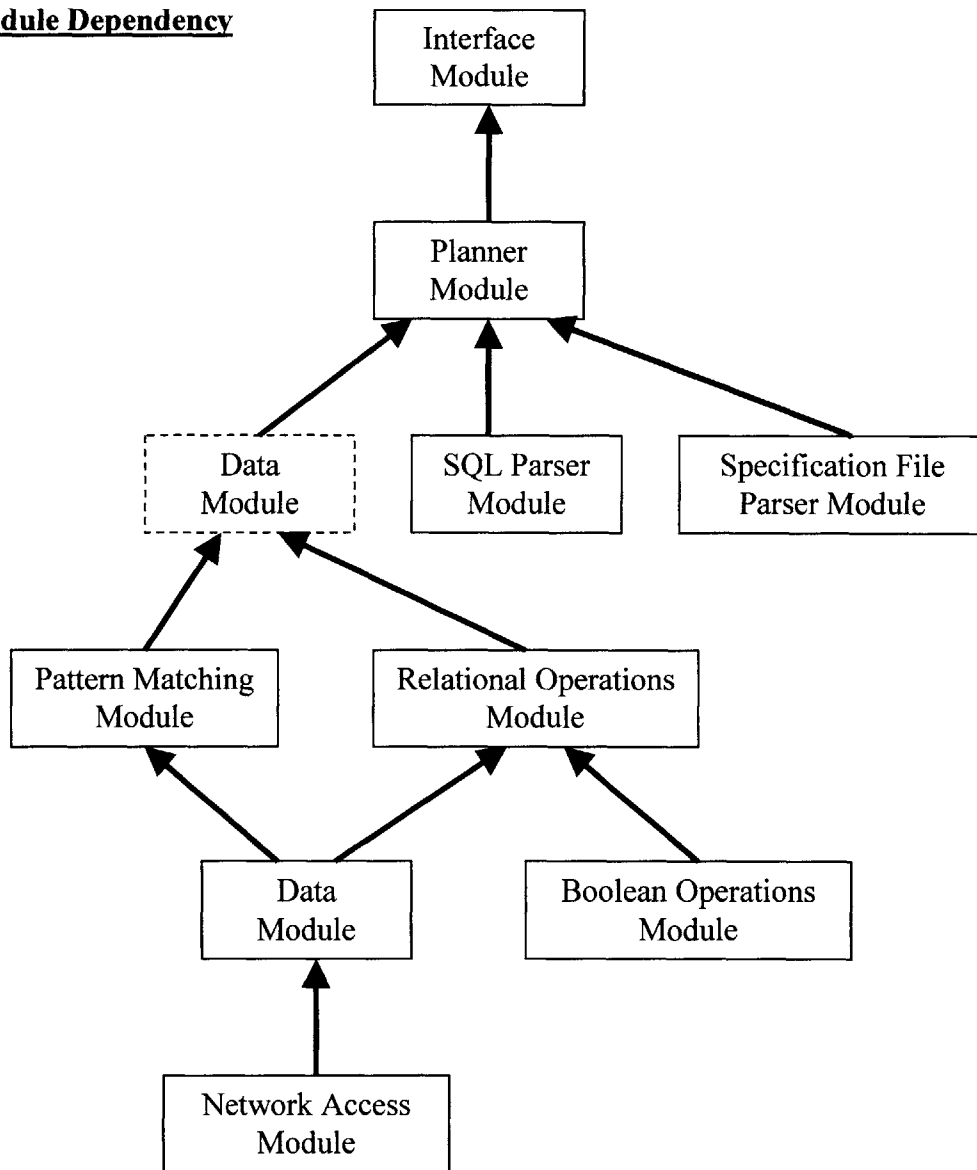


Figure 6.1: Module Dependency

6.2.1 Network Access Module

All remote network accesses are done via the Hypertext Transfer Protocol (HTTP) or the File Transfer Protocol (FTP). This module is implemented as a package called `accessdata` using the base `java.net` package. The package contains three classes, together they function to retrieve data from remote network sources and return the results as an output stream buffer. Figure 6.2 shows the inheritance tree of the `accessdata` package.

Accessdata Classes Inheritance Tree

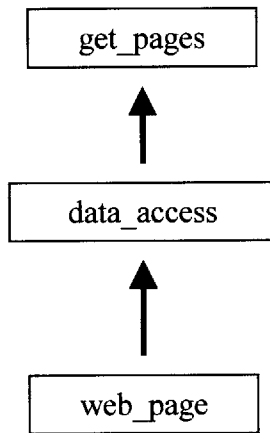


Figure 6.2: Accessdata Classes Inheritance Tree

6.2.2 Boolean Operations Module

The boolean operations module is implemented as a package called `bool_op`. The package facilitates the representation of specific boolean elements as well as the use and the evaluation of various conditional statements. The base operators (`true`, `false`) always return their named values (`true`, `false`). Another set of operators (`and`, `or`, `not`) use the results from other operators to determine their value. Hence, a condition can be described as a tree of boolean operator. This structure lent itself to a few optimizations in the experimental concurrent implementation, as not all sub-trees had to be evaluated. The following figure shows the inheritance relationship between the different classes in the `bool_op` package.

<code>bool_op</code>	==	<code>const_bool_op static_bool_op norm_bool_op</code>
<code>const_bool_op</code>	==	<code>true false</code>
<code>norm_bool_op</code>	==	<code>not bool_op and bool_op bool_op or bool_op bool_op</code>
<code>static_bool_eval</code>	==	<code>static_op data data</code>
<code>static_op</code>	==	<code>> => <= <</code>

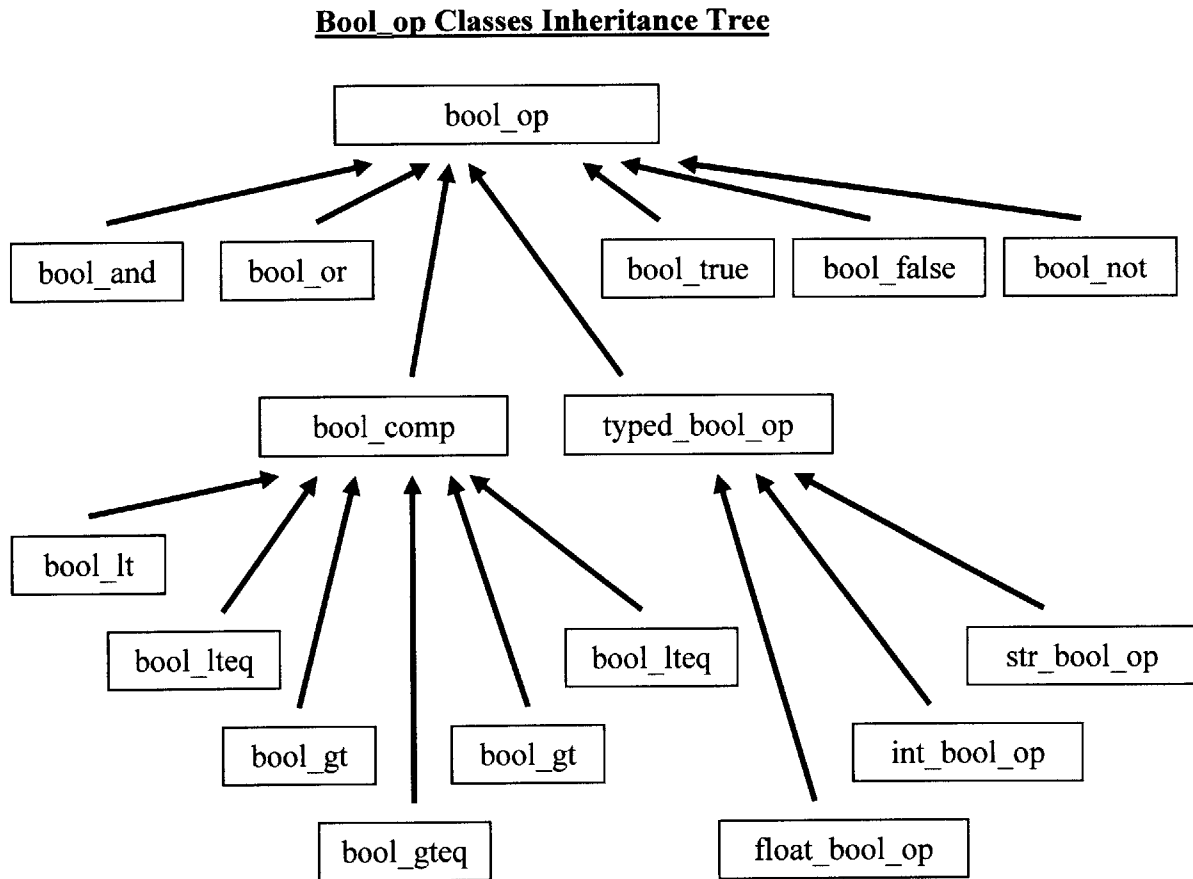


Figure 6.3: Bool_op Classes Inheritance Tree

6.2.3 Data Module

This module is implemented as a package called data. It is used as a container data structure to make manipulation of single and multiple pieces of data more manageable. The main classes within this module are the datatable and the tuple classes. A tuple is a container for attributes and a datatable a container for tuples. Figure 6.4 shows the data classes inheritance tree.

Data Classes Inheritance Tree

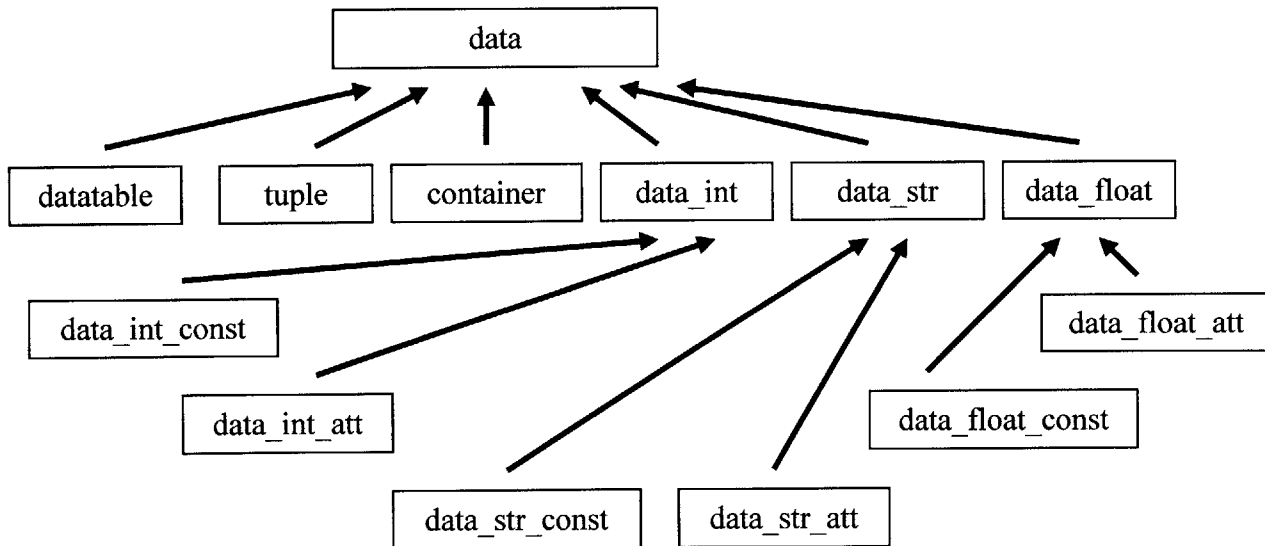


Figure 6.4: Data Classes Inheritance Tree

6.2.4 Pattern Matching Module

The pattern matching module uses a commercial regular pattern matcher in Java to conduct data extraction. The package downloaded and used is the OROMatcher version 1.1. The OROMatcher package is a set of regular expression pattern matching and utility classes for Java descended from a package originally written by Daniel Saverese. It is geared toward programmers who are already familiar with regular expressions, having used them with other languages, and who now want to apply them in their Java programs. In brief, a regular expression is a pattern denoted by a sequence of symbols representing a state-machine or mini-program that is capable of matching particular sequences of characters. Regular expression provides a very powerful technique for matching and extracting data from semi-structured web pages.

6.2.5 Relational Operations Module

Relational operations module is implemented as a package called relational_op. Main classes include project, union, join, select, regex, scan, submit, multiscan and constanttable. Figure 6.5 provides the class inheritance tree of the classes within relational_op.

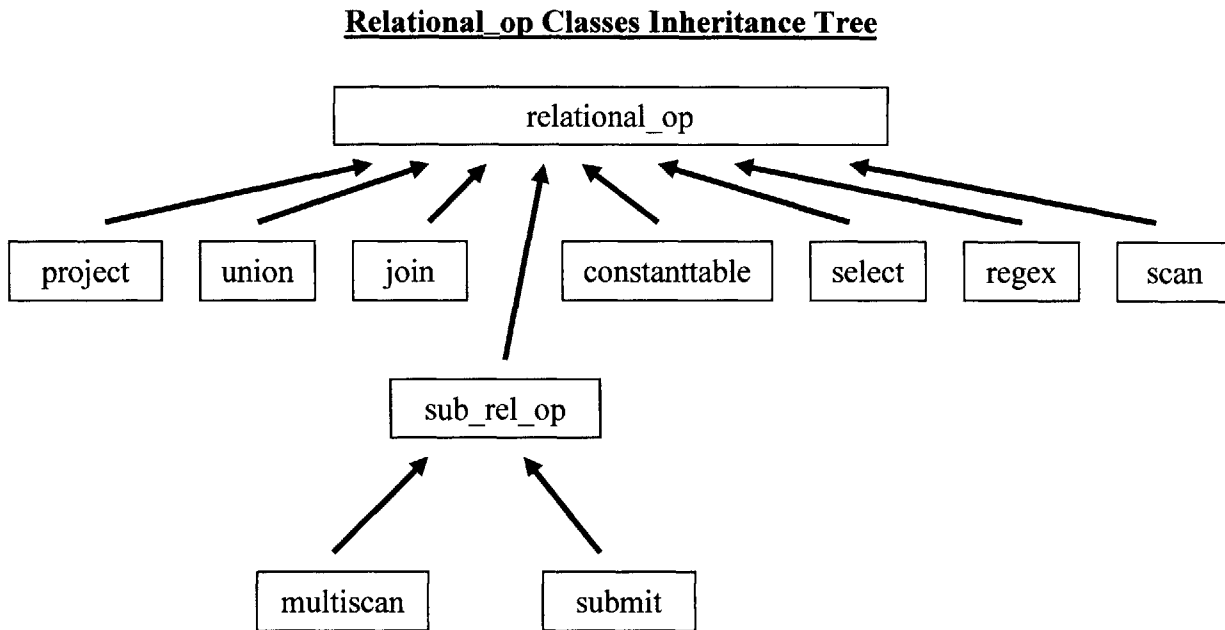


Figure 6.5: Relational_op Classes Inheritance Tree

6.2.6 SQL Parser Module

The SQL parser is generated by the Java Compiler Compiler (JavaCC) automatically with a limited grammar as input. The resulting parser is not an exact SQL parser because of the limited grammar. The advantage is that it provides more flexibility in terms of the amount and the variety of queries that can be processed and parsed. The disadvantage of the limited grammar is that the resulting parser is not mutually compatible with standard SQL queries. A complete SQL parser can be implemented by feeding into the parser generator the full set of grammar describing the semantics of SQL. JJTree, the tree building preprocessor which is used with JavaCC, enables the generation of a compilation tree that hold the components of the SQL that are needed. The compilation tree is converted into an internal SQL data structure, which is used throughout the planning.

6.2.7 Specification File Parser Module

The specification file parser is written from scratch because of the simplicity of the design of the specification files. This compiler uses the same regular expression technology as in data extraction. As with the SQL compilation, there is a SPEC data structure that holds all the relevant parameters as needed.

6.2.8 Planner Module

The planner module is implemented as a collection of classes. They include planner, SPEC, SQL, constant, path, condition, conditionSet, attribute, attributeSet, and several

others. The main classes are the planner which determines the plan and also call the execution engine to carry out the query execution, the SQL which holds the data structure for the parsed SQL query, and the SPEC which holds the data structure for the parsed specification files.

6.2.9 Interface Module

The interface module is implemented to facilitate testing and usage of the execution engine and the planner. It is implemented as a simple applet; it allows the user to input the appropriate query and it returns the result on another frame within the same applet. The interface is primitive and it does not have any exception handling mechanisms to catch input errors and other illegal query accesses. However, it suffices for our stated purpose to test and to facilitate the initial usage of the wrapper engine. The eventual objective of our research effort is to create a more intelligent and user-friendly interface or to incorporate our engine into the existing COIN architecture to allow access to the wrapper engine via existing interfaces. Figure 6.6 below provides a shot of the interface as implemented.

Interface of the Wrapper Engine

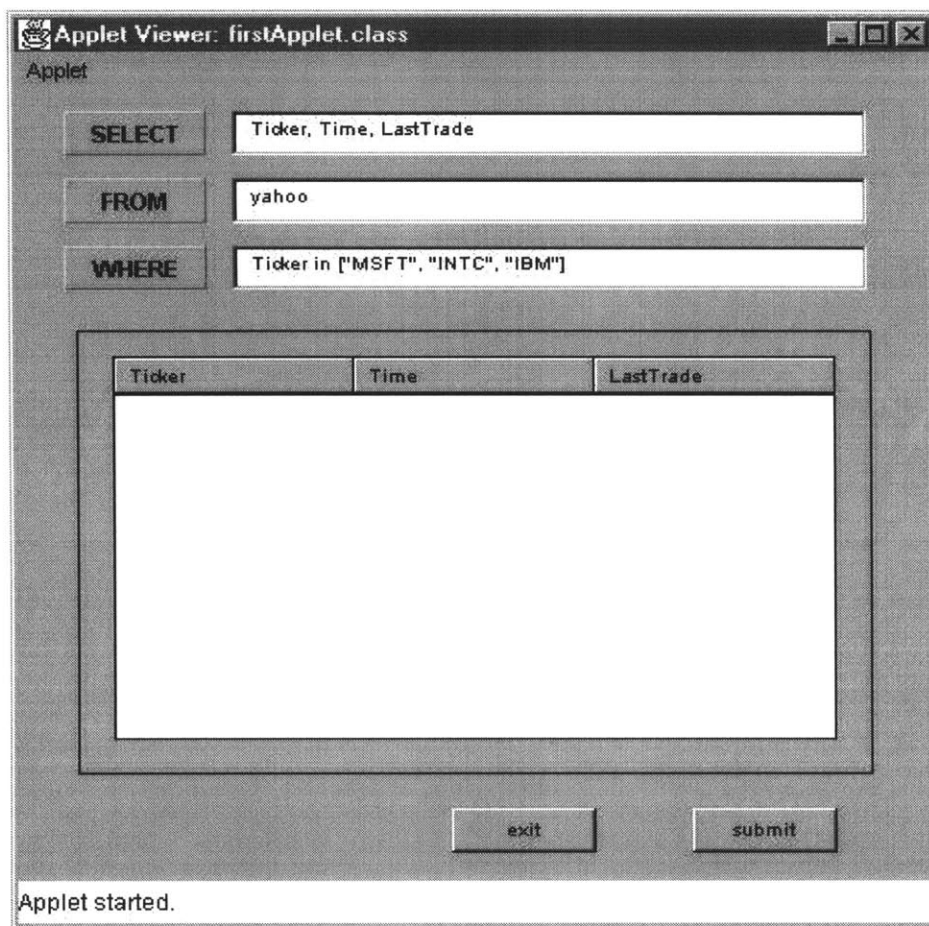


Figure 6.6: Interface of the Wrapper Engine

6.3 Scheduling Threads

Limited machine resources require that a monitor be used to ensure that the number of active data access threads does not threaten to use all the available resources. This necessitated the implementation of a scheduler that controlled the total number of access threads that could be opened simultaneously. Access operators can only make request for data through the scheduler. Once this request is made, the scheduler will assign a new accesstthread to the operator's request. The thread will only be started if the number of active requests is below some pre-determined number. In addition to being a system that controls the use of machine resources, this functionality can actually be used as a tuning device for the system. Given that the trade-off is between network parallelism and thread processing overhead, changing the maximum number of active threads can affect both of these factors and then observations can be used to determine what sort of dependency the total execution time has on the two factor, and hence find the best setting. In the current implementation, the maximum number of active data access threads is 25. Preliminary experiments conducted have shown the thread maintenance scheduling overhead will become excessive once the number of active threads goes above 25. The accesstthread will initialize the data access for the request and deliver the resulting stream to the operator that originally submitted the request to the scheduler. Figure 6.7 shows the scheduler's operation. All the data that is read is returned to the access operators as an output stream.

Scheduler

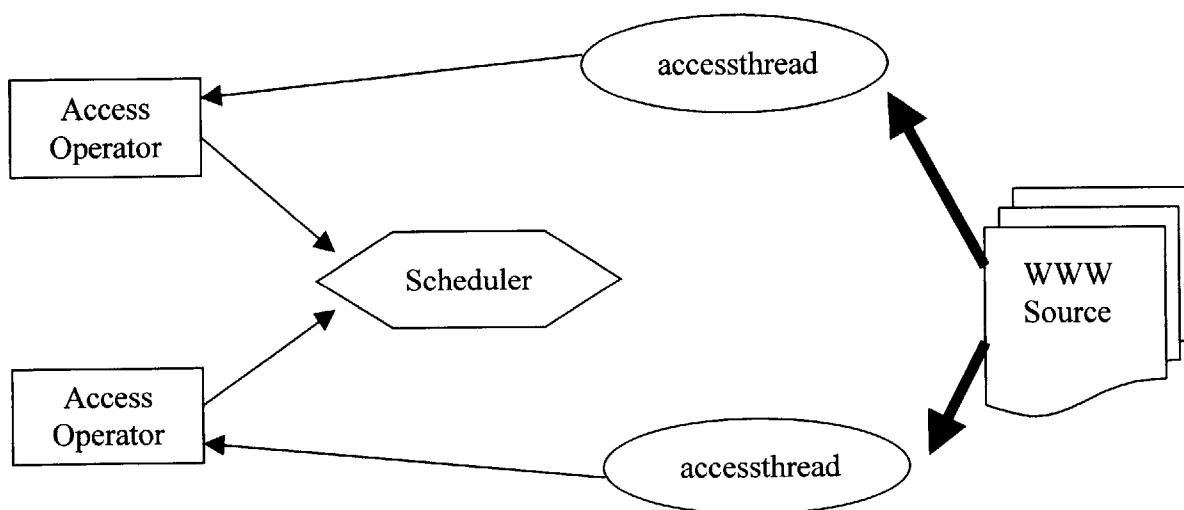


Figure 6.7: Scheduler

6.4 Concurrent Execution

The system is operated on a single processor machine, but the model used to implement the concurrency was geared towards leveraging the ability for the network parallelism and hence reducing the average network access time. Although the average access time may be reduced, there is a definite overhead in the scheduling of various threads. Hence, the sequential version will require less local processing. However, the concurrent model is fairer in that it produces early results as soon as possible and is able, in this task, to avoid bottlenecks created by data sources with low data rates (e.g. "slow" web sites). As most of the time is spent on the network retrieving data, the higher network parallelism (as per requests) should result in an improved performance (although this is bound by the network bandwidth and the servers' ability to handle multiple requests).

Figure 6.8 shows the concurrency model implemented in the system. The concurrency goes both vertically and horizontally. Each operator is a thread that will start to execute once the engine begins executing the plan. This allows interleaving of lower level instructions, hence increased parallelism. The fact that all the operators can execute concurrently maximizes the number of active access nodes, where the higher network parallelism is manifested.

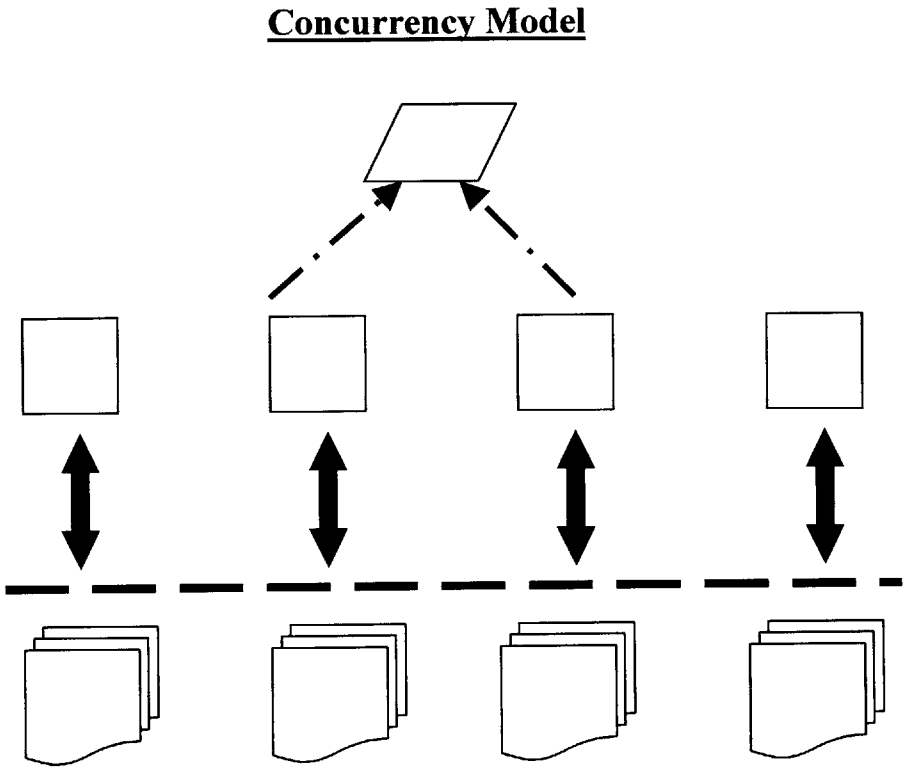


Figure 6.8: Concurrency Model

6.5 Dynamic Optimization

Whether the engine is to be used for ad hoc queries or queries which can be compiled, planned and optimized in advance, the availability of dynamic optimization mechanisms is critical for the system to be able to react to unpredictable and uncontrolled behaviors and performances of the network and sources. Network access is prone to slow or non-responding servers and networks being randomly congested. One simple way to reduce the system susceptibility to these random events is to eliminate unnecessary access.

Dynamic optimization can help to reduce unnecessary access to data sources. Consider the following query:

```
SELECT    Ticker, LastTrade, Headline
FROM      usatoday, fastquote
WHERE     Ticker in ["AAPL", "IBM"] AND LastTrade>100
```

Ticker and Headline are attributes for the Fastquote web site, while Ticker, LastTrade are attributes for the USA Today site. There are two ways to return the correct result of this query. The first involves concurrently accessing USA Today pages to find the last trading price of Apple and IBM, while simultaneously accessing Fastquote pages to find all recent headlines involving Apple and IBM. The results are then tabulated and the relational operations are evaluated. Because the stock price of Apple is trading below 100 and those of IBM are trading above, all the information about Apple are taken out, returning only the last trading price and the headlines of IBM to the receiver. The second method involves getting the last trading price of Apple and IBM. Relational operations are performed next to take out Apple. Only then are sites from Fastquote accessed to get the recent headlines of IBM. Obviously, the second method, one involving dynamic optimization, is much better than the first because it eliminates one access to Fastquote for Apple's headlines. The implementation uses the second method. The heuristic for choosing the second method is to choose to access sites involving relational operations first and suspend access to all other sites. Relational operations can then be performed to screen out unnecessary accesses before further network accesses are resumed. Although some advantages from concurrent execution are sacrificed to implement this feature, experiences with remote access reveal that the bottleneck often lies with network delays while the loss in sequential execution against concurrent execution is minor in comparison.

Consider the following query:

```
SELECT    Ticker, LastTrade, Headline
FROM      yahoo, bloomberg
WHERE     Ticker in ["AAPL", "IBM", ..., "INTC"]
```

Ticker and Headline are attributes for the Bloomberg web site, while Ticker and LastTrade are attributes for the Yahoo site. Assume for the example that there are 25 stocks in the portfolio and the number of active threads allowed is 25. There are two ways to return the result to the query. This first is to concurrently access yahoo sites and to get the last trading price of the 25 stocks. Then issue 25 requests to Bloomberg for headlines. The second method does the reverse - Bloomberg is accessed first for headlines. However, because each company has multiple headlines, the resulting data

table for this intermediate result has many more than 25 entries. Thus, this system will need to issue many more than 25 requests for last trading prices of the stocks. Again, it is obvious which method is better; with dynamic optimization, the first method is chosen and unnecessary network accesses can be eliminated and avoided. This example is much more difficult to implement than the first. The application cannot know in advance which of the various network accesses will generate unnecessary accesses in the future. Instead of issuing 25 threads to Yahoo sites or 25 threads to Bloomberg sites, the current implementation issues half to them to Yahoo and half to Bloomberg to determine which of the two methods is optimal. The remaining execution follows that of optimal method described.

There are also many cases involving sources that are non-responding, taking into consideration that useful work towards answering the original query can still be done with the remaining sites. In future implementation, a partial answer could be returned using a form of query scrambling.

Chapter 7

Conclusion and Future Research

7.1 System Evaluation

The wrapper engine implemented, running as a server process, has been tested with hundreds of queries and it has proven to be useful and reliable. The first run of the engine is typically very slow, often requiring more than one to two minutes of execution time. This can be explained because the local processor needs this overhead processing time to load all the Java classes into its memory and operating systems. Further runs of an opened application are very fast; even complicated and work-intensive queries are usually executed within one minute of query submission.

The system can be easily incorporated into a larger framework, such as COIN. The query execution, the planner and the optimizer are all necessary components; together they will serve as a back-end web wrapper engine and they are compatible with the various front-end applications available within COIN.

7.2 Future Research and Concluding Thoughts

There are numerous improvements to be made within the existing design and implementation of the wrapper engine:

- An intelligent mechanism to facilitate the process of maintaining and generating specification files. This will require some additional research and design; in particular, it can be very difficult for users unfamiliar with the system to create specification files even with the help of an intelligent tool and a friendly interface.
- Current implementation uses a pseudo-SQL parser generated by a parser-generator. A pseudo-SQL grammar is fed into a JAVA language parser and it generates a parser according to those rules. Future implementation can provide a more complete SQL parser.
- Extension of the engine to accept non-HTML sources, for example XML sites. This is an important extension because other sources, such as XML, can provide a better information structure to facilitate mediated querying in a remote, distributed fashion.

- The current planner is not implemented to find the most optimal solution. The algorithm used is depth-first search but without expanding all the leaves at any given node of the planning tree. Hence, the planner finds a reasonably good solution but it is not guaranteed to be the optimal one. A novel approach must be employed to find the optimal path because the time required to correctly depth-first search the query execution tree is simply too long and unjustified.
- The current implementation does not allow for nested SQL query. The issue with this problem begins with our SQL parser that cannot parse nested query in the first place. In addition, changes in the implementation are required to return the results of a query as input of another even in the presence of a complete SQL parser.
- Future implementation should also allow for access to password protected sites as well as data sites that require cookies.
- Network delays are major issues in spite of concurrent execution. Often useful work is accomplished but the query cannot be completed because part of it involves accessing information from busy or shut-downed sites. Future design and implementation should look into this area of concern.
- The planner and executioner can be designed to aggregate all the queries to relations in the same data source, so that each source would only need to be assessed once.

Some of these further works can be achieved in a matter of a few days; others involve a coordinated effort among many different components of the system. This list is by no means exhaustive as possibilities of future research are unlimited. Yet, one should not be discouraged by the current design and implementation limitations. From a simple, sequential wrapper engine to the current complex and concurrent implementation, a lot of grounds have been covered in web wrapper technology development in a very short span of time.

References

- [1] **Ambrose, R.** A Lightweight Multi-Database Execution Engine. *MIT Sloan School of Management CISL Working Paper* (1998).
- [2] **Bressan, S. and Bonnet, P.** Extraction and Integration of Data from Semi-structured Documents into Business Applications. *Conference on Industrial Applications of Prolog* (1997).
- [3] **Bressan, S., Fynn, K., Goh C., Madnick, S., Pena, T., and Siegel, M.** Overview of a Prolog Implementation of the Context Interchange Mediator. *Proceedings of the Fifth International Conference and Exhibition on the Practical Applications of Prolog* (1997).
- [4] **Bressan, S., Goh, C., Fynn, K., Jakobisiak, M., Hussein, K., Lee, T., Madnick, S., Pena, T., Qu, J., Shum, A., and Siegel, M.** Context Interchange Mediator Prototype. *ACM SIGMOD International Conference on Management of Data* (1997).
- [5] **Fynn, Kofi.** A Planner/Optimizer/Executioner for Context Mediated Queries. *MIT Sloan School of Management CISL Working Paper #97-07* (1997).
- [6] **Goh, C.** Representing and Reasoning about Semantic Conflicts in Heterogenous Information Systems. *MIT Sloan School of Management CISL Working Paper #97-01* (1997).
- [7] **Gruser, J., Raschid, L., Vidal, M., Bright, L.** Wrapper Generation for Web Accessible Data Sources. *University of Maryland, In Proceedings CoopIS* (1998).
- [8] **Jakobisiak, Marta.** Programming the Web - Design and Implementation of a Multidatabase Browser. *MIT Sloan School of Management CISL Working Paper #96-04* (1996).
- [9] **Lee, Y.** Rainbow: Prototyping the DIOM Interoperable System. *University of Alberta* (1996).
- [10] **Liu, L. and Pu, C.** A Metadata Based Approach to Improving Query Responsiveness. *Proceedings of the 2nd IEEE Metadata Conference* (1997).
- [11] **Marais, H.** WebL – A Programming Language for the Web. *Compaq Computer Corporation Systems Research Center* (1998).
- [12] **Ramakrishnan, Raghu.** Database Management Systems. *University of Wisconsin* (1998).

- [13] **Shah, S.** Design and Architecture of the Context Interchange System. *MIT Sloan School of Management CISL Working Paper #98-05* (1998).
- [14] **Shum, A.** Open Database Connectivity Development of the Context Interchange System. *MIT Sloan School of Management CISL Working Paper #96-07* (1996).

Appendix A: User's Manual

Modules

- Network Access Module: Java source and class files of this module are kept in the *wrapper\accessdata* directory and the corresponding files are compiled into a *accessdata.lib* library object.
- Boolean Operations Module: Java source and class files of this module are kept in the *wrapper\boolean_operator* directory and the corresponding files are compiled into a *bool_op.lib* library object.
- Data Module: Java source and class files of this module are kept in the *wrapper\datatype* directory and the corresponding files are compiled into a *data.lib* library object.
- Pattern Matching Module: Java source and class files of this module are kept in the *wrapper\regexp\com\oroinc\text\regex* directory.
- Relational Operations Module: Java source and class files of this module are kept in the *wrapper\rel_op* directory.
- SQL Parser Module: Java source and class files of this module are kept in the *wrapper\planner\SQL* directory.
- Specification File Parser Module: Java source and class files of this module are kept in the *wrapper\planner\SPEC* directory.
- Planner Module: Java source and class files of this module are kept in the *wrapper\planner* directory.
- Interface Module: Java source and class files of this module are kept in the *wrapper\demol* directory.

Because of the level of dependency amongst the various modules, source and class files must be kept in their corresponding directory before successful attempts can be made to recompile the source code.

Specification Files

All the specification files must be kept in the *registry* directory and have the *spec* extension. Simply use the name of the file in the query statement will allow the engine to identify and use the specification file accordingly. In creating the file, you need to find an appropriate data source and, within it, the appropriate regular expression. Place the name of the attribute along with the regular expression required to extract it, separated by colon, between two pairs of ##'s. For example: ##Ticker(*?)##. The specification files are a major part of the wrapper engine and it ceases to function in the absence of appropriate and properly maintained specification files. A system administrator is required to regularly update and monitor the data sources and the regular expression as specified within the specification files for changes in the remote data sources.

Appendix B: Sample Specification Files

Yahoo Specification

```
<HEADER>
  <RELATION>yahoo</RELATION>
  <HREF>GET http://quote.yahoo.com</HREF>
  <SCHEMA> Ticker:string, Time:string,
  LastTrade:real, Changepts:real, Changepct:real,
  Volume:integer</SCHEMA>
</HEADER>
<BODY>
  <PAGE>
  <METHOD> GET </METHOD>
  <URL>
  http://quote.yahoo.com/q?s=##Ticker##&d=v1&o=t
  </URL>
  <PATTERN><a
  href="/q?s=.*?&d=t">.*?</a>\s+##Time:(.*)##\s+<
  b>##LastTrade:(.*)##</b>\s+##Changepts:(.*)##\s
  {2,}##Changepct:(.*)##\s+##Volume:(.*)##
  <small> </PATTERN>
  </PAGE>
</BODY>
```

Fastquote Specification

```
<HEADER>
  <RELATION> fastquote </RELATION>
  <HREF> GET http://fast.quote.com </HREF>
  <SCHEMA> Ticker:string, NewsURL:string,
  Headline:string </SCHEMA>
</HEADER>
<BODY>
  <PAGE>
  <METHOD> GET </METHOD>
  <URL>
  http://fast.quote.com/fq/quotecom/headlines?mode=
  headlines&symbols=##Ticker##&mode=NewsHeadlines
  </URL>
  <PATTERN> <FONT SIZE=2 FACE="TIMES NEW ROMAN"><A
  HREF="##NewsURL:(.*)##"><B>##Headline:(.*)##</B
  ></A><BR> </PATTERN>
  </PAGE>
</BODY>
```

Nasdaq Specification

```
<HEADER>
  <RELATION> nasdaq </RELATION>
  <HREF> GET http://www.nasdaq-amex.com/ </HREF>
  <SCHEMA> Ticker:string, Cname:string</SCHEMA>
</HEADER>
<BODY>
  <PAGE>
  <METHOD> GET </METHOD>
  <URL> http://www.nasdaq-
amex.com/asp/quotes_multi.asp?mode=Stock&symbol=#
#Ticker##</URL>
  <PATTERN><td colspan=4
valign=middle><font><b>##Cname:(.*)##</b></PATTE
RN>
  </PAGE>
</BODY>
```

Bloomberg Specification

```
<HEADER>
  <RELATION> bloomberg </RELATION>
  <HREF> GET http://quote.bloomberg.com </HREF>
  <SCHEMA> Ticker:string, LastTrade:Real,
Changepts:String, Changept:String, High:Real,
Low:Real, Open:Real, Volume:Real, NewsURL:string,
Headline:string </SCHEMA>
</HEADER>
<BODY>
  <PAGE>
  <METHOD> GET </METHOD>
  <URL>
http://quote.bloomberg.com/analytics/bquote.cgi?v
iew=bq&version=marketslong99.cfg&ticker=##Ticker#
# </URL>
  <PATTERN> <BR><LI><A
HREF="##NewsURL:(.*)##"><SPAN
CLASS="headln">##Headline:(.*)##</SPAN></A></LI>
</PATTERN>
  </PAGE>
</BODY>
```

Appendix C: Query Trace

The trace is not entirely self-explanatory to anyone unfamiliar with the workings of the application. Comments italicized and in bold are annotations to provide some understanding.

Query 1

```
SELECT    Ticker, LastTrade, High, Low, Headline
FROM      usatoday, fastquote
WHERE     Ticker in ["AAPL", "IBM"]
```

Query 1 Trace

query begins by parsing the SQL query

Start

selectnode

selectlist

Column: Ticker

Column: LastTrade

Column: High

Column: Low

Column: Headline

fromnode

Identifier: usatoday

Identifier: fastquote

wherenode

Binary operator : null

comparator: in

Column: Ticker

inlist

Value: "AAPL"

Value: "IBM"

SQL Parser Version 1.1: SQL program parsed successfully.

Done with step 1

retrieves and parses the appropriate specification files

2 spec files need for this query

Number of spec files: 2

first file - specification for USA Today

c:/wrapper/registry/usatoday.spec

Got the table

<http://quote.bloomberg.com/usatoday/bquote.cgi?ticker=>

Ticker

<TD ALIGN=CENTER VALIGN=TOP>

LastTrade:(.*)

</TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-

1">##Changepts:(.*)##</TD><TD ALIGN=CENTER VALIGN=TOP><FONT

FACE="arial,helvetica"

SIZE="-1">##Changepct:(.*)##</TD><TD ALIGN=CENTER VALIGN=TOP><FONT

FACE="arial,helvetica" SIZE="-1">##High:(.*)##</TD><TD ALIGN=CENTER

VALIGN=TOP><FONT FACE="arial,helvetica" SI


```

ZE="-1">##Low:(.*)##</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT
FACE="arial,helvetica" SIZE="-1">##Open:(.*)##</FONT></TD><TD ALIGN=CENTER
VALIGN=TOP><FONT FACE="arial,helvetica"
SIZE="-1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
Changepts:(.*)
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Changept:(.*)##</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT
FACE="arial,helvetica" SIZE="-1">##High:(.*)##</FON
T></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Low:(.*)##</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT
FACE="arial,helvetica" SIZE="-1">##Open:(.*)##</FONT></TD><TD
ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
Changept:(.*)
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##High:(.*)##</FONT></TD><TD ALIGN=CENTER VALIGN=T
OP><FONT FACE="arial,helvetica" SIZE="-1">##Low:(.*)##</FONT></TD><TD ALIGN=CENTER
VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">##Open:(.*)##</FONT></TD><TD
ALIGN=CENTER VALIGN=TOP><FONT
FACE="arial,helvetica" SIZE="-1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
High:(.*)
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Low:(.*)##</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT
FACE="arial,helvetica" SIZE="-1">##Open:(.*)##</FONT></TD><TD
ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
Low:(.*)
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Open:(.*)##</FONT></TD><TD ALIGN=CE
NTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
Open:(.*)
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-
1">##Volume:(.*)##</FONT>
</FONT></TD><TD ALIGN=CENTER VALIGN=TOP><FONT FACE="arial,helvetica" SIZE="-1">
Volume:(.*)
</FONT>

```

Made the parser

second file: specification for Fastquote

c:/wrapper/registry/fastquote.spec

Got the table

<http://fast.quote.com/fq/quotecom/headlines?mode=headlines&symbols=>

Ticker

&mode=NewsHeadlines

<A HREF="

NewsURL:(.*)

Headline:(.*)

Made the parser

Done with step 2

attributes are taken from the query and stored into an array-like structure

```
( (class: class collections.LLMap) (size:13) (elements: ( (83) (fastquote) ) ( (82) (NewsURL) ) ( (81) (Volume) ) ( (80) (Open) ) ( (79) (Changept) ) ( (78) (Changepts) ) ( (77) (Headline) ) ( (76) (Low) ) ( (75) (High) ) ( (74) (LastTrade) ) ( (73) (Ticker) ) ( (72) (bloomberg) ) ( (71) (usatoday) ) ) )
```

attributes required by the query

Ticker
LastTrade
High
Low
Headline

search for plan to execute the mediated query

Now to start the actually plan phase....

Plan found...

a plan is found

```
select( true,true,true,true,false,true, , TRUE)
regex_fn( com.oroinc.text.regex.Perl5Pattern@2269c4 )
join-submit ( [http://fast.quote.com/fq/quotecom/headlines?mode=headlines&symbols=,0,
&mode=NewsHeadlines] )
select( true,true,false,false,true,true,false, , TRUE)
regex_fn( com.oroinc.text.regex.Perl5Pattern@226af0 )
join-submit ( [http://quote.bloomberg.com/usatoday/bquote.cgi?ticker=, 0, ] )
constanttable ( [AAPL, IBM] )
Done with step 3
```

query execution; remote data sources are accessed and results formulated according to the query execution plan

In select, run()
Data display: done
In regex, run()
In multiscan, run()
In select, run()
Data display: done
In regex, run()
In multiscan, run()
In constanttable, run()
In constanttable: done
In multiscan: done

scanning the USA Today sites first

IBM,209.1875,+4.3750,+2.14,211.0000,203.4375,208.5000,4515500
in select:

select the appropriate attributes; in this case Ticker, LastTrade, High, Low

attributes: true,true,false,false,true,true,false,false,
cond: TRUE
IBM
209.1875
+4.3750
+2.14
211.0000
203.4375
208.5000
4515500
AAPL,46.0000,+3.0000,+6.98,47.1250,44.0000,44.0000,13140600
in select:

attributes: true,true,false,false,true,true,false,false,

cond: TRUE

AAPL

46.0000

+3.0000

+6.98

47.1250

44.0000

44.0000

13140600

In regex_fn:done

In select: done

In multiscan: done

scanning the Fastquote sites second

IBM,209.1875,211.0000,203.4375,news?story=9966179&symbols=IBM,IBM Design Center Helps Customers Build Advanced e-businesses

in select:

select the appropriate attributes; in this case Ticker, LastTrade, High, Low, Headline

attributes: true,true,true,true,false,true,

cond: TRUE

IBM

209.1875

211.0000

203.4375

news?story=9966179&symbols=IBM

IBM Design Center Helps Customers Build Advanced e-businesses

IBM,209.1875,211.0000,203.4375,news?story=9966169&symbols=IBM,New IBM S/390 G6 Servers Power e-business and ERP, Extend Leadership In Data Centers

in select:

attributes: true,true,true,true,false,true,

cond: TRUE

IBM

209.1875

211.0000

203.4375

news?story=9966169&symbols=IBM

New IBM S/390 G6 Servers Power e-business and ERP, Extend Leadership In Data Centers

IBM,209.1875,211.0000,203.4375,news?story=9965085&symbols=IBM,WALL ST WEEK AHEAD - Bulls navigate data, Goldman

in select:

attributes: true,true,true,true,false,true,

cond: TRUE

IBM

209.1875

211.0000

203.4375

news?story=9965085&symbols=IBM

WALL ST WEEK AHEAD - Bulls navigate data, Goldman

IBM,209.1875,211.0000,203.4375,news?story=9964954&symbols=IBM,RealNetworks Releases MP3 Software

in select:

attributes: true,true,true,true,false,true,

cond: TRUE

IBM

209.1875

211.0000
203.4375
news?story=9964954&symbols=IBM
RealNetworks Releases MP3 Software
IBM,209.1875,211.0000,203.4375,news?story=9963163&symbols=IBM,IBM To Unveil Powerful
Mainframe
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.1875
211.0000
203.4375
news?story=9963163&symbols=IBM
IBM To Unveil Powerful Mainframe
IBM,209.1875,211.0000,203.4375,news?story=9962419&symbols=IBM,IBM To Unveil Powerful
Mainframe
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.1875
211.0000
203.4375
news?story=9962419&symbols=IBM
IBM To Unveil Powerful Mainframe
IBM,209.1875,211.0000,203.4375,news?story=9961295&symbols=IBM,New IBM mainframes to ship in
May, ahead of plan
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.1875
211.0000
203.4375
news?story=9961295&symbols=IBM
New IBM mainframes to ship in May, ahead of plan
IBM,209.1875,211.0000,203.4375,news?story=9961160&symbols=IBM,IBM may testify for US in
Microsoft case -NY Times
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.187
5
211.0000
203.4375
news?story=9961160&symbols=IBM
IBM may testify for US in Microsoft case -NY Times
IBM,209.1875,211.0000,203.4375,news?story=9953491&symbols=IBM,SmartPortfolio.Com Announces
Investment Opinion
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.1875

211.0000
203.4375
news?story=9953491&symbols=IBM
SmartPortfolio.Com Announces Investment Opinion
IBM,209.1875,211.0000,203.4375,news?story=9951266&symbols=IBM,U.S. names 12 firms to vie for
\$25 bln in contracts
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
IBM
209.1875
211.0000
203.4375
news?story=9951266&symbols=IBM
U.S. names 12 firms to vie for \$25 bln in contracts
AAPL,46.0000,47.1250,44.0000,news?story=9953491&symbols=AAPL,SmartPortfolio.Com Announces
Investment Opinion
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9953491&symbols=AAPL
SmartPortfolio.Com Announces Investment Opinion
AAPL,46.0000,47.1250,44.0000,news?story=9947436&symbols=AAPL,DIARY - U.S. technology news
events from May 3
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9947436&symbols=AAPL
DIARY - U.S. technology news events from May 3
AAPL,46.0000,47.1250,44.0000,news?story=9947183&symbols=AAPL,Singapore-Based Singapore
Computer Systems Agrees to Distribute F5 Networks' Products
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9947183&symbols=AAPL
Singapore-Based Singapore Computer Systems Agrees to Distribute F5 Networks' Products
AAPL,46.0000,47.1250,44.0000,news?story=9946495&symbols=AAPL,ValleyJobs.com Offers Free Job
Postings for Silicon Valley Companies
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250

44.0000
news?story=9946495&symbols=AAPL
ValleyJobs.com Offers Free Job Postings for Silicon Valley Companies
AAPL,46.0000,47.1250,44.0000,news?story=9944102&symbols=AAPL,SOFTBANK Names Roizen
Partner & Portfolio Mentor Capitalist
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9944102&symbols=AAPL
SOFTBANK Names Roizen Partner & Portfolio Mentor Capitalist
AAPL,46.0000,47.1250,44.0000,news?story=9938629&symbols=AAPL,Japan PC Shipments Jump 10%
In FY98 - Study 04/29/99
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9938629&symbols=AAPL
Japan PC Shipments Jump 10% In FY98 - Study 04/29/99
AAPL,46.0000,47.1250,44.0000,news?story=9933445&symbols=AAPL,Wright Williams & Kelly Adds
Another Top 10 IC Manufacturer to Its TWO COOL Client List
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9933445&symbols=AAPL
Wright Williams & Kelly Adds Another Top 10 IC Manufacturer to Its TWO COOL Client List
AAPL,46.0000,47.1250,44.0000,news?story=9927212&symbols=AAPL,Investors Forecast Surveys Show
Optimism in Net and Tech Stocks for This Friday.
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9927212&symbols=AAPL
Investors Forecast Surveys Show Optimism in Net and Tech Stocks for This Friday.
AAPL,46.0000,47.1250,44.0000,news?story=9917713&symbols=AAPL,Friends And Family Call
Entrepreneurs ``Paupers Or Princes, Bums Or Workaholics'' -- Survey
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000

news?story=9917713&symbols=AAPL
Friends And Family Call Entrepreneurs ``Paupers Or Princes, Bums Or Workaholics" -- Survey
AAPL,46.0000,47.1250,44.0000,news?story=9907330&symbols=AAPL,Pixar sees big year ahead with
new "Toy Story"
in select:
attributes: true,true,true,true,false,true,
cond: TRUE
AAPL
46.0000
47.1250
44.0000
news?story=9907330&symbols=AAPL
Pixar sees big year ahead with new "Toy Story"
In regex_fn:done
In select: done

Query 2

```
SELECT    Ticker, Time, LastTrade
FROM      yahoo
WHERE     Ticker in ["MSFT", "YHOO", "INTC", "IBM"]
AND LastTrade < 100
```

Query 2 Trace

query begins by parsing the SQL query

Start

selectnode

selectlist

Column: Ticker

Column: Time

Column: LastTrade

fromnode

Identifier: yahoo

wherenode

Binary operator : and

comparator: in

Column: Ticker

inlist

Value: "MSFT"

Value: "YHOO"

Value: "INTC"

Value: "IBM"

comparator: <

Column: LastTrade

Value: 100

SQL Parser Version 1.1: SQL program parsed successfully.

2nd one

Done with step 1

retrieves and parses the appropriate specification files

1 spec file need for this query

Number of spec files: 1

specification for Yahoo

c:/wrapper/registry/yahoo.spec

Got the table

http://quote.yahoo.com/q?s=

Ticker

&d=v1&o=t

<a href="/q?s=.*&d=t".*?\s+

Time:(.*?)

\s+##LastTrade:(.*?)##\s+##Changepts:(.*?)##\s{2,}##Changepct:(.*?)##\s+##Volume:(.*?)##

<small>

\s+

LastTrade:(.*?)

\s+##Changepts:(.*?)##\s{2,}##Changepct:(.*?)##\s+##Volume:(.*?)## <small>

\s+

Changepts:(.*?)

\s{2,}##Changepct:(.*?)##\s+##Volume:(.*?)## <small>


```
\s{2,}
Change pct:(.*)
\s+##Volume:(.*)## <small>
\s+
Volume:(.*)
<small>
Made the parser
Done with step 2
```

attributes are taken from the query and stored into an array-like structure

```
( (class: class collections.LLMap) (size:9) (elements: ( (77) (Volume) ) ( (76) (Change pct) ) ( (75) (Change pcts) ) ( (74) (Time) ) ( (73) (LastTrade) ) ( (72) (Ticker) ) ( (71) (yahoo) ) )
```

attributes required by the query

```
Ticker
Time
LastTrade
```

search for plan to execute the mediated query

```
Now to start the actually plan phase....
Plan found..
```

a plan is found

```
select( true,true,true,false,false,false, , int 2 :: INT_ATT:0/0 < 100)
regex_fn( com.oroinc.text.regex.Perl5Pattern@223c3c )
join-submit ( [http://quote.yahoo.com/q?s=, 0, &d=v1&o=t] )
constanttable ( [MSFT, YHOO, INTC, IBM] )
Done with step 3
```

query execution; remote data sources are accessed and results formulated according to the query execution plan

```
In select, run()
Data display: done
In regex, run()
In multiscan, run()
In constanttable, run()
In constanttable: done
In multiscan: done
MSFT,12:41PM,79 9/16,+7/16,+0.55%,12,189,100
```

```
in select:
attributes: true,true,true,false,false,false,
cond: int 2 :: INT_ATT:0/0 < 100
MSFT
12:41PM
79 9/16
+7/16
+0.55%
12,189,100
```

scanning the Yahoo sites

```
YHOO,12:41PM,157 7/8,-3 15/16,-2.43%,2,070,600
in select:
```

select the appropriate attributes; in this case Ticker, Time, LastTrade

```
attributes: true,true,true,false,false,false,
```

relational operations on LastTrade

```
cond: int 2 :: INT_ATT:0/0 < 100
INTC,12:41PM,58 15/16,-9/16,-0.95%,7,513,400
in select:
```

attributes: true,true,true,false,false,false,
cond: int 2 :: INT_ATT:0/0 < 100
INTC
12:41PM
58 15/16
-9/16
-0.95%
7,513,400
IBM,12:41PM,238 9/16,+1 1/16,+0.45%,1,786,800
in select:
attributes: true,true,true,false,false,false,
cond: int 2 :: INT_ATT:0/0 < 100
In regex_fn:done
In select: done