DISCOVER the Web-Database

A Project Report
Submitted in Partial Fulfilment of the
Requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

by

Nalin Gupta

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

July 1997

Contents

Acknowledgments

I'm indebted to Dr. *Jayant R. Haritsa* for being a constant source of inspiration throughout the duration of the project. Many insightful discussions with him directionalized and enabled me to enhance the project design and realize it. His constant urging motivated me to perform at my peak. I thank him for many other things I learnt while working with him. Database System Laboratory provided all the resources and support required for the project and even for my other interests. My colleagues Binto, Ramesh, Mouli, Vidya, Nars and Shri provided a very accommodating and learning environment at DSL.

I'm greatful to Dr. Ramesh Hariharan and Maya, who provided valuable suggestions. Thanks to George A. Mihaila for providing the basic understanding of WebSQL. JavaCC team comprising of Srini, Sankar and Rob along with the users on JavaCC mailing-list gave valuable feedback in designing the parsers.

The efforts made by Narender provided the much needed help to shape this thesis. KVS, Amit, Mouli, DT and Arghh were always available source of information from CS to many other interesting aspects of life. I always relied on KVS, Malay, Senthil and Sachin with whom I spent most of my time at IISc. The love and affection I received from them made my stay at IISc a memorable one. Thank to Muthu and Pramod who made a "sweet" channel of communication across DSL and SSL. With the wonderful MEI batch of ours, I always felt at home. I'm thankful to Jagneet for being a marvelous friend and a constant source of motivation.

Love and patience shown by my parents contribute beyond any quantification to the purpose of my stay at IISc. The support from my younger brother Vicky can be understood by me only.

Abstract

The WWW is an ubiquitous, open, large heterogeneous, distributed Hypertext System. It has become a major means of disseminating information. The Openness and uncontrolled growth of the web has posed problems in searching or querying the web for relevant information. Currently searching information on the web involves the use of search engines which use keywords supplied by the user to search through a centralized index of web pages. The centralized index seriously affects the performance of the search and additionally the index often contains outdated information.

Our goal is to provide the user with up-to date information by querying existing documents on the web. To this end we treat the web as a database and adapt conventional database querying techniques to search the web for required information. This thesis discusses the design and implementation of the Discover system; a distributed architecture for querying the web. Discover uses WSQL, a SQL like query language to specify queries which can exploit both the structure and content of the web. Web servers under the Discover system co-operate with each other by servicing/forwarding user queries taking care to avoid multiple computations of the same query. We have also provided a protocol to allow the user to determine when the query is completely answered by the web.

Chapter 1

World Wide Web

1.1 What is W3?

The Internet is the world wide computer network consisting of millions of computers. The network information retrieval (NIR) techniques are used to transfer information between computers across the Internet. The list of the NIR techniques [Fos94] includes Email, Telnet, (Anonymous)FTP, News, WAIS, and Gopher. Since early 90's, the Hyper-Text Transfer Protocol (HTTP) is also used as one of the NIR techniques. The HTTP protocol has been described later in the section 1.3. The HTTP provides the medium called *Hyper-Media* on the Internet to access information. This information is a collection of *hypertext documents* maintained on the computers linked to the Internet. A hypertext document allows interaction with it. Along with information a hypertext document contains interactive regions called *Hyper-Links or Links.* A link is a reference to another hypertext document. A user views a hypertext document using a tool called *Browser.* The user interacts with links on this hypertext document using mechanisms like clicking the mouse over the interactive region. This interaction leads the user to another hypertext document. The user keeps following links in this manner to browse through a subset of hypertext documents maintained on this hyper-media. In Figure 1.1 hyper-links contained in hypertext documents residing on different servers is shown. This method of accessing information on the Internet has created an environment, coined the *World Wide Web*

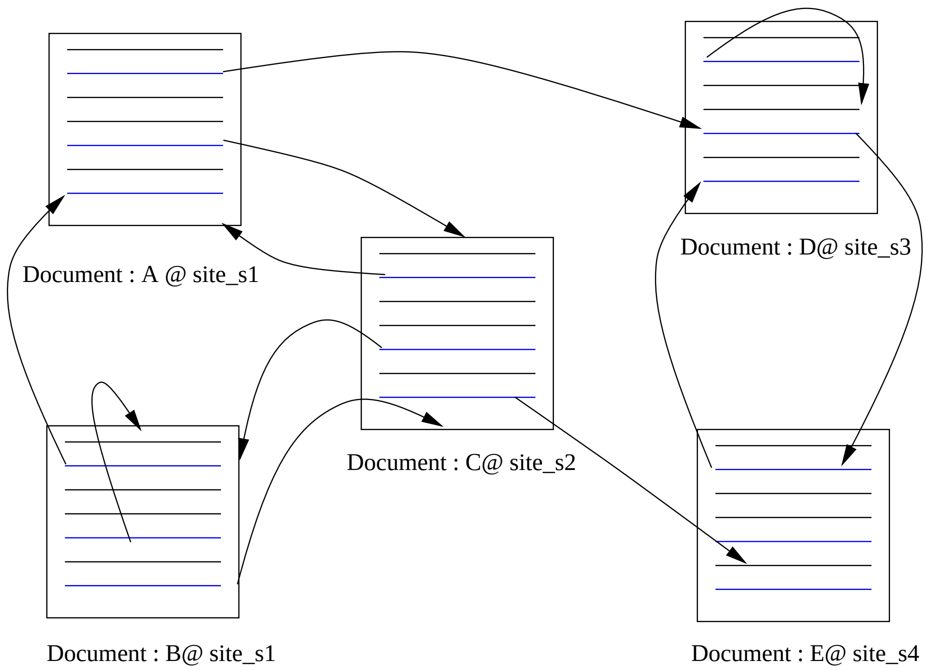

Figure 1.1: Browsing through hypertext documents

(alias WWW, Web, W3) [BLCGP92, BLCG92, Bou, Wor]. The Web has developed into a uniform, easily accessible, and user-friendly interface to many of the NIR techniques, as explained in section 1.4. A few NIR techniques that have been mapped to the web are News, WAIS, Gopher, FTP archives, and Mailto. The web hides the details of the various NIR techniques from the user. It overcomes the problem of heterogeneous platforms, data formats and access protocols (NIR techniques) used.

1.2 The W3 Architecture

The web has merged the techniques of hypertext, information retrieval, and wide-area networking, to produce a global information system. It uses the *client-server* architecture as shown in Figure 1.2. The following entities participate in this architecture :

- **Client** resolves a document address into a document using its repertoire of network protocols.

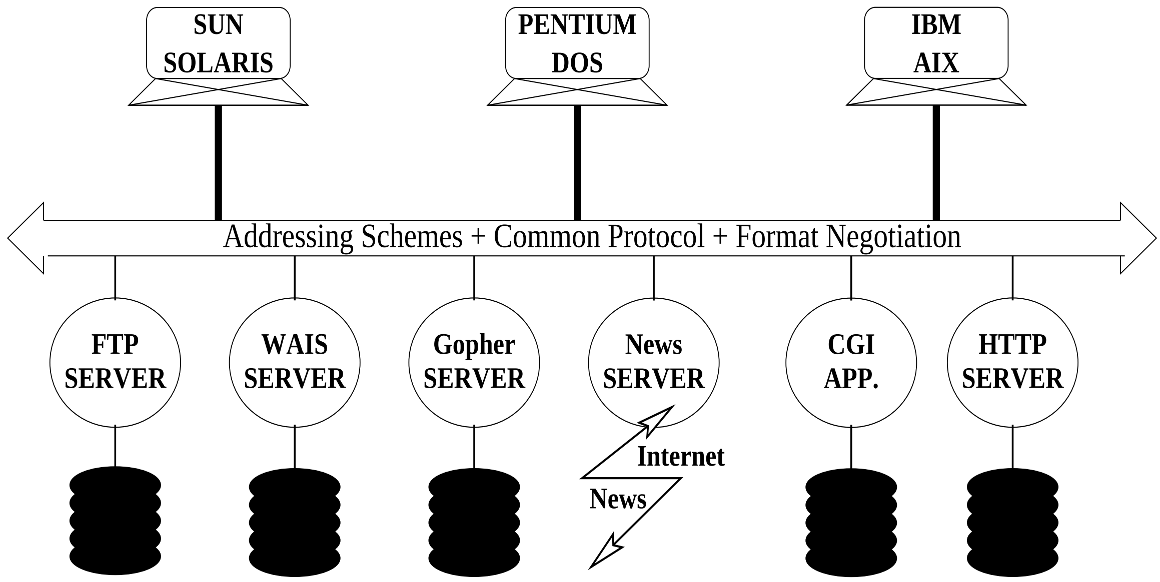

Figure 1.2: W3 Client-Server Architecture

- **Server** provides data in a simple hypertext or plain text form, or by negotiation with the client, in any other data format.

1.3 Hyper-Text Transfer Protocol

The Hyper-Text Transfer Protocol (HTTP) [BLFN96, FGM$^+$97] provides uniform handshaking and format conventions for client/server communication. It is an Internet protocol, similar in its readable, text-based style to the FTP and NNTP Protocols that have previously been used to transfer files and news on the Internet. The data transfered may be plain text, hypertext, images, sounds, applets etc. Unlike these protocols, however, HTTP, is *stateless* [BLCG92]. That is, although it runs over a TCP connection, the connection is held only for the duration of one operation. This operation involves a client computer establishing a connection with the server computer on the Internet by making a request, accepting a response, closing the connection, and taking some specified action. This stateless model is efficient when a link from one object may lead equally well to an object stored on any other (remote) server. In addition, it is possible that a web-resource at some URL is replaced by another web-resource. Such a situation advocates the HTTP statelessness, as then it would retrieve on the basis of the currently specified URL, but not on its previous experiences.

1.4 Uniform Resource Locator

In addition to the HTTP, the web provides information accessible using a few other NIR techniques such as FTP, Mailto, NNTP, WAIS, and Gopher. An information accessible through the web is termed as *Web Resource.* The web identifies each web-resource using a common address (naming) system called *Uniform Resource Locator (URL)* [Fie95, BLMM94, BL93]. URL is a string describing the location of a web-resource and the protocol required to access the web-resource. URLs are "Universal" in the sense that they encode members of the universal set of network addresses. For a new network protocol that has some concept of object, one can form an address of any object by specifying the set of protocol parameters necessary to access the object. For example,

- The URL of the main page of my department is:
  http://www.csa.iisc.ernet.in/

- The URL of the news group "csa.comp.java" served by the news server currently running in CSA department is:
  nntp://news.csa.iisc.ernet.in/csa.comp.java

- The URL of the faculty profile of our dept is:
  http://www.csa.iisc.ernet.in/People/Faculty/sketch.html

- The URL to ftp the gcc compiler from a site within our campus is:
  ftp://144.16.64.2/pub/compilers/gcc-2.7.2.bin.tar.gz

The above examples show the similarity of URLs' specification, with that of UNIX's file/directory-path specification. In this thesis we are interested in "HTTP" protocol and "HTML" document, with respect to that the formal URL syntax used to access html document is :

$\langle$protocol$\rangle$://$\langle$domainname$\rangle$[:$\langle$portnumber$\rangle$]/$\langle$dir-path$\rangle$/$\langle$objectname$\rangle$[#$\langle$spot$\rangle$]

It is composed of six parts, that works as follows:

1. *protocol/data-source* : The prefix specifies the NIR protocol used to access the web resource. The syntax for this part is : "**http://**" points to a hypertext document (typically, an HTML file) accessible thorugh the HyperText Transfer Protocol.

2. *domain name* : This is the domain name for the web server where the required web document or resource resides.

3. *portnumber* : This number denotes the port address, at which the server provides the service, to be used by the HTTP protocol. To specify portnumber is optional. Most of the web servers run HTTP daemon process at a default port number, like other common NIR protocols. The default portnumber 80 is used for HTTP daemon process.

4. *directory path* : This specifies the location of a web resource on the server's file system.

5. *object name* : This is the actual name of the web resource required.

6. *spot* : In case of hypertext documents, finer specification is provided by the "tagged" spot within the document. This is done by preceding the name of an HTML anchor with the pound sign (#) and tacking it onto the HTML file name.

For example : The URL of my project mentioned in my current home-page is :

"http://dsl.serc.iisc.ernet.in/~nalin/index.html#PRJ" .

The formal syntax for the URL specifications of other NIR protocols is described in rfc1738 [BLMM94].

1.5 Hyper-Text Markup Language

The Hyper-Text Markup Language (HTML) [BLC95, Con97] defines the internal structure of hypertext documents maintained on the web. It accomplishes this by a tagging convention that identifies contained or referenced resources. For example, a sensitive (click-able)

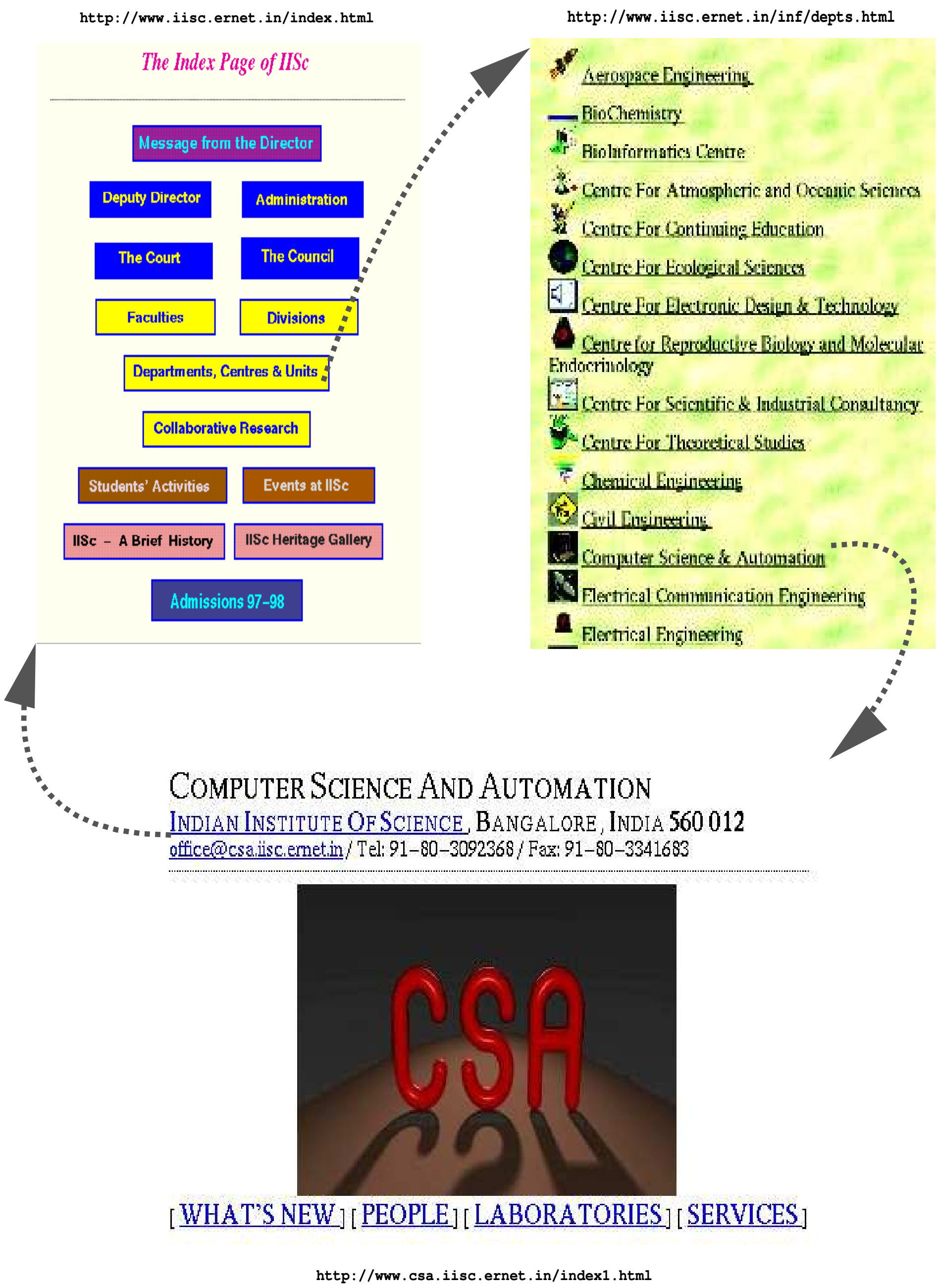

Figure 1.3: Hyper-Links across HTML Documents

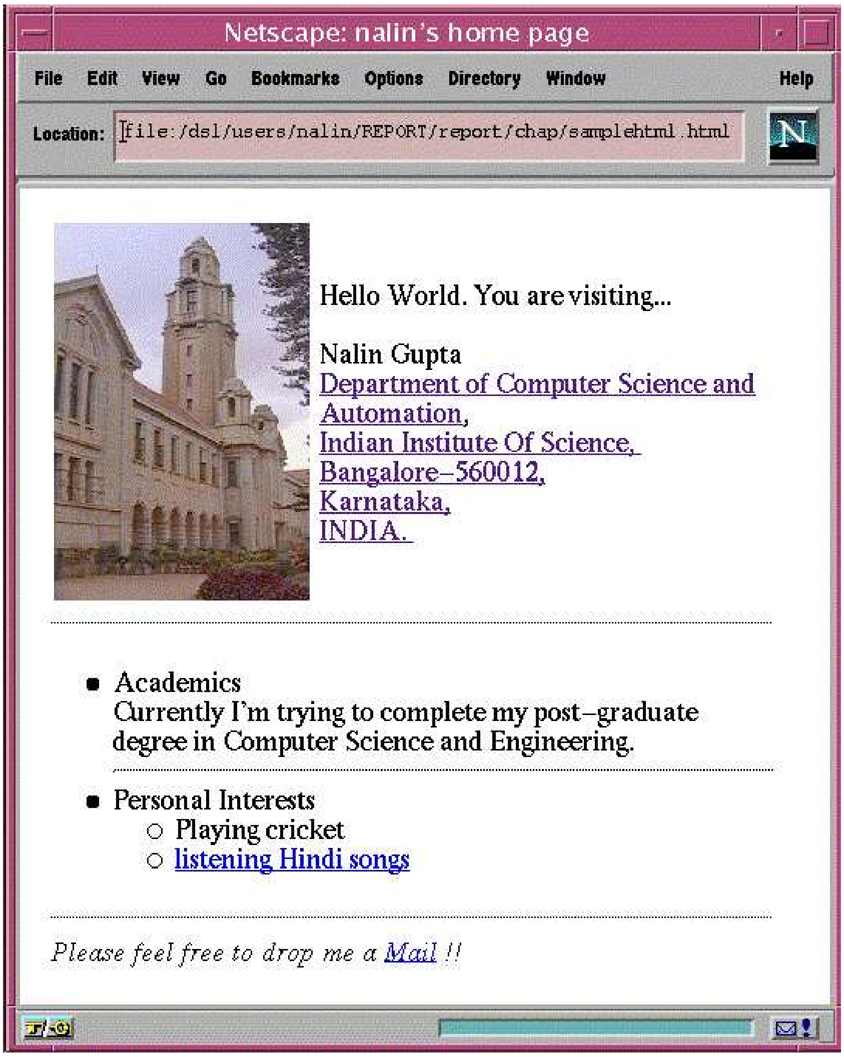

Figure 1.4: A Sample HTML File

document anchor which points to a web resource is couched within the tag pairs "<A HREF=...>" and "</A>"; an image is identified by the tag "<IMG SRC=...>", and so on. The HTML provides the W3 with a common basic language for interchanging hypertext. The HTML information which corresponds to a web resource is called its *source*. The hyper-links across few HTML documents in our campus is shown in Figure 1.3. The listing of the sample HTML document shown in Figure 1.4 is :

A Sample HTML File

```
<HTML>
<HEAD>
   <TITLE> nalin´s home page </TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE CELLSPACING=0 CELLPADDING=0>
<TR ALIGN=LEFT VALIGN=CENTER>
<TD > <CENTER> <IMG SRC="central-building.gif"> </CENTER> </TD>
<TD>
Hello World. You are visiting...<BR><BR>
Nalin Gupta<BR>
<A href="http://www.csa.iisc.ernet.in/">
Department of Computer Science and Automation</A>, <BR>
<A href="http://www.iisc.ernet.in/">
Indian Institute Of Science,<a> <BR>
Bangalore-560012,<BR>
Karnataka,<BR>
INDIA.
</TD>
</TR>
</TABLE>
<hr>
<UL>
```

```
<li> Academics<BR>
<a name="#PRJ">
Currently I´m trying to complete my post-graduate degree
in Computer Science and Engineering.
<hr>
<li> Personal Interests<BR>
<ul>
<li> Playing cricket
<li> <a href="http://144.16.79.158/resources/audio_files.html">
  listening Hindi songs</A>
</UL>
</UL>
<hr>
<ADDRESS>
Please feel free to drop me a
<a href="mailto:nalin@csa.iisc.ernet.in">Mail</A> !!
</BODY>
</HTML>
```

1.6 Browsing on the W3

A user runs a web-client program called *Web-Browser* or *Browser* to view a hypertext document and browse through a part of the web as described in Section 1.1. Commonly used web-browsers are Netscape, Mosaic, Lynx, InternetExplorer, etc. A user starts browsing the web using either of the following way :

1. Specifies a known URL.

2. Uses a tool available on the web called *Search Engines.* Search engines are index servers and provide references (URLs) to web-resources maintained on the web. A user queries these search engines by sending them a query string. Search engines query their indices using this query string. Then they return a ranked list of URLs of those documents that may have some related information to the query string sent by the user. A few well-known

search engines are AltaVista, Excite, Yahoo, etc.

1.7 Hypertext Organization

On the basis of physical locations of hypertext documents on the Internet, we define the following terms for the hypertext organization :

1. A *logical hypertext organization* is a set of documents on the web, that are reachable from a given URL.

2. A *physical hypertext organization* is a set of documents on the web in same physical sub-domain of the Internet.

1.8 Summary

The WWW is an ubiquitous, open, large heterogeneous, distributed Hypertext System. It has added to the list of NIR techniques a new protocol HTTP, and given an identification scheme that uses URLs for the information resources available on the Internet. Its simplicity of use and feature to add information to the network has made it synonymous with the Internet.

Chapter 2

Integration of W3 and DBMS

The Openness and uncontrolled growth of the web has posed problems in searching or querying the web for relevant information. This chapter discusses this problem and the current solution, *search engines.* It also reviews the search engines' limitations and the attempts to resolve them by applying the "database technology" to the web. Such attempts have brought up a new term *Web Database.* Later, a few other notions of the Web Database are discussed.

2.1 Lost in the Web

Browsing on the web is analogous to Non-Deterministic Finite Automaton (NFA) like processing, which is used for operations like testing whether a given string belongs to the language represented by the NFA or not. A user starts from a URL in search for an information. Usually, he follows a link from this document to obtain another hypertext document. He keeps following links in this manner, until he finds a relevant information. Browsing in such a manner often disorients the user. This could be either due to a wrong link followed or the user's unawareness about the logical hypertext organization from this URL. This is known as infamous "Lost-in-Cyberspace" [KS95] or "Lost-in-hyperspace" [MMM96] or "Lost-in-Web" phenomenon, due to which a user encounters the following limitations :

- uncertainty to determine that a new path taken after moving back by a few steps would lead to related information or not.
- unnecessary network bandwidth consumption,

- human fatigue.

Search engines are alternative solution used by a user. They may provide direct references to a collection of informations related to the user. As discussed in Section 2.2, the search engines have a few limitations too.

2.2 Search Engines

Search engines are index servers available on the web and provide references (URLs) to web-resources maintained on the web. A user queries these search engines by sending them a query string. Search engines query their indices using this query string. Then they return a ranked list of URLs of those documents that may have some related information to the query string sent by the user. A few well-known search engines are AltaVista, Excite, Yahoo, etc. Search engines uses a tool called *Web-Robot* or *Robot* [Kos] to construct their indices. A *robot* is a program that automatically traverses the web's hypertext structure by retrieving a document, and recursively retrieving all documents that are referenced. Web robots are sometimes referred to as *Web Wanderers, Web Crawlers,* or *Spiders*. These names are a bit misleading as they give the impression that the software itself moves between sites (like a virus); this is not the case, however a robot visits sites by simply requesting documents from them. Though search engines provide a valuable service, they have a few limitations :

1. There is no barrier in publishing information on the Internet. Due to which, the central indices maintained by the search engines do not map to millions of documents on millions of web sites. In addition, most of the existing search engines are independent of each other. These reasons imply replication of information across the search engines and a user needs to query multiple search engines.

2. A user faces the problem of forming "good" queries, as many search engines exist and they support different formats.

3. A user may like to perform "structural queries" as explained in Section 2.3 in combination to keywords based "content queries". At present, the search engines support only keywords based queries.

4. A lot of information on the web is often maintained for a short period. In addition, most of the robots either do not revisit sites or do so very rarely. Due to these reasons, the result from the search engines may contain "floating references". A floating-reference is a reference to an information that no longer exists.

5. Typically, a search engine queries its index for a list of keywords. It may have a few entries which satifies for few of these keywords. Usually, current search engines do not filter such duplicates.

6. The search engines do not restrict the query to either physical or logical hypertext organization. For example A user wishes to know a list of departments of the IISc. She may like to restrict the query within the *physical sub-Internet of the IISc* or within the *logical hypertext organization* reachable from the IISc home-page.

7. A user has to follow links obtained from a search engine to find whether they lead him to related information or not.

2.3 Web Queries

The web users like to perform various types of queries on the web, such as :

- A user likes to find documents on the web which contain related information or keywords. For example Find documents related to or containing keywords "Web Database".

- A user knows the starting URL. With a physical or logical hypertext organization constraint, he likes to know how to reach another document. For example Find documents containing a word "senthil" within IISc (physical); *OR* Starting from CSA home-page, how to reach documents containing "websql" (logical).

- Another possibility is that a user wants to know how many documents within a physical or logical hypertext organization contain a reference to a particular document. For example A user wants to know how many documents within IISc sub-Internet contain reference to Dr. MJT's home-page maintained on CSA web site (physical); *OR* a user wishes to know how many documents reachable from IISc home-page contain reference to Dr. JRH's home-page maintained on DSL web site.

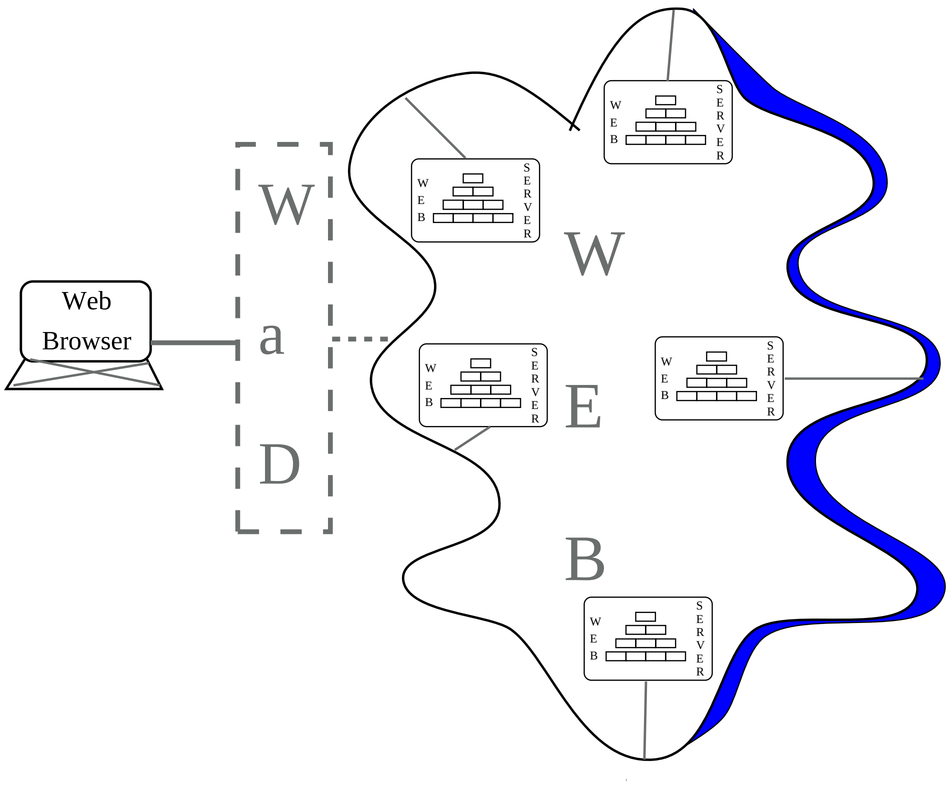

Figure 2.1: The Web viewed as a Database

- A user likes to know the average length of all HTML documents or average number of hyperlinks contained by all HTML documents maintained within a physical or logical hypertext organization. For example The DSL web administrator likes to know the average number of hyperlinks to resources either within IISc (physical) or anywhere (logical).

The above list covers many requirements, but it may not be complete. Typically, such queries can be classified as a combination of the following two types of queries [KS95] :

Content Query is specified to query the content of a document.

Structural Query is specified to query the information conveyed in the hypertext organization. The entity selected by such a query is a set of URLs (nodes) from the hypertext structure that satisfy a given graph.

2.4 Web as Database : WaD

A few of the limitations to search information on the web as discussed in Section 2.2 are addressed by attempts applying conventional database features to query the web. Such attempts try to overcome the restrictions posed by the central search indices by directly querying the documents maintained on the web. The web is viewed as a huge unstructured repository [KS95, MMM96] of information as shown in Figure 2.1. It is analogous to a huge file system maintained across wide-area network with few useful indices. These projects suggest a "virtual schema" for the hypertext documents maintained on the web and construct a temporary database for the documents to process a web-query. A case study of few of these projects is discussed in chapter 3. With such a facility, the user can directly query the web and can exploit :

- web's current state,
- web's graph like structure,
- web's document contents,
- query could be restricted to physical or logical domain of the web,
- any partial information,
- standard query language for the web.

The projects studied [LSS96, KS95, MMM96] achieve these features at different level. These attempts to apply the database technology to the web has started a new field called *Web-Database* or viewing *Web as Database* (WaD).

2.5 Web Database : other notions

Unlike WaD, there have been a few orthogonal attempts to integrate database technology to the web. These projects reflect the domain of application and how the web has been viewed. This has caused the same term *Web Database* to be used for different purposes. The classifications of the integration of the web and database, other than the WaD, are explained below.

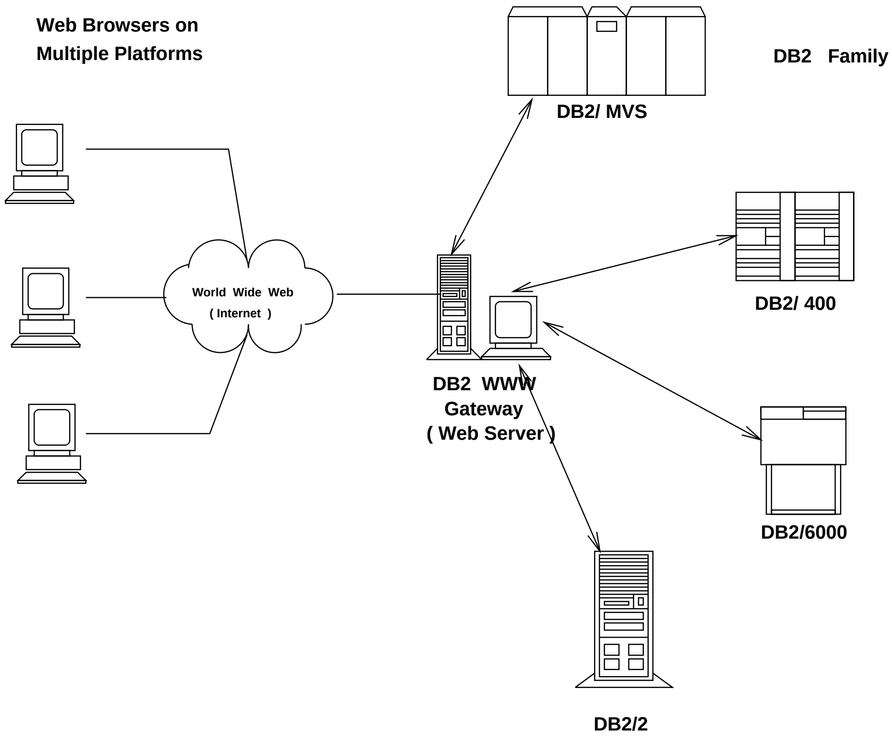

Figure 2.2: The Database on the Web

2.5.1 Database on the Web : DoW

Many organizations maintain up-to-date information about themselves using a database system, like DB2. They provide different views of their organization to various people related to them. (For example organization officials, customers and general public). These organizations exploit the low-cost and easily accessible feature of W3. Web servers of such organizations are equipped with the *Common Gateway Interface*[1] (CGI) techniques [CGI95], enabling related users to access the corresponding view of the Database [NS96]. This is possible using HTML forms [Per95]. Such web servers provide a starting HTML form to be filled and submitted by the user. This form is processed by executing the CGI-script at the server. The CGI-script produces another related HTML document, which may be a form again. This document corresponds to the view of the database/organization, valid as per the user identity/requirement. Communication in such a manner between the client's browser and the web server, lets an organization provide access to its database to related users from anywhere on the web. This has also led to the concept of

[1]CGI-script is a legal executable program residing at a Web Server, which is recognized using URL Syntax and executed by the Web Server on behalf of the user to dynamically produce HTML Documents.

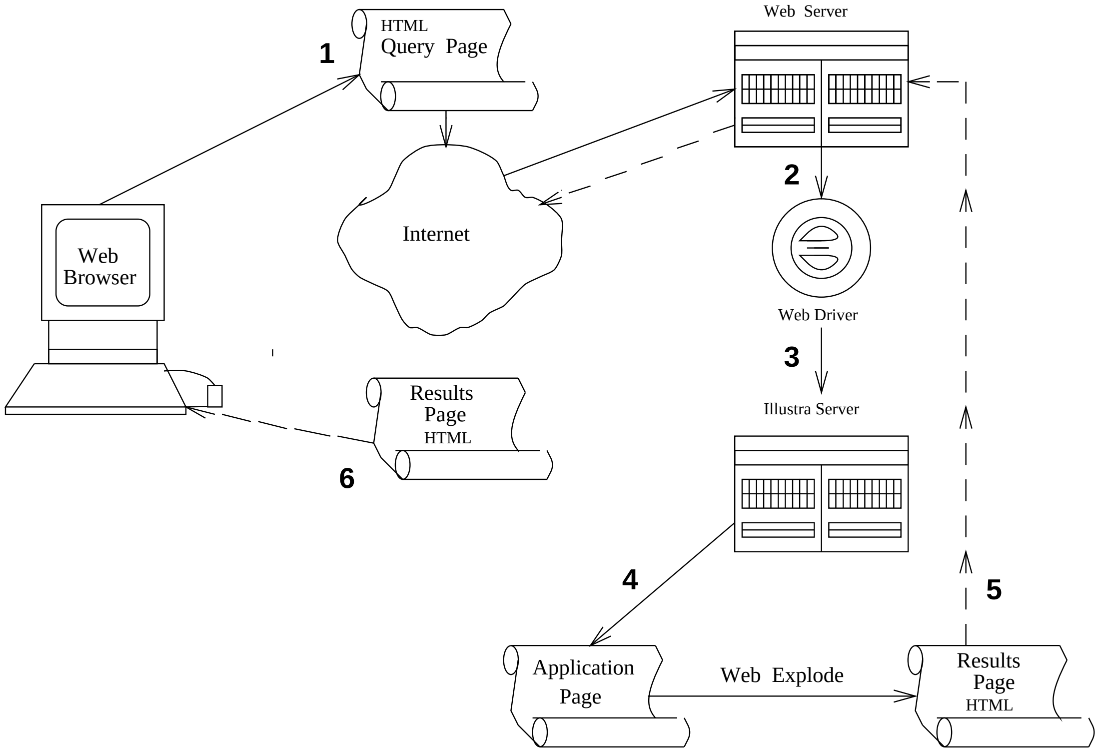

Figure 2.3: A Web-Site on the Database (Web DataBlade Architecture)

closed Internet, called *Intranet,* within the organization. We term this extension of the purpose of the web, as *Database on the Web* (DoW). Figure 2.2 illustrates how database servers are connected to the world wide web as suggested by project *DB2-WWW Connection* [NS96]. A list of a few such projects include JDBC [JDB], DB2-WWW Connection [D2W95, NS96].

2.5.2 Web on the Database : WoD

For larger organizations, having their web sites as a collection of HTML files is a major maintenance problem. They require to dynamically generate HTML documents, equip their sites with data/content query processing power, analyze their log records to improve and tune their performance, and organize their sites better. This motivates to move from a standard file system to a database system [Gaf96]. The web server or HTTP daemon process is equipped with features similar to that of database, to index HTML documents. The HTTP requests to this web server won't see this difference. Such databases are required to handle complexities such as heterogeneous data (For example html files, images, applets, associated CGI-scripts), their indexing, querying, caching etc. We term such maintenance of a "web site" using a database server, as

Web on the Database (WoD). Figure 2.3 shows *Illustra's Web Datablade System,* which provides a toolset to maintain database-driven Web-Sites [Gaf96].

2.6 Summary

The limitations of browsing as well as existing searching techniques often disorients the user. A user can query search engines using a few keywords only. Problems, such as to follow the links obtained from search engines to find whether they are relevant or not, and the user's unawareness with the layout of the web site, still persist. The need to directly query the web has become an important requirement. With appropriate modifications in representing query and storing data along with supports for the heterogeneous data, could be an alternative solution for quering the web.

Chapter 3

A Case Study - Related Projects

This chapter gives a brief description of a few projects which address solutions to the problems in searching informations on the web. These projects apply a few new methods similar to that of the conventional database to query the web[1]. In general, they suggest "virtual schema" to the unstructured web or a schema-less web resources. A *virtual schema* is a scheme suggested for either web or web resource to construct database for them. (For example recognizing few tables and their respective attributes in a manner similar to that used in Relational Database Systems). Then these systems construct a temporary database for a web resource to process a web-query. The projects studied are WebLog [LSS96], W3QS [KS95], and WebSQL [MMM96].

3.1 WebLog

The framework to resolve the problem of searching information on the web, through a new declarative, logic-based query language, called the *WebLog* is suggested by the project [LSS96]. The goals of the WebLog are:

- to process query on the existing documents on the web, rather than catalogs.
- to provide a declarative interface for querying as well as restructuring. Restructuring implies a format specification given by a user for a result of a web-query.
- to exploit partial knowledge that the user may have on the information being queried,

[1]Henceforth, we would call a query to the web as a *web-query.*

- to provide the user either built-in programming predicates (library functions, like isa, substring) or allow to use user-written functions in form of built-in programming predicates for specifying a query.

3.1.1 Model

Within an HTML document, the WebLog recognizes regions consisting of related informations. Usually, an HTML document consists of "groups" of related information that are spatially close together in the page. Information within each such group is expected to be homogeneous. Each such group is called a *Rel-Infon.* A user can specify an HTML tag (For example ⟨HR⟩, ⟨LI⟩, ⟨BR⟩) as the delimiter to split a document in terms of rel-infons. Each rel-infon is given a unique *id.* (This id could be the token appearing in a header associated with a rel-infon, the most prominent 'keyword' in a rel-infon, or even the byte offset of the start of the rel-infon from the start of a page.) A rel-infon has attributes which come from a set consisting of strings 'occurs', 'hlink', and various tags (such as ⟨title⟩, ⟨b⟩, ⟨em⟩) that adorn strings in an HTML document. The attributes of a rel-infon map to values that are strings, except for the hlink attributes that is mapped to a hlink-id[2]. The role of various attributes are :

- 'occurs' is mapped to a set of strings occurring in a rel-infon.
- 'hlink' is mapped to a set of hlink-id's of hyperlinks appearing in a rel-infon,
- the tag attributes, if defined, are mapped to the tokens they adorn in the rel-infon. (the tag 'title' is special in the sense that it is mapped to the same string for all rel-infons).

3.1.2 Query

A Query written in WebLog is an expression consisting of terms of type atomic formula, molecular formula, and built-in predicates. The syntax[3] for these terms are as follows:

Atomic Formula :

- ⟨url⟩[⟨rid⟩: ⟨attr⟩→→⟨val⟩]

[2]An id associated with each hyperlink in the document, consisting two components : the hypertext, and the href.

[3]For complete details, refer [LSS96]

- $\langle \text{url} \rangle [\langle \text{rid} \rangle \text{: } \langle \text{attr} \rangle \longrightarrow \langle \text{val} \rangle]$
- $\langle \text{url} \rangle [\langle \text{attr} \rangle]$
- $\langle \text{url} \rangle$

Molecular Formula :

- $\langle \text{url} \rangle [\langle \text{rid} \rangle \text{: } \langle attr_1 \rangle \rightarrow\!\!\rightarrow \langle val_1 \rangle, \ldots, \langle attr_n \rangle \rightarrow\!\!\rightarrow \langle val_n \rangle]$

Built-in Predicates :

- *href(* $\langle$*hlink-id*$\rangle$*,* $\langle$*url*$\rangle$ *)*
- *htext(* $\langle$*hlink-id*$\rangle$*,* $\langle$*string*$\rangle$ *)*
- *substring(* $\langle$*string*$\rangle$*,* $\langle$*string*$\rangle$ *)*
- *isa(* $\langle$*string*$\rangle$*,* $\langle$*type*$\rangle$ *)*

Example: 1 : Collect all citations(hyperlinks) referring to HTML Documents that appear in the "Database Systems and Logic Programming" home-page at URL :

"http://www.informatik.uni-trier.de/ ley/db/index.html".

The collection should also contain the title of the document the citation refers to.

WebLog's representation :

$$ans.html[title \longrightarrow `allcitations', hlink \rightarrow\!\!\rightarrow L, occurs \rightarrow\!\!\rightarrow T]$$
$$\longleftarrow leyurl[hlink \rightarrow\!\!\rightarrow L], href(L, U), U[title \longrightarrow T].$$

leyurl is a substitution for the URL "http://www.informatik.uni-trier.de/ ley/db/index.html". Variable L in the first subgoal ranges over all the hyperlinks in *leyurl*. The built-in predicate *href* is used to navigate over the citations in the page at *leyurl* . The rule generates a new HTML document *ans.html* that is a collection of all citations in *leyurl* , annotated with the title of the cited example.

3.2 W3QS

A world wide querying system has been suggested by the project W3QS [KS95]. The goals of the project are :

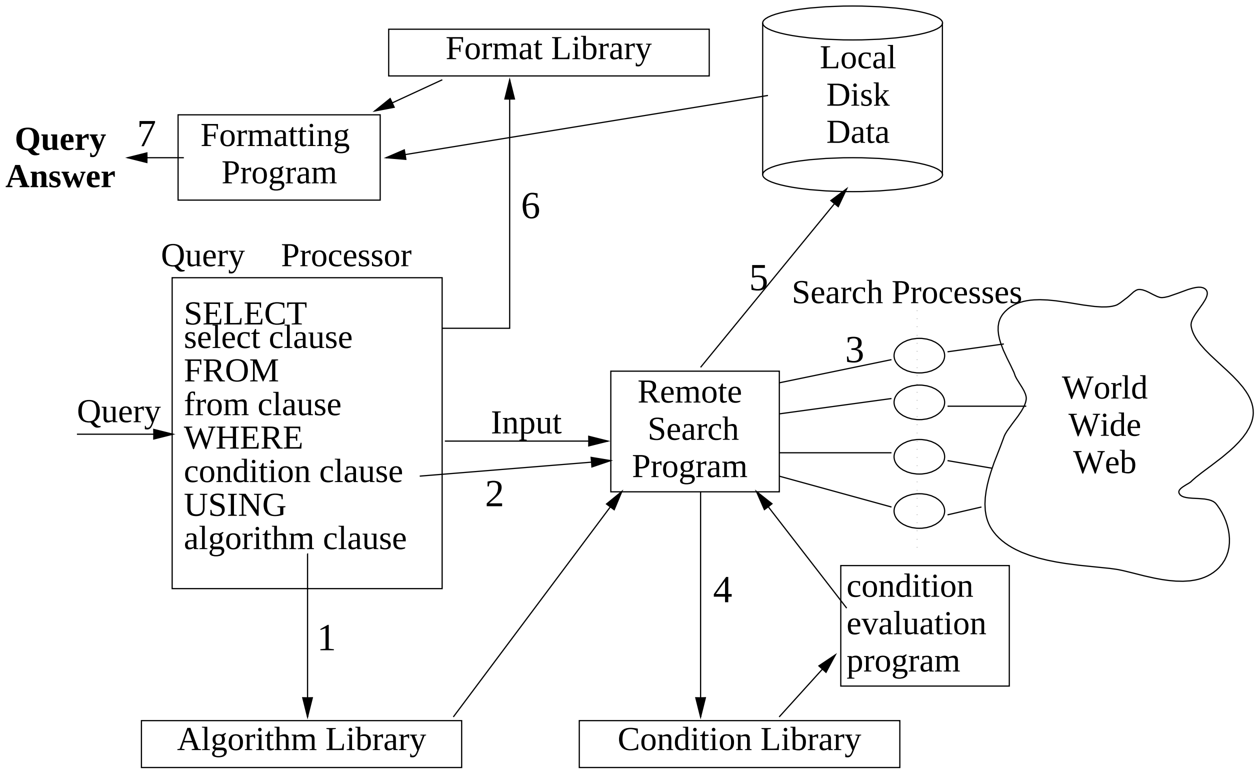

Figure 3.1: The W3QS Model

- an extensible system.
- the syntax and semantics of a high level query language, that captures both structural and content aspects of the web. The project has proposed a new declarative query language, called *W3QL*.
- using existing UNIX (or user written) utilities through a user-friendly interface.
- an advanced display facilities for the collected information.
- a view maintenance facility.
- a flexibility to deal with a variety of file formats.
- a provision to deal with the HTML form, which may be required to be filled-in, while processing the query. Uses the past experience (if any) to fill that form.

3.2.1 Model

The W3QS system is shown in the Figure 3.1. The Query Processor receives query, and uses algorithm clause(1)[4] to obtain search program. The Remote Search Processes(3) uses this search program to fetch the documents from the web. Using condition library(4), the information corresponding to the query is selected. At the end of the search, the query processor formats(6) the information collected to present the result(7).

3.2.2 Query

The structural aspect of a query is specified through a directed graph called *Pattern Graph,* where :

- There is at most one edge between any two nodes.
- There is at most one edge from a node to itself (self-loop).

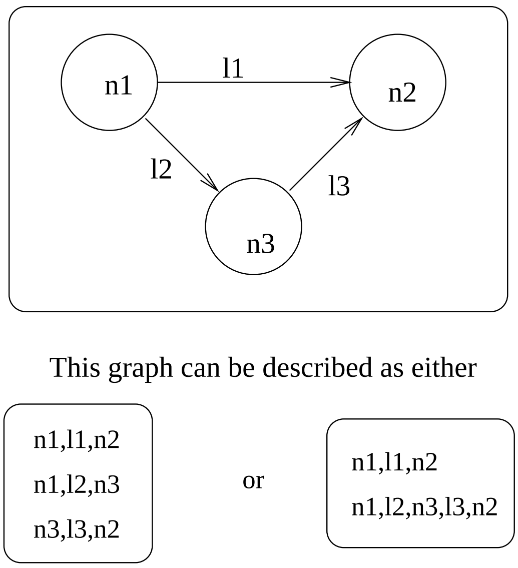

Figure 3.2: a pattern graph with bounded length

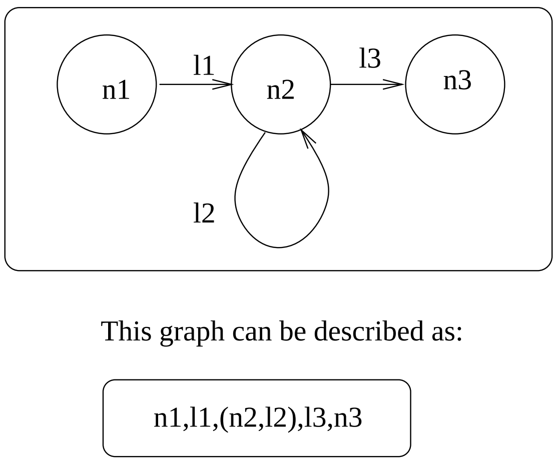

Figure 3.3: a pattern graph with unbounded length

In Figure 3.2, an example where the user describes a path with *bounded length* pattern is shown.Whereas, if a user is unawared of the hypertext organization that he is querying then he

[4]These numbers refer to operation indicated in the Figure 3.1.

may use *unbounded length* pattern graph, as shown in Figure 3.3. The syntax and the semantics of the query language W3QL is shown through the following example taken from [KS95].

Example: 2 : Start from URLs mentioned in file “importantIndices.url”and search for images in GIF format appearing in HTML files following only links with the REL[5] argument equals to “example”.

W3QL's representation :

SELECT *cp* n3/* result;
FROM (n1,l1),l2,n3;
WHERE
  n1 IN importantIndices.url;
  SQLCOND (n1.format = HTML) AND
    (l1.REL = “example”) AND (n3.name = “*.gif”) ;

Typically, the query finds all the subgraphs of WWW that are similar to the pattern specified in the from-clause. Each subgraph whose nodes satisfy the where-clause, defines a *row* in the returned table. The row contains the nodes and links used to exhibit similarity.

3.3 WebSQL

A relational model to the schemaless web has been suggested by the project [MMM96]. Each web resource is associated with a *tuple* in a *virtual relation* :

Document[url, title, text, type, length, modif]

The *url* is the key, and all other attributes can be null. This relation is used for a *content query.* The *structural queries* are addressed by associating each hyperlink in a hypertext document (an HTML file) with a tuple in another *virtual relation* :

Anchor[base, href, label]

where *base* is the URL of the HTML document containing the link, *href* is the referred document and *label* is the link description. The three possible types of hyperlinks in as HTML file recognized are :

1. *interior* if the destination is within the source document;

[5]an HTML argument for an Anchor tag ⟨A⟩

2. *local* if the destination and source documents are different but located on the same server;

3. *global* if the destination and the source documents are located on a different server.

3.3.1 Model

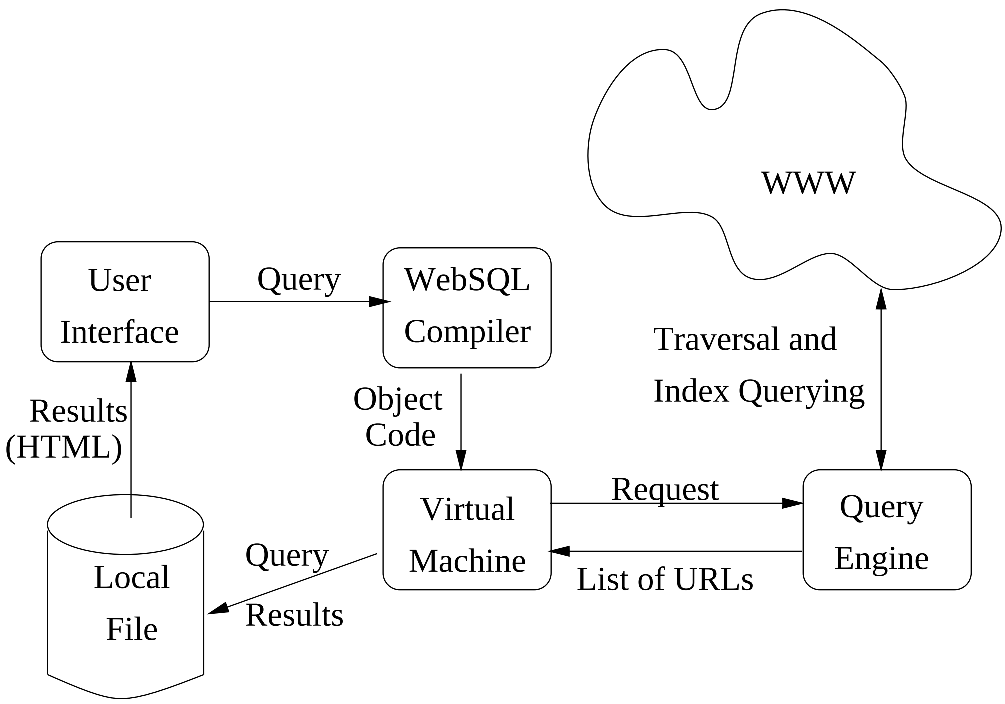

Figure 3.4: The WebSQL System

The WebSQL System Architecture is shown in Figure 3.4. The WebSQL *compiler* parses the query and translates it into a nested loop program in a custom-designed object language. The object code is executed by a *virtual machine interpreter* that implements a stack machine. Virtual machine invokes *query engine* to perform the actual evaluation, which may involve retrieving the document, parsing it and then building a 'document' and an 'anchor' virtual tables.

3.3.2 Query

WebSQL uses *Path Regular Expression* (PRE) to represent the path to be traversed on the web for a structural query. PRE uses alphabets from the set $\{\mapsto, \rightarrow, \Rightarrow, =\}$ to form a regular expression, where:

('$\mapsto$' denotes interior link), ('$\rightarrow$' denotes local link), ('$\Rightarrow$' denotes global link), and ('$=$' denotes null link or empty path). For example : The PRE for a set of paths that start from some node (HTML file) and traverse :

- two global links followed by at most one interior link; Or
- any number of local links followed by at least one global link.

is $(\Rightarrow . \Rightarrow .(= \mid \mapsto) \mid (\rightarrow)^{*}. \Rightarrow . \Rightarrow^{*})$.

Symbols concatenation (.), alternation(|) and repetition (*) have their conventional meanings.

Example 3 : Starting from the Department of Computer Science home page, find anchors in all documents that are linked through paths of length two or less containing only local links. Keep only the documents containing the string 'database' in their title.

WebSQL's representation :

SELECT *a.base, a.href, a.label*
FROM Document *d* SUCH THAT "http://www.cs.toronto.edu" $= \mid \rightarrow \mid \rightarrow . \rightarrow d$,
Anchor *a* SUCH THAT $a.base = d$
WHERE *d.title* CONTAINS "database";

Unlike conventional SQL, here the *from-clause* is evaluated by navigating the web. Documents are downloaded recursively by following the hyperlinks present in the last processed document, using the combination of stack machine and NFA processing over PRE. On receiving a final document corresponding to a path belonging to the PRE, that document is subjected to *where-clause* evaluation. If the document satisfies, the *temporary virtual database* being constructed for this query is updated.

3.4 Summary

The projects discussed in this chapter suggested a few methods to model web as a database. All three projects have a common approach to view the web as a graph and at present process a web-query on a local machine. A web resource (typically HTML file) is retreived from the web to answer a content web-query. To process a structural web-query (in combination with a content query), a document is downloaded from the web at a time and selective hyperlinks contained in the document are followed. In order to follow multiple hyperlinks obtained from a document, a query engine either implements a stack machine [MMM96] or coordinates a set of parallel processes [KS95].

Chapter 4

The DISCOVER System

Motivated by the challenges in the emerging field of the *web database,* in this chapter we propose the *Discover* system to resolve many of the difficulties faced while searching information over the web. The Discover system enables a user to query the web to search and retreive related information. The Discover system introduces searching or query processing technique analogous to the combination of Distributed Query Processing and Multi-threaded programming. The query processing technique is *distributed* because the query is processed on *independent* web sites over the web. In the Discover system, the notion of "distributed processing" implies that a web site receives a query for a web-resource maintained at it. The query may have a sub-query, required to be processed at this site. The web site processes it and forwards the rest of the query to the next set of web sites, using hyperlinks contained in the processed web-resource (HTML file). This leads to multiple copies (clones), explained later, of the rest of the query traversing different parts of the web. So at the same instant, the copies of the query are being processed along the various paths on the web. This requires web sites to *co-operate* among themselves to answer the query, as they forward the query on the user's behalf. Such features gave the Discover system its name, which means *Distributed Independent Servers Co-operating Voluntarily to Extract web-Resources.*

The following sections explain the evolution of the Discover system and the issues that arose while designing it. A few examples and definitions are introduced, when required to explain a feature of the Discover system.

4.1 Goals and Features

The Discover system is a new approach for largely the same problem, i.e. querying the web, as differently addressed by [LSS96, KS95, MMM96]. In this section, we state the goals of the Discover system, and the inherited features from the above mentioned projects. Also, we mention how our approach is different from these projects.

Goals

In designing the Discover system, we have extended the ideas of the projects discussed in the previous chapter to meet the following goals :

1. to process a web-query in a *distributed* manner over the web (i.e. *query-shipping* rather than *data-shipping*)
2. to process a web-query, with *minimal* overheads for the web site and *minimal* network bandwidth consumption
3. to provide a *simple* query processor
4. to provide a *query language* for the web
5. to provide *support* for heterogeneous data.

Inherited Features

The features of the Discover system that are common with the projects discussed in the previous chapter are :

1. The web is modeled as a directed graph, where
   - each web-resource on the web is treated as a *vertex* or a *node* and a set of nodes are maintained at each web-site
   - each hyperlink[1] to another web-resource is an *edge*

[1] Henceforth we do not distinguish between a hyperlink and a link.

2. The relational model for WWW suggested by [MMM96] forms the base of the Discover system. In addition, we merge the concept of a rel-infon [LSS96] to this model to provide users with more flexibility to query information, constraint within regions.

3. Use Path Regular Expression (PRE) to specify "connecting-path" defined in the following section. We use the following symbols :

   - '`#>`' for interior hyperlink
   - '`=>`' for local hyperlink
   - '`->`' for global hyperlink
   - '`@>`' for null hyperlink (i.e. empty path)

   PREs are built from these symbols using concatenation (·), alternation (`|`) and repetition (*). Concatenation symbol is optional. For Example, `@>|->=>*` is a regular expression that represents the set of paths containing the zero length path and all the paths that start with a global link and continue with zero or more links.

New Features

The Discover system is different from WebSQL, WebLog, and W3QS systems in the following ways :

- processes a web-query over the independent web sites in a distributed manner. The web acts as a database server constituting a huge collection of nodes (i.e. web-resources), where each web-server maintains a set of such nodes. A user as a client submits his query to the web. Unlike a "client-server" model, a user submits a query to a *set* of nodes belonging to the web. As explained later, the query traverses the web and may reach a set of nodes. If any such node has query related information, it sends that directly to the user. In addition, such a node may forward the received query to the next set of nodes by following a selective list of hyperlinks contained in the node (HTML file).

- incorporates an optimization to minimize multiple computations of the same query at any node on the web.

- lets a user specify a structural query as a set of logically ordered set of sub-queries called node-queries.

4.2 Model

We propose a query mechanism for the web, which processes a query in a distributed manner, over the existing nodes rather by using central search indices. A user can exploit both the content and the structural aspects of the web, if each web site serves query requests along with HTTP requests. This can be achieved by adding standard query processing techniques at each web site, in one of the following ways :

- equip the HTTP daemon process running at each web site with a standard query processor
- equip all web sites with a standard query processor as a CGI-script
- equip all web sites with a standard query processor as a daemon process

The only difference among these is the manner by which a query is received from the network and given as input to the query processor provided by the Discover system. In order to make the explanation of the Discover system simple, we assume that each web site is running a standard query processor as a daemon process. Also, when we say that a query is being sent to a node, we mean that a query is being sent to a query server[2] which owns that node. Whereas, when we say that a node sends back the result, we mean a query server sends back the result of a query which was received corresponding to a node residing at this site. It should be noted that all the operations done by a query server are on behalf of a node for which a web-query has been received by this web site.

A user sends his query to a set of query servers corresponding to the nodes residing on them. Such nodes are called the *Starting Nodes* and the query servers to which these nodes belong are called *Starting Points.* The set of starting nodes are obtained from either the user knowledge or already existing search-indices (this process can be automated and made invisible to the user). A query server receives a query corresponding to a node, typically an HTML file, residing at its site. The query server does its part of processing for the query, and may have to send back the result to the user. Apart from this, if the query is to explore the structural organization too, then the query server may have to *forward* the rest of the query to the next set of query servers. This set of query servers is obtained from the set of hyperlinks contained in the processed HTML document. The set of next query servers also perform similar query processing operations. Such

[2]Henceforth, in context of query, we call a web server as a *Query Server.*

a processing of a web-query and its traversal over the web is analogous to distributed query processing and spawning of multiple threads. Apart from this, a query server does not have any other overheads.

4.3 Design

In this section, we present the conceptual features of the design and the working of the Discover system. The current design of the Discover system do not distinguish a node and an HTML file, as it stresses on the structural queries. The reason is the need to develop techniques to reach the nodes of the web following the hyperlinks contained in the HTML files. Then handling heterogenous resources (files) can be taken care of, as they are required for content based queries only.

Query's web traversal mechanism

To process a structural query, the query needs to traverse the web. The path to traverse the web is a combination of types of hyperlinks (*global, local and interior*), specified as a regular expression, called Path Regular Expression (PRE) [MMM96]. A query server receives a query corresponding to a node belonging to it. Using the PRE and the HTML file associated to the node, the query server determines the set of hyperlinks to be followed by the query. For each hyperlink to be followed by the query, the query server does the following (more details in Section 5.3) :

- a duplicate of the current state of the web-query object, called *clone* is created,
- modifies the PRE information carried by the clone, marking the PRE as if the next hyperlink has been followed,
- then it dispatches the clone along the hyperlink to the next node.

This effort of the query server is called *forwarding*. It should be noted that at each node, along a path, the query gets cloned to reach the next node. This leads to a multiple clones of the actual query, may be in different states, moving along various possible paths specified by the PRE. The phrase "the web traversal of the query" is used to imply that there exists multiple clones of the query which are traversing the web at that instant. A node along a path visited by a clone of the query, while traversing the web, plays either of the following roles :

1. **Server-router node** : A user specifies a condition for a node. If this node fails to satisfy this condition, then this node is the *dead-end* of the path taken by the query to reach this node. In case the node does not fail, then it processes a part of web-query called *node-query,* and sends back the results to the user. After this, if more node-queries are left in the web-query then, the node forwards the rest of the web-query (i.e. the remaining node-queries) to the next set of nodes. Such nodes with context to the web-query are called *Server-router nodes* or *Server-routers.* A Server-router which processes the last node-query is the final node along the path taken by the query to reach this node. Such Server-routers are called *destinations* and they are not required to forward the web-query, because all the node-queries of the clone of the web-query has been processed. They are included in the Server-router class because the procedure they follow to process a web-query is same as that of a proper Server-router.

2. **NonServer-router node** : The query is simply forwarded as explained earlier in this section (more details in Section 5.3).

Structural queries

A user specifies a web-query as a set of node-queries corresponding to a logically ordered set of Server-routers. The user provides PREs called *connecting-paths* to specify how the web-query should traverse the web in order to reach the next Server-router. Consider an example of a web traversal of a web-query :

Example 1 : A user wants a web-query to

- start from a node (lets denote it as ‘sp’) belonging at a set of Starting Nodes
- follow the multiple paths in accordance with connecting-path “`=>(->|=>)`”
- reach a node (lets denote that as ‘n1’) along all possible paths, which should act as a Server-router,
- follow the multiple paths from all such ‘n1’ in accordance with connecting-path “`(=>|->)`”

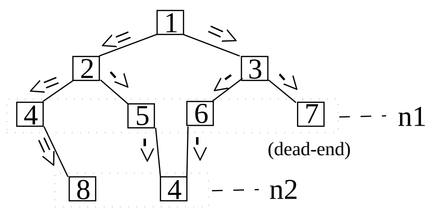

Figure 4.1: Web Traversal Path

- reach a node (lets denote that as 'n2') along all possible paths, which should act as a destination.

This is symbolically expressed as :

`sp =>(->|=>) n1 (=>|->) n2`

A user specifies a web query as a set of node-queries $\{nq1, nq2\}$, where 'nq1' and nq2' are processed at the nodes which qualifies as 'n1' and 'n2' respectively. It should be noted that Starting Node 'sp' belongs to a set of nodes, though the current implementation of the Discover system starts querying the web from one node only. The Starting Node could be either a Server-router or a NonServer-router. If the connecting-path from the Starting Node to the next Server-router does not include an empty path, then the Starting Node is a NonServer-router, as in Example 1. For this example to specify the Starting Node as a Server-router, the set of node-queries would be $\{spq, nq1, nq2\}$ and the logical nodes 'sp', 'n1', and 'n2' would be arranged as :

`sp (@>|=>(->|=>)) n1 (=>|->) n2`

Let nodes 1,2,3,4,5,6,7,8 be visited by the query as shown in the web traversal diagram in Figure 4.1. Nodes {1,2,3} are NonServer-routers, and {4,5,6,7,8} acts as Server-routers. Node 4 acts twice as a Server-router at different logical level for the query along different path as query re-visited Node 4 in a more advanced state of computation. Later in Section 4.4, we discuss such re-visits in detail, which may cause multiple computation of the same web-query.
For the sake of clarity, we introduce few definitions.

Complete-Path : A *Complete-Path* is defined as a path out of all multiple paths, which the query follows by virtue of its clones, beginning from a starting-node to a destination node. The Complete-Path in Figure 4.1 are {1,2,4,8}, {1,2,5,4} and {1,3,6,4}.

along-path-set : This set consists of both Server-routers and NonServer-routers, which a query visits along a path. Each path followed by the query is associated with such a set. This set is a *bag,* as along a path, query may revisit this node, but for different purpose. The purpose could be different because query will be in a more advanced state of computation.

across-paths-set : A user specifies his query by chaining Server-routers in a logical order. Refer to Example 1, where 'n1' and 'n2' are specified as *logical Server-routers,* and are chained by a connecting-path. Symbols 'n1' and 'n2' are called *logical-nodes.* All Server-routers at a same logical order, which are along various paths a query takes, belong to a set called *across-paths-set.* Each logical level of the order specified in a query is associated with such a set. It may be noted that Server-routers visited by a query along a dead-end path also are included in the across-path-set associated with their logical order. Such a set is also a *bag* because a Server-router may belong to multiple paths of the query corresponding to the same logical level (node).

In Figure 4.1 for the above mentioned example, across-paths-set associated with

- node n1 is {4,5,6,7} and
- node n2 is {8,4}.

Whereas there are four along-path-sets as follows :

- {1,2,4,8},
- {1,2,5,4},
- {1,3,6,4} and
- {1,3,7}. It leads to a dead-end path.

Simple query processor and query processing technique

To process a web-query, a query server dynamically creates a temporary database in memory from the HTML file. The database is constructed in memory as typically the HTML files are not huge. After processing the web-query, the query server immediately purges the database. For a structural query, the query is similarily processed and then forwarded at each node the query visits along the multiple traversal paths over the web. Such distributed processing avoids

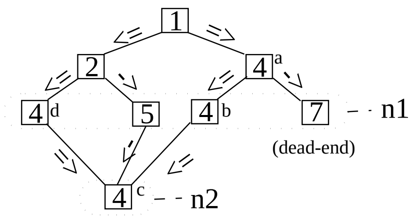

Figure 4.2: Multiple visit to a node

Node	Query-State	Node 4 Role	Logical Node
4^a	$Q^2_{g\|l}$	NonServer-router	–
4^b	Q^2_n	Server-router	n1
4^c	Q^1_n	Server-router	n2
4^d	Q^2_n	Server-router	n1

Table 4.1: Query-States seen at node 4

query processor either to be a complex stack machine [MMM96] or to coordinate parallel processes [KS95] corresponding to various possible paths. Unlike Distributed-DBMS, local indices or coordinators are not required. The query processor design can therefore be kept simple with focus on other important issues, like handling heterogeneous web resources.

4.4 Performance and Optimization

1. *Query Processing Optimization :*

   Instead of the web traversal path shown in Figure 4.1, for the Example 1, consider a different possible scenario as shown in Figure 4.2. Here multiple clones of a query reach the same node multiple times possibly at different instants. This is because node '4' belongs to multiple paths of the query. Table 4.1 tabulates the different roles played by the node 4, when it receives multiple clones of the query at different stages of their respective computation. The stage of computation done or left for a clone of the query can be easily determined as follows :

   - use the information of the number of Server-routers yet to be seen by this clone along a Complete-Path, and

- use the information of how much path it has covered along a connecting-path from the last Server-router visited by this clone to reach the next Server-router.

Each clone inherently carries this information, which could be recognized as the *Query-State.* Along any path, the Query-Sate of a query 'Q' is denoted as $Q^{sr\_left}_{cp\_left}$, where,

- *sr_left* indicates the number of Server-routers yet to be seen on the way to the destination along a Complete-Path. It includes a destination.
- *cp_left* indicates the rest of connecting-path to be followed to reach next Server-router from current node.

(At a node the query-state of a clone of a query is the received state of the clone, i.e. state just before reaching a node.)

Table 4.1 indicates that due to two clones of the query reaching Node '4' with the same query-state leads to Node '4' being added twice to same 'across-paths-set' corresponding to logical node 'n1', as 4^b and 4^d. At this Server-router, both clones seek same piece of computation for the query, and if forwarded by it, they would see the same "*gossamer*", i.e. same piece of the web. In such a situation, all later coming clones would cause re-computation. Further, consider that this Server-router forwards all later clones. Consequently, each Server-router along all residuals paths from this Server-router would re-compute the node-query corresponding to their respective logical order.

Such multiple computation of the same query is eliminated by maintaining log records at query server. A log record of a web-query is ${}^{log\_id}Q^{sr\_left}_{cp\_left}$, where :

(a) *log_id*, consists of the URL of the node and query_id of the web-query, where query_id includes user-id, ip-address of the user's machine respectively.

(b) *sr_left*, as mentioned earlier in this section.

(c) *cp_left*, as mentioned earlier in this section.

The query server uses the following algorithm to process a web-query :

Step 1. receive a web-query ${}^{id}Q^{n}_{c}$

Step 2. construct a set 'H' from its log record, where

H = $\left\{ {}^{id'}Q^{n'}_{c'} \mid id' == id \wedge n' == n \right\}$

Step 3. **if** 'H' is an empty set, **then**

    if connecting-path c is nullable, **then**

        process the web-query; // acts as a Server-router

        send the result;

    endif

    make log entry ${}^{id}Q^{n}_{c}$;

    forward the web-query;

else

    if $\exists\ {}^{id'}Q^{n'}_{c'} \in$ H, $\ni c' == c$, **then**

        purge the web-query;

    else

        make log entry ${}^{id}Q^{n}_{c}$;

        forward the web-query;

    endif

endif

In the environment like the web, the log entries should not be maintained forever. The log records may consume lots of disk space and this would be the overhead for the query-server. A decision regarding the time period to keep the log entries is required. Once the time-period of a log entry expires, the query-server deletes it from its records.

2. *Conserve network bandwidth :*

   In search of an information from the current state of the web, such distributed processing also saves network bandwidth on following grounds :

   (a) Unlike search engines or local processing as done in [MMM96, LSS96, KS95], *no web resource* (HTML file) is downloaded to perform a query operation over it.

   (b) Only query and result are transfered across the network. Usually, these are small objects.

   (c) Query carries information about the IP-Address and listening port-number of a machine on which the user is waiting to receive results. A Server-router uses this information to make a connection to a user to send the result. Therefore, a Server-router need not be concerned about which node has sent the query to it, but directly sends the result to the user. Such bypassing avoids any overloading for the forwarding nodes.

(d) Instead of sending the contents of an HTML file as the result, the hyper-references to them are provided to the user. This saves user's disk space and his time.

(e) If a query server fails to make a network connection while sending a result back to user, due to any reason (like the user finished his session, network failure, etc.), it makes a log record but purges the query and query's forwarding operation. While if network connection can not be made during forwarding a query along a hyperlink, the query-server forgets to follow this link and continues with the rest of the hyperlinks to be followed. The reason for such a strategies is not to *overload* the web site. The query-server never tries to determine why it can not make a connection. It simply assumes that the other side is off.

4.5 Query

4.5.1 The Relational Virtual Tables

The information found on the web is stored as unstructured data (files) while conventional DBMS uses structured data (such as tables). However, in order to give a database-like access to the web, information that is found in the files must be queried. To achieve this, we extend the relational schema for the web proposed by the project WebSQL [MMM96] but with slightly different interpretation. At present, the Discover system associates the following virtual relations to the web-resources :

Document : The Discover system maps a node to a tuple in Document relational table. In WebSQL, the query processing is local to the user machine and involves multiple downloads of various HTML files. So for each node downloaded, a tuple is inserted in the Document relational table. Unlike WebSQL, in the Discover system the query processing is distributed and for a node the query server processes a node-query corresponding to this node. This implies that Document relational table will have only one tuple mapped to this node.

Anchor : A hyperlink contained in a web-resource is mapped to a tuple in this relational table. A user may specify a condition for the hyperlink selection.

Relinfon : This virtual relation is introduced to provide a user more flexibility to query information in an HTML file, constraint within regions as suggested in WebLOG [LSS96]. A user may specify an HTML tag as a "delimiter" to be used to split the HTML file into regions called *rel-infons.* Default region is the entire HTML file.

The attributes of these relations in the schema suggested for the web are tabulated in Table 4.2. The virtual tables declarations specified in a node-query are used to first construct a temporary database out of a web-resource (HTML file). It should be noted that in a node-query there can be only one declaration for the Document relational table, whereas there can be multiple declaration for tables of Anchor or Relinfon types. Apart from the default specification, a user can direct how to construct a virtual table by specifying a predicate like condition. For example A user may like to construct an anchor table consisting of only local links. For that, the declaration looks like :

anchor a such that a.ltype = "l"

Such a specification we call *Virtual Table Construction Specification* or *VTSpec.* As mentioned in Table 4.2, all the attributes of a virtual relation are not used in VTSpec. Also, for all three types of virtual relational tables, the VTSpec has different semantics.

- For a table of type Document, the VTspec implies that if a node (Server-router) fails this condition, then this node is a dead-end. This condition can be evaluated only after constructing the temporary database.
- For a table of type Anchor, the VTspec implies that if a hyperlink contained in this node satisfies this condition than insert a tuple in the table mapped to this hyperlink.
- For a table of type Relinfon, the VTspec informs which HTML tag is to be used as a delimiter.

In Table 4.2, a few attributes are marked with a '*', for the following reason. An HTML file contains textual information embedded with the HTML tags. We use a string buffer to store all

Virtual Relation	Attribute	Type	VTSpec	Trans-ferable	Semantic
Document	url	string	yes	yes	Node's URL
	title	string	yes	yes	HTML doc's title
	text *	HTMLTxt	yes	no	Viewable Text
	textlength	integer	yes	yes	Text length
Anchor	base	string	yes	yes	Node's URL to which hyperlink belongs
	label *	HTMLTxt	no	yes	Active text viewable on a browser
	href	string	yes	yes	URL of referred object
	ltype	string	yes	yes	Link type (l$\vert$g$\vert$i)
Relinfon	delimiter	string	yes	yes	HTML tag used to split the file in rel-infons (default: $\langle$HTML$\rangle$ tag)
	url	string	no	yes	Nearest fragment's URL in HTML file (default:Node's URL)
	text *	HTMLTxt	no	no	Viewable text in rel-infon
	textlength	integer	no	yes	Text Length

Table 4.2: Virtual Relations for the web

textual information in the HTML file in it. We call the string stored in this buffer as *HTML-Text* . For few attributes like 'text' of relation Document, 'label' of relation Anchor and 'text' of relation Relinfon would have value which is a substring of HTML-Text. For such attributes, we define a data type called *HTMLTxt* , which contains the information of a substring, such as offset and length, in the HTML-Text.

The attributes like 'text' of relation Document and Relinfon may map to a large string, and for a web-query the user may obtain them as the result from many nodes. The Discover system instead of sending these values as a result, send hyper-reference to them. This would save user's disk space, network bandwidth, and the time spent to transfer this information across the network. Later, the user can follow the hyper-reference using a browser and obtain this information directly from the web site. Though this decision is subjective, but the current implementation of the Discover system sends the result this way. For this reason, we classifies an attribute as *transferable* if its actual value is sent as a result. Otherwise, a hyperlink with some default message is sent as a result.

4.5.2 Discover Query Language : WSQL

The Discover system provides a user with a SQL-like query language, called *WSQL.* WSQL constitutes of a set of sub-queries called *node-queries.* A single node-query is specified for a content query, whereas a set of node-queries are specified for a combination of a structural query with a content based search. A user specifies a set of node-queries for a set of nodes, whom he expects to act as Server-routers. In web-query specification and explanation of WSQL, we call these Server-routers "logical-nodes". These node-queries corresponding to a set of logical-nodes are arranged in a logical order, using connecting-paths. The connecting-paths indicate how the query should traverse the web, via a set of NonServer-routers, to reach the next Server-router as described in detail in Section 4.3. Now we discuss how to specify a node query and arrange them in a logical order. Consider the following incomplete BNF specification of the WSQL query language (refer Appendix A for complete details) :

```
-----------------------------------------------------------------------
WEB_QUERY        ::=  SELECT_CLAUSE    FROM_CLAUSE  <SEMICOLON>
-----------------------------------------------------------------------
FROM_CLAUSE      ::=  FIRST_NODE_QUERY   (<COMMA> OTHER_NODE_QUERY)*
-----------------------------------------------------------------------
FIRST_NODE_QUERY ::=  STARTING_NODE_TABLES_DECL    [WHERE_CLAUSE]
-----------------------------------------------------------------------
OTHER_NODE_QUERY ::=  OTHER_NODE_TABLES_DECL    [WHERE_CLAUSE]
-----------------------------------------------------------------------
STARTING_NODE_TABLES_DECL ::=
<DOCUMENT> NODE_VAR <SUCHTHAT>
           URL_STRING   PATH_REG_EXPR NODE_VAR  (<AND> DOCUMENT_VTSPEC)?
(<COMMA> ANCHOR_RELINFON_TABLE_DECL)*
-----------------------------------------------------------------------
OTHER_NODE_TABLES_DECL    ::=
<DOCUMENT> NODE_VAR <SUCHTHAT>
           LAST_NODE_SOME_ANCHOR_TABLE_VAR   PATH_REG_EXPR   NODE_VAR
           (<AND> DOCUMENT_VTSPEC)?
(<COMMA> ANCHOR_RELINFON_TABLE_DECL)*
```

```
-----------------------------------------------------------------------
ANCHOR_RELINFON_TABLE_DECL ::=
           <ANCHOR>    TABLE_VAR  (<SUCHTHAT> ANCHOR_VTSPEC)?
|          <RELINFON>  TABLE_VAR  (<SUCHTHAT> RELINFON_VTSPEC)?
-----------------------------------------------------------------------
```

Instead of a single *select-clause* , a single *from-clause* and an optional *where-clause,* as in conventional SQL, WSQL has virtual tables declaration and an optional *where-clause* for each logical node. At user interface level WSQL provides a single *select-clause* . However, in the web-query object created for this query, the select-clause is splitted into the set of attributes list, such that table variable names associated with the attributes in such a list are declared in one and only one node-query. The WSQL specification and the web-query object would look like :

```
  ----                                          ----
  | SELECT tableVarName.attrName  ...            |  ----
  | FROM:                                        |  | select ...
  |     ----                                     |  | from   ...
  |     |                                       W|  | [where]
  |     | NODE QUERY                            E|  ----
W |     |                                       B|    .
S |     ----                                     |    .
Q |       .                                     Q|    .
L |       .                                     U|  ----
  |       .                                     E|  | select ...
  |     ----                                    R|  | from   ...
  |     |                                       Y|  | [where]
  |     | NODE QUERY                             |  ----
  |     |                                       ----
  |     ----
  ----
```

The comparison operators currently supported by the WSQL are given in Table 4.3. Whereas, the logical operators supported are conventional “AND” and “OR” operators denoted as “and”

and "or" respectively.

Relation Operator	Semantic	Operands Type
=	equals to	string/integer
<	less to	string/integer
>	greater than	string/integer
<=	less than or	string/integer
	equal to	string/integer
>=	greater than or	string/integer
	equal to	string/integer
! =	not equal to	string/integer
contains	string comparison	string
like	string comparison (case insensitive)	string

Table 4.3: Relational operators supported by WSQL

We explain the semantics of the WSQL using few examples in the increasing order of complexity :

4.5.3 Examples of Content Query

In this section, we give few examples to specify just content queries. A simple content query has only one node-query, and the Starting Point itself is the node in query.

Query 1 Find whether the CSA home-page contains the keyword "student" or not.

```
select       d.text
from         document  d  such that "http://www.csa.iisc.ernet.in/" @> d
             where
                       d.text contains "student" ;


This query can be also written as :
select    d.text
from      document  d  such that "http://www.csa.iisc.ernet.in/" @> d
                                              and  d.text contains "student" ;
```

Query 2 Select all the anchors in a region of the CSA home-page which contains a keyword like "sTuDenT". Split the file in rel-infons using the tag ⟨hr⟩ as a delimiter.

```
select    a.href, a.label
from      document  d  such that "http://www.csa.iisc.ernet.in/" @> d ,
          anchor    a ,
          relinfon  r such that r.delimiter = "hr"
          where
                    r.text like "sTuDenT" ;
```

Query 3 Select only local or global anchors in a region of the CSA home-page which contains the keyword "student". Split the file in rel-infons using the tag ⟨hr⟩ as a delimiter. Also select the text in that region.

```
select    a.href, a.label
from      document  d  such that "http://www.csa.iisc.ernet.in/" @> d ,
          anchor    a ,
          relinfon  r such that r.delimiter = "hr"
          where
                    r.text contains "student"
               and  (   a.ltype = "l"
                     or a.ltype = "g" ) ;
```

Query 4 Construct an anchor table for only local hyperlinks contained in the CSA home-page, and select only those anchors whose label as well as the region containing them has the keyword "student". Split the file in rel-infons using the tag ⟨hr⟩ as a delimiter. Also select the text in that region.

```
select    a.href, a.label, r.text
from      document  d  such that "http://www.csa.iisc.ernet.in/" @> d ,
          anchor    a  such that a.ltype = "l" ,
          relinfon  r such that r.delimiter = "hr"
          where
                    a.label contains "student"
               and  r.text contains "student" ;
```

4.5.4 Examples of Structural Query

Query 5 Starting from the CSA department, find the URL of all the nodes which are in the CSA department and have the keyword like "lAb".

```
select    d.url
from      document  d  such that "http://www.csa.iisc.ernet.in/" =>* d
                                                    and d.text like "lAb" ;
```

This query can also be expressed as below. Here, any node visited can not be a dead-end because there exists no VTSpec condition in the declaration for the Document relational table 'd'. Also, as mentioned in the beginning of this chapter that '*' is limited to the maximum of 10 repetitions, this implies that for this particular query and the PRE (`=>*`), the total number of nodes seen along any path would be 10 and each node visited would process the node query.

```
select    d.url
from      document  d  such that "http://www.csa.iisc.ernet.in/" =>* d
          where
                    d.text like "lAb" ;
```

Query 6 Restrict the above query to at most 6 local jumps.

```
select    d.url
from      document  d  such that "http://www.csa.iisc.ernet.in/" =>*6 d
                                                     and d.text like "lAb" ;
```

Query 7 Start from CSAGUPTA's home-page on krithika to find his class-mate "vidya", who did her project on Database Mining and took system programming course under Dr. MJT. Find all the anchors on the Dr. MJT's home-page, if he is still working in the field of computer architecture. Do not follow more than one local or global link from CSAGUPTA's home page, in search for Vidya's home-page. Similarly, do not follow more than one local or global link from vidya's home-page in search of her professor's home-page. Also get the name of her project.

```
select    rv.url, rv.text,
          ma.label, ma.href, ma.ltype
from      document v such that "http://144.16.65.83/~csagupta" =>|-> v
```

```
                                          and v.text like "vidya",
          relinfon  rv such that  delimiter = "hr"
          where     rv.text like  "database" ,

          document  m    such that  v =>|-> m  and  m.text like "MJT",
          anchor    ma
          where     m.text like "arch" ;
```

Query 8 Follow only local links from the Vidya's home page in the above example.

```
select    rv.url, rv.text,
          ma.label, ma.href, ma.ltype
from      document  v such that "http://144.16.65.83/~csagupta" =>|-> v
                                          and v.text like "vidya" ,
          relinfon  rv such that delimiter = "hr" ,
          anchor    av such that av.ltype = "l"
          where     rv.text like  "database" ,

          document m such that  av =>|-> m  and  m.text like "MJT",
          anchor   ma
          where    m.text like "arch" ;
```

4.6 Issues

With an aim to process a web-query in a distributed manner and on the existing documents on the web, we designed the Discover system. The features for such a distributed computing evolved a few issues which are uncommon in the conventional distributed DBMS. Before discussing these issues, we mention the generic reasons for these differences :

- The main reason is that all the servers in the Discover environment are independent and they are required to cooperate among themselves to make web-query's distributed computing feasible with less cost in terms of network bandwidth and time consumed.
- The feature to keep minimum overheads for a query-server.

- Apart from these, there exists no controller or coordinator to check the traversal of a web-query by virtue of its clones.

The possible solutions to all of the issues discussed below is subject to how much a query-server be overloaded in terms of its role and resources. The few issues with their possible and simple solutions are explained below :

1. *Query servers co-operating among themselves voluntarily* : In order to query the existing documents on the web in a distributed manner, it is essential that these web sites are equipped with a canonical query processing technique. Apart from simply processing a query sent for them, they should help user by efforts like cloning and forwarding the rest of his query. This is important because apart from some related information, the user may want to follow the hypertext organization from that document for more or complete information. If web-server provides only query services, then that is not of much use. As the user would have to download the document, apart from the result, in order to follow the hyperlinks. This requires a standard protocol among user and all web sites, where web sites perform a few overheads, on behalf of a user querying the web.

   In the Discover system, the role of a query server is to receive a web-query corresponding to some node maintained at its site, process the node-query of the web-query, send back the result to the user, and forward the rest of the query.

2. *Starting Points for a web-query* : The main drawback in the querying mechanism like our Discover system, WebSQL, WebLog or W3QS is that user has to determine from where the query processing should start. This problem still forces the new querying mechanism to depend on either user knowledge or search engines.

3. *Multiple visits at a node by same query* : Due to uncontrolled hypertext structure of the web, a web-query in the Discover system may visit a node multiple times in different states of computation. This could cause the same query to be computed multiple times at a node and in the gossamer that could be followed by the query in that state. We have addressed this problem and described the solution in Section 4.4. The solution requires query servers to maintain log records of the state of the query processed.

4. *When does a web-query get completely processed* :

To avoid overloading of a query server, in brief the role of it is to process, send the result, and forward a web-query. Due to such a strategy, the user never has any information about the number of the clones fanned out at any node. Though any Complete-Path of a web-query is finite, (refer Section 5.3), still the user cannot determine whether there exist any more clones of the query on the web, or not. Therefore a question arises that for how long should the user wait for the reply, as he can *never be sure* that he has received all the possible results from all the relevant nodes which could be visited by his web-query. As there can always be a clone of the query alive in some part of network waiting to be processed due to reasons like huge pending queue at that site, network delay, etc. As explained later, a simple solution to this problem could be :

(a) Approximate Query Solution

(b) Complete Query Solution

Approximate Query Solution : A user stops waiting for the result either after some time (say by killing her session) or feels that he has got sufficient result. However this approach is wrong due to the following reasons :

- All the clones left in the web causes at the visited nodes to perform unnecessary computations and further forwarding of the web-query. (In the Discover system, a Server-router node does not forward a query, if it can not successfully transfer the result. It assumes that the user does not exist any more.)
- The killing of the session by the user gives rise to one more problem. It may happen that new user starts his session and waits for the results on the same port number as used by the former user. Due to this the new user may receive the result from some node for the query made by the earlier user. This problem can be easily solved by a protocol that the user has to send his encrypted signature to Server-router before being provided with the result. Also this signature must be made available in the web-query object.

Complete Query Solution : We define *Complete Query Solution* (CQS) as the total of all results available at all the relevant nodes belonging to the set of nodes which could be visited by a query, such that no clones are left in the web (i.e. is completely answered). To obtain CQS, how long a user should wait can be answered using the following *CHT* protocol :

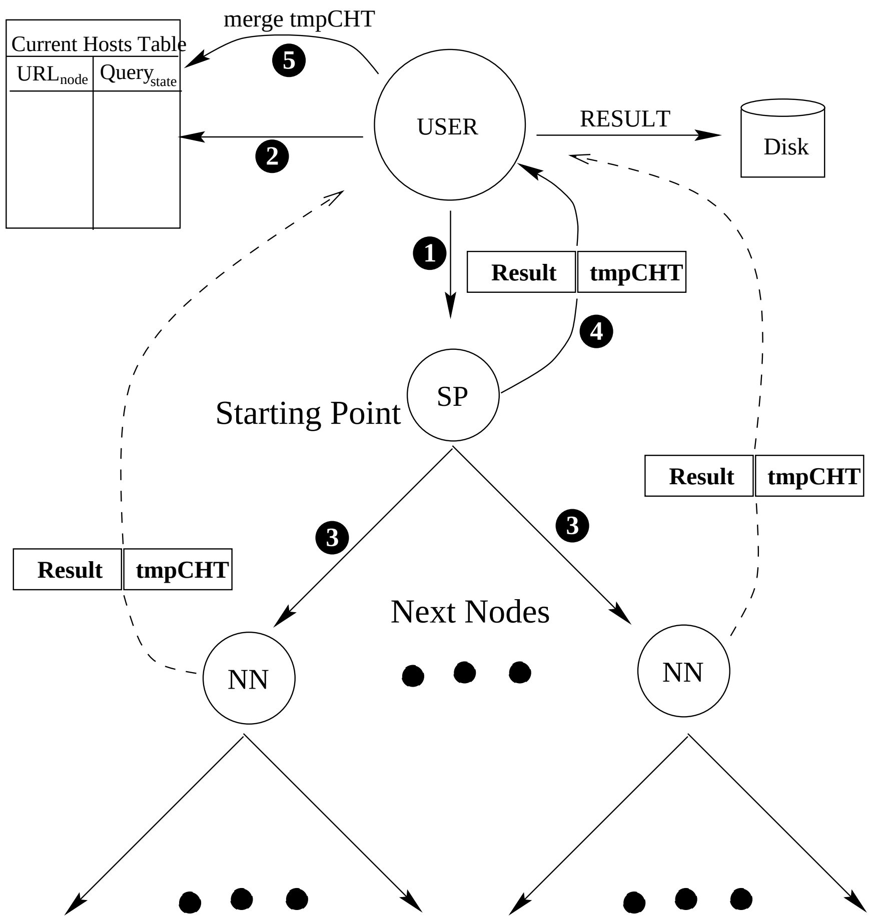

Figure 4.3: CHT protocol

A user maintains a local table called *Current Hosts Table* (CHT) to keep track of nodes currently hosting clones of his query. CHT has records(rows) of type (URL$_{node}$, Q$_{state}$), where URL$_{node}$: node currently hosting a clone; Q$_{state}$: the state of the clone at the node. After completely forwarding the query, the node supplies the list of next nodes to whom the query was transferred successfully.

To understand the protocol, consider the following example. For example let 'u' be a user, who sends a query to a starting point 'SP', which forwards the query to the set of next nodes like 'NN'. Figure 4.3 shows the flow of information among user and these nodes in conformance with the Protocol.

1. User 'U' sends query to 'SP'.

2. If successfully transferred, then it enters a record (URL$_{SP}$, Q$_{curr_state}$) in its CHT.

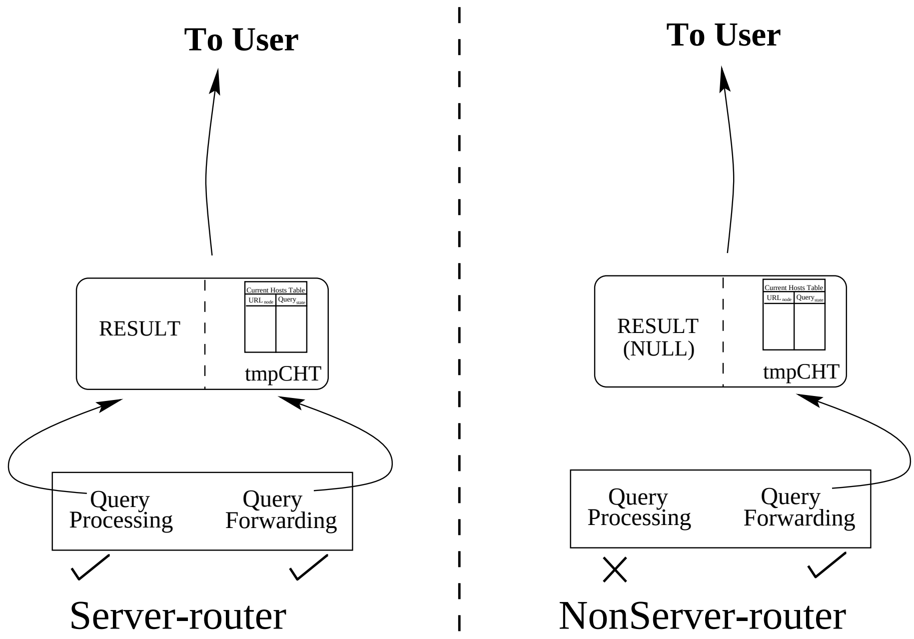

Figure 4.4: Information sent by Server-router and NonServer-router

3. 'SP' forwards query to the set of next nodes, like 'NN'. Simultaneously it prepares a 'tmpCHT' (temporary CHT) of the next nodes, to whom the query was successfully transferred, using (URL_{NN}, $\mathrm{Q}_{advanced\_state}$).

   After completing this, 'SP' inserts an entry (URL_{SP}, $\mathrm{Q}_{received\_state}$) at the top of tmpCHT, where URL_{SP} is its URL and $\mathrm{Q}_{received\_state}$ is the state in which it received the query.

4. 'SP' sends tmpCHT to the user.

5. It should be noted that user will always find one entry in its CHT which equals the first entry in the tmpCHT received. It removes that entry from its CHT along with the first entry in the tmpCHT. Then the user merges the tmpCHT with its CHT.

As node may have to send result and it knows all the next nodes to whom it has transferred the query, so its sending the tmpCHT along with the result will be cost effective in terms of network as compared to the alternative where every next node acknowledging the receipt of the query. In the Discover system, the NonServer-router will send a null-result along with tmpCHT, as shown in Figure 4.4. Clearly at any instant, user knows URLs of all the nodes, which are currently hosting the clone. So, whenever the user CHT becomes empty, it implies that the query has been completely answered by the web and the user can stop waiting for the result from this query.

4.7 Summary

The Discover system has been proposed, which could process a web-query using existing documents on the web in a distributed manner. The web has been viewed as a relational DBMS, by associating a few virtual relations to its resources. A user logically specifies a scheme to traverse a hypertext organization of the web and node-queries for a few intermediate nodes which may be visited by the web-query. A web-query traverses the web by virtue of the visited nodes voluntarily cloning the current state of the web-query and forwarding it to the next set of nodes. A simple query processing optimization is provided to avoid re-computation of the web-query, which re-visits a Server-router node in a same state more than once. Also, we have introduced the CHT protocol, which could help the user to keep track of his query and decide when the query has been completely answered by the web.

Chapter 5

Implementation Details

In the previous chapter we discussed the various conceptual issues that came across while designing the Discover system. In the following sections, we discuss some implementation issues involved in the development of the Discover system. A user specifies a web-query in WSQL using the user-interface. Then user-interface will create a web-query object which is clonable. By virtue of clones, the web-query object traverses the web and reaches the query-servers to be processed for some node residing on them. The three basic components of the Discover system are Web-Query Object, User-Interface and Query-Server. In the following sections we describe these components and their respective constituents.

5.1 Web-Query Object

The Web-Query Object used by the Discover system constitutes of the following two components, as shown in Figure 5.1 :

1. A QueryID,
2. A set of node-queries.

QueryID

In the Discover system, the QueryID object of a query carries the following information :

- *UserID* is the login-name of the user on his machine

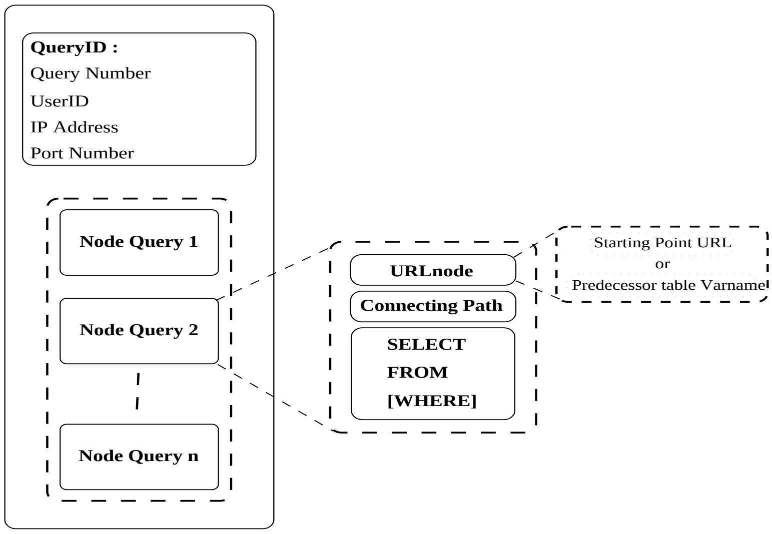

Figure 5.1: A Web-Query Object

- *IP-Address* is the Internet address of the user's machine
- *Port Number* is the port number at which user is waiting for results
- *Query Number* is the number given to each web-query submitted by the user

This information is used to send back the results directly to the user, to make a log entry at the query-server, to automatically collect all the results of a web-query in one file.

Node-Queries

A node-query is a part of web-query, which is used to make the content based search at a node. Whereas a set of node-queries are specified to explore the web.

- *URLnode* field of the top most node query (i.e. Node Query 1 in Figure 5.1) is the URL of the node corresponding to which the query reached that node. Whereas, for the rest of the lower node-queries, it is a table variable name declared in the node-query just above them as shown in the Figure 5.1. This table variable name must be associated to some "Anchor" virtual relational table.
- *Connecting-Path* field of a node-query, which maintains a PRE as a String. A query follows this PRE in order to reach a node, which would process this node-query.
- *SELECT* is a list of attributes of the tables declared for this node-query.

- *FROM* is a list of tables declaration for this node-query.
- *WHERE* is a tree of predicates for this node-query.

5.2 Client

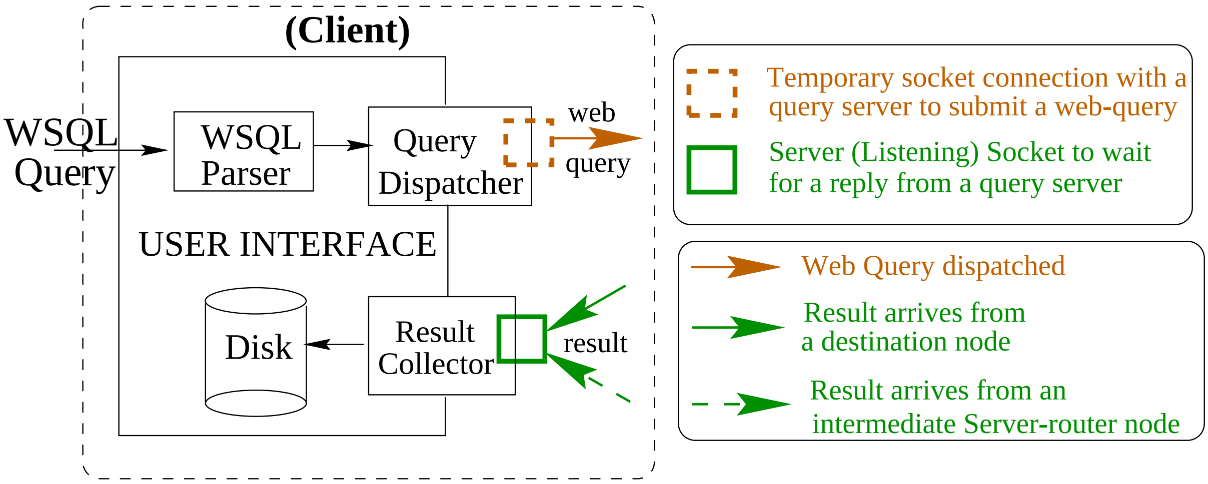

Figure 5.2: The Discover's User-Interface

In this section, we describe the user-interface used by a user to write a WSQL query, submit it to the web and wait for the results. A user can submit her query written in WSQL through one of the interfaces GUI, Command-Line and File. As shown in Figure 5.2, the user-interface used by a user constitutes of :

- *WSQL Parser* receives WSQL query from GUI/Command-Line/File, parses the query, converts the query into a web-query object and passes the web-query object to the Query Dispatcher.
- *Query Dispatcher* makes a socket connection with a starting-point for the web-query and dispatches it. Before dispatching, it initializes the QueryID object of the web-query.
- *Result Collector* opens a listening socket at some port number to receive results. The Result Collector always tries to open the listening socket at the port number used in the last execution session, if fails, it then chooses a random number. This port number is used in the QueryID object, while dispatching a web-query. It ignores the results, if they have

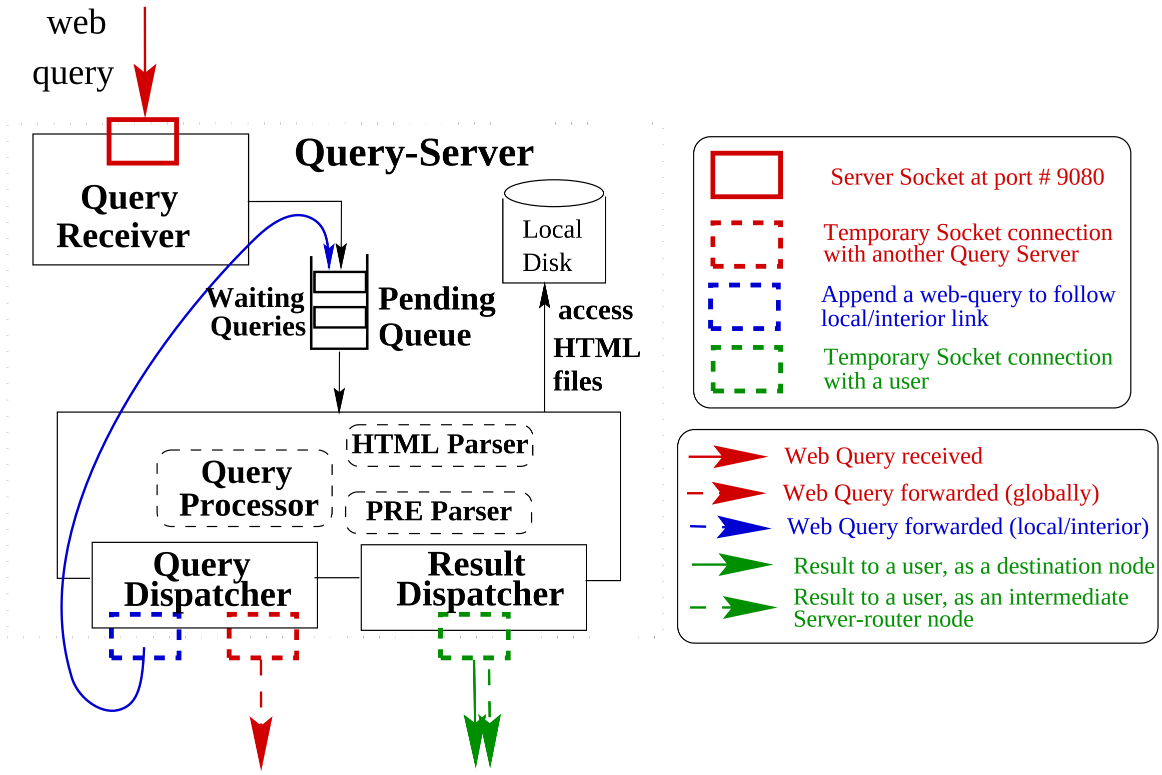

Figure 5.3: The Discover's Query Server

invalid UserID. This may happen if a user finishes her session and later some other user starts the session on the same machine and at same port number.

5.3 Server

The Discover system's Query Server, is proposed to be added over all web sites is shown in Figure 5.3. This model corresponds to a query server running as a daemon process at a web site (refer section 4.3). The constituents of this model are :

Query Receiver opens a server socket at the fixed port number to receives the web-queries. At present the port number used is 9080. Query Receiver runs as a thread of the Query-Server.

Pending Queue contains the web-queries waiting to be processed. Its a thread-safe common resource used by Query Receiver thread and Query-Processor thread (described below).

Query-Processor is a thread of the Query-Server, which processes a web-query. If the Pending Queue is non-empty, then the Query-Processor removes the first web-query from the queue and finds the URL from the top most node-query contained in this web-query. This URL corresponds to a node maintained at this site. The Query-Processor accesses the HTML file corresponding to this node (URL). Then it uses the *HTML parser* to parse the HTML file and construct a temporary database in the main-memory. If the node corresponds to a NonServer-router, then it constructs two special tables, *Super Document Table* (SDT) and *Super Anchor Table* (SAT), else the node corresponds to a Server-router. In that case, the query processor also constructs the other tables declared in this node-query. It checks whether the node satisfies the VTSpec condition specified in the Document table declaration of this node query or not. If the node does not satisfies then the Query-Processor makes a log entry and purges the query else it processes the node-query, makes a log entry and sends back the results to the user using *Result Dispatcher* . If no more node-queries exits in the web-query, then this clone of this web-query has seen a Complete-Path (i.e. reached a destination). Otherwise, the query server has to forward this web-query. According to the role played by the node, the query server forwards the web-query as follows :

If this node is a "NonServer-router", then a non-null Connecting-Path field exists in this node-query (refer Figure 5.1). The Query-Processor uses the Anchor relation table SAT and in accordance with the Connecting-Path, selects the hyperlinks from the SAT and forwards the clones of the web-query along each of these links. It should be noted that the Connecting-Path field in the top most node-query of the clone is modified to mark the link to be followed, as followed (more details, later in this section). (For example if the Connecting-Path is (`=>->=> | ->=>#>`), then before following a local link, the modified Connecting-Path for a clone would be (`->=>`). Whereas before following a global link, the modified Connecting-Path for a clone would be (`=>#>`).

If this node is a "Server-router", then it uses the Connecting-Path specified in the top most node-query of the web-query. In addition, which Anchor relation table of this node is to be used to forward the clones of the web-query, is specified in the "URLnode" field of the top most node-query in the web-query. The URLnode of the top most node-query in the web-query may contain the Document variable name corresponding to this node. In that case, SAT is used as the Anchor relation table. Once the Anchor relation table is selected, then in accordance with the Connecting-Path the multiple clones of the web-query are forwarded, as done for the NonServer-router case. But before forwarding the clone, the URLnode field of the top most node-query of this clone is replaced by the URL of a node to be reached.

Query Dispatcher is used by Query-Processor to forward the web-queries to another query-server corresponding to some node. Query Dispatcher is functionally same as that of in the User Interface.

Result Dispatcher is used to send back the results of a query to the user. QueryID object of the web-query supplies information of users' machine IP-Address and the port number on his machine at which he is expecting the results.

PRE-parser is used by a query-processor to resolve the PRE representing Connecting-Path into a set of PREs such that each have a form :

"⟨link-type⟩REMAINING-PRE".

Internally in a Query Object a PRE is stored as a combination of letters 'l','g','i' and 'n' corresponding to local(`=>`), global(`->`), interior(`#>`), and null(`@>`) links respectively. PRE-parser takes a PRE represented in any combination of these letters and symbols. In order to avoid runaway queries, we restrict the repetition symbol '*' used in connecting-path to zero or at the most ten repetitions, unlike zero or more repetitions used in [MMM96]. Also, a positive integer with value less than 10 can be appended to '*' to provide the maximum allowable repetitions limit. (For example (`=>*`) implies zero or at the most 10 consecutive local links to be followed, whereas (`=>*`3) implies zero or at the most 3 consecutive local links to be followed along a path.) So, the PRE used to specify connecting-path are always finite. The following are few examples of PRE and corresponding PRE-parser output.

```
        PRE                            PRE-parser OUTPUT
----------------------------------------------------------------------
        g                              g
----------------------------------------------------------------------
        g|l                            g
```

```
                                l
------------------------------------------------------------------------
        g*                      n
                                gg*9
------------------------------------------------------------------------
        g*5                     n
                                gg*4
------------------------------------------------------------------------
        (->|=>|#>)->->          ggg
                                lgg
                                igg
------------------------------------------------------------------------
        (gi*|ll*)*              n
                                gi*10(gi*10|ll*10)*9
                                ll*10(gi*10|ll*10)*9
------------------------------------------------------------------------
        (i*g|@>l*i)*             n
                                 ii*9g(i*10g|nl*10i)*9
                                 nl*10i(i*10g|nl*10i)*9
------------------------------------------------------------------------
        l*(i*g|@>l*i)*           n
                                 ll*9(i*10g|nl*10i)*10
------------------------------------------------------------------------
```

In order to forward a web-query, the query-processor does the following :

- obtains the set of PREs as the PRE-parser output for the Connecting-Path
- decides which Anchor relation table to use to forward a web-query, as explained earlier in the description of a Query-Processor in this section
- if the attribute 'ltype' of any tuple in this table matches to a prefix (i.e. first alphabet) of any PRE contained in the above set of PREs, then
  - a clone of the web-query is created

- the contents of the 'Connecting-Path' field in the top most node-query in this clone is replaced by the 'REMAINING-PRE' part of the matched PRE
- the URL of the link to be followed is obtained from the value of the 'base' and 'href' attribute for this tuple. The contents of the 'URLnode' field of the top most node-query in this clone are replaced by this URL
- the IP-Address of the next query server is obtained from this URL
- a connection is established with this next query server and query is dispatched to it.

This is done for all such matching (tuple, PRE) pairs. The BNF grammar specification for PRE parser is given in Appendix C.

5.4 Database construction

Unlike RDBMS, the virtual relational schema used by the Discover system has semantics associated with the tables of various types (Document, Anchor, Relinfon) while they are under construction. These semantics are specified by using VTSpec (refer Section 4.5.1). The use of object oriented language Java to implement the entire prototype of the Discover system lets us model each entity as an object of some class. So, we defined a class for each virtual relation used by the Discover system, inheriting a common base class. This lets us implement different semantics associated with them in the cover of common interfaces. All the tables, i.e the database, are constructed and maintained in the main memory for the following reasons :

1. The web-queries are read only.
2. Typically, HTML files are not huge.
3. It is temporary, as it lasts as long as the query is processed.

A *from-clause* in a node-query is used to obtain the declarations of all the virtual tables, along with their VTspecs required to be constructed. The approach developed to construct such a database in only one parse of an HTML file is analogous to a "Event" based system. In our system, the event is an HTML tag found while parsing the HTML file. The construction of the tables and inserting tuples in them has been explained by describing the components involved

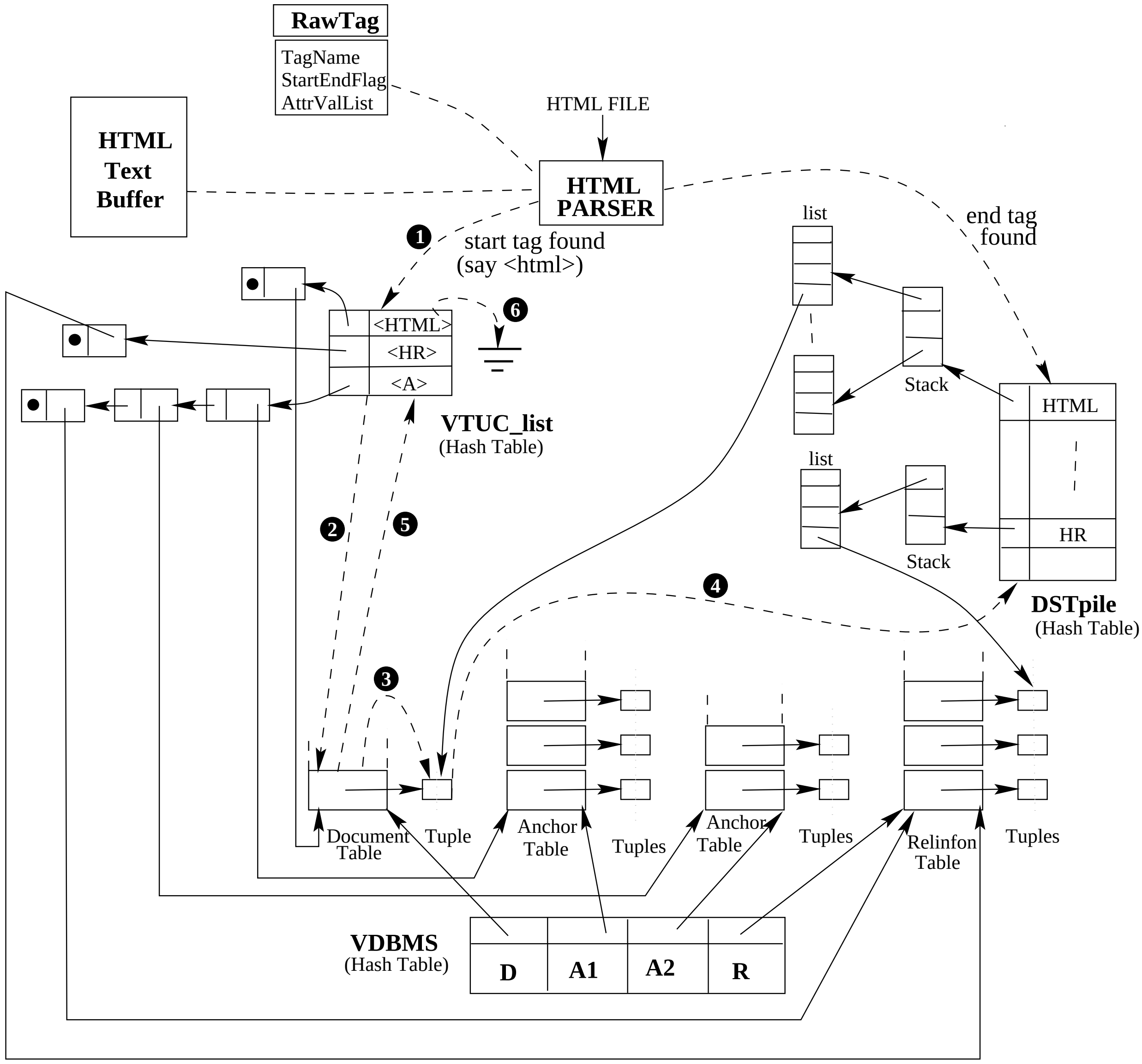

Figure 5.4: Web Database Under Construction

in this procedure and shown in Figure 5.4. Each component is an object of a class defined for it.

1. VDBMS

The entire database (i.e. all the tables) constructed is referred by a hash-table called *VDBMS* . It is informed by the query processor of all the table types to be constructed, along with their associated identifier and VTSpec. It instantiates these tables by supplying them with their respective VTspecs. It hashes their references into itself, using their respective identifiers as table name. Its main use is to supply references to these tables on the basis of the table name, For example while taking cartesian product of all the tables. Also, after the query-processor processes the query, it informs VDBMS to purge the database.

2. VTUC_list

The mechanism to insert tuple is based on *subscription* . Each relational table when instantiated, subscribes to a hash-table called *VTUC_list* , for being notified when a start tag is found in the HTML file. For these tags they have some action to be taken. It is an open hash-table, which maintains the references of all the tables, which are *notifiable* , (i.e. wants to be informed whenever a start tag is seen in an HTML file) keyed by the HTML tag string. Parsing of an HTML file, and actual flow of operations to fill tables is discussed later in this section. The table of type :

- Document always register for an ⟨HTML⟩ and ⟨TITLE⟩ tags only.
- Anchor always register for an ⟨Anchor⟩ tag only.
- Relinfon always register for a tag specified as a delimiter in its VTSpec, ⟨A⟩ and ⟨HTML⟩.

It should be noted that apart from the table of type Document, there could be multiple tables of other two types.

3. RawTag

A single *passive* object of this type, is shared among everybody. Whenever a tag is found in an HTML file, this resource is reset to the value associated with the tag. The tables use the information supplied to them through RawTag object, to add a new tuple in their list. The information set by an HTML parser in this object, when a tag is found is :

- tag name
- start or end tag (For example ⟨HTML⟩ or ⟨/HTML⟩ respectively
- Attribute and its value pairs for the tag

4. Virtual Relational Tables

There are three virtual relational classes :

1. Document
2. Anchor
3. Relinfon

The VDBMS object would instantiate these classes by supplying them with their VTSpecs. These tables subscribe themselves into VTUC_list. Table of type Document and Relinfon append an incomplete tuple to their list, which is to be completed when they see a particular tag. VTUC_list uses an interface "insertTuple" to inform tables that the tag they are waiting for has been found and RawTag has been reset to it. On being informed, these tables do not create a tuple and append to it but do as follow :

- Document type table would be informed only for HTML tag and then TITLE tag. It uses them to fill few values in its already existing tuple. It then subscribe to the DST-pile (explained later) for the corresponding end tag. It then unsubscribe itself from the VTUC_list. It should be noted that there is always one Document table with one tuple in it, corresponding to the node being queried.
- Anchor type tables always wait for an ⟨A⟩ tag. VTUC_list informs them that anchor tag has been found. Anchor type tables insert a tuple corresponding to it, only if the anchor tag seen in an HTML file has a 'href' specified.
- Relinfon type tables are informed for the tag they use as delimiter, HTML and Anchor tag. It uses delimiter to complete the last tuple inserted, and insert a new incomplete

tuple. It uses HTML tag to determine the offset of the string in HyperTxt at that state. Then Relinfon type table requests VTUC_list to unsubscribe it for the HTML tag. It uses anchor tag, if it contains attribute "name", then it uses this anchor tag to update the last URL field of the table. This way we associate each 'rel-infon' with a nearest spot in the file, that can be addressed as URL.

5. DSTpile

DSTpile is a hash table which contains references to the incomplete tuples waiting for some end tag to be found. All the tuples (any type) which are waiting for the same tag, are grouped together. But instead of hashing this list into DSTpile with the name of the tag they are waiting for as key, following is done :

As HTML is a growing language, we may expect nested tags, like

```
<Tag1 ... >
    - - -
    <Tag1 ... >
       - - -
    </Tag1>
</Tag1>
```

To handle them, we create a stack, hash it into DSTpile with the end tag's name as the key. On this stack we push the vector list of tuples waiting for the same end tag and at the same nesting level.

6. HTMLparser

This object is an event generator to fill the database. This object parses an HTML file and recognizes the following tokens [BLC95] and does the given action :

tag : Whenever it encounters a construct like "⟨ItIsStartTag ...⟩", it assumes that it has found a start tag. It resets RawTag object. Then it notifies VTUC_list that a start tag has been found, which further notifies the tables waiting for this tag. Similarly, for a construct like "⟨/ItIsEndTag ... ⟩", it notifies the DSTpile that an end tag has been found. It treats "tag" as an event.

word : any text word viewable on a common browser is accepted and appended to HTML_Text buffer.

punctuation : all punctuations found are appended to HTML_Text buffer.

entity : currently ignores entities.

comment : ignores it.

declaration : ignores it.

Note that the HTML parser written for the Discover is in no way a validator of an HTML file.

7. HyperText Buffer

It is a common string buffer and also provides a few methods like "substring" and relational operations. All viewable strings found in the HTML file are appended to it. It serves as a data type to attributes text and title of Document, label of Anchor and text of Relinfon. All these attributes are basically strings. It should be noted that length of text of Document could be large and values of all the attributes mentioned have string, which is a substring of it. Instead of creating separate strings for each of them, it is better to have one common buffer, and provide these attributes with the positions of the substrings corresponding to their values in this buffer.

8. Virtual Relational Tuples

There are three types of tuples : Document-Tuple, Anchor-Tuple, and Relinfon-Tuple. These are instantiated by their counterpart tables only. These are provided with information in terms of RawTag and HTML-Text Buffer to fill up themselves. A tuple may subscribe to DSTpile, to be notified for some particular end tag. Only in order to complete itself, particularly for an attribute of type Hyper-Txt, a tuple needs to subscribe to DSTpile.

We now briefly explain the order of actions as shown by dashed lines in Figure 5.4 to insert a tuple in the table.

1. HTML parser finds (say) an ⟨HTML⟩ tag. HTML parser resets the RawTag object. Then HTML parser informs VTUC_list about the tag found.

2. VTUC_list checks if any table has subscribed for $\langle$HTML$\rangle$ tag. It finds a Document table is waiting for this tag. It notifies the Document table by interface method "insertTuple" that a required tag is kept in RawTag.

3. The Document table creates an incomplete tuple and adds to itself.

4. The Document tuple created, gets itself subscribed to the DSTpile, in order to wait for end $\langle$/HTML$\rangle$ tag.

5. The Document table informs the VTUC_list that it is no more waiting for the $\langle$HTML$\rangle$ tag.

6. VTUC_list unsubscribes the Document table.

Once the HTML file is completely parsed, all the tables and their tuples are informed in order to take any default action, if required. For example if some tuple is still incomplete, it may use some default action to complete it.

5.5 Other details

All the parsers used (HTML, PRE and WSQL) were generated using *JavaCC* [JAVb] parser generator, whereas rest of the project has been implemented in Object-oriented programming language *Java* [JAVa]. Apart from the OO feature of the language Java, the other features of it exploited are portability, threaded programming, and serialization. Whereas, JavaCC provided us a combined tool to implement lexical analyzer and parser generator. The generated parser was in language Java and a top-down parser. The efforts involved in implementing a simple prototype was ~9000 lines of documented code.

5.6 Summary

In this chapter, we described the internal details of few major components of the Discover system and how a PRE is resolved to simpler PREs to follow links. Then, we discussed single parsing of the HTML file to construct the database temporarily in the memory.

Chapter 6

User Interface

The Discover system provides three user interfaces to submit a web-query written in WSQL. They are : 1) File, 2) Command-Line, and 3) GUI. The purpose of these various user interfaces is to submit :

1. web-queries in batch mode using file.
2. web-query either by a advanced user or from the output of other application.
3. web-query using graphical user interface, which hides most of the syntax details required to specify the WSQL query. It also automates most of the features.

All the three user interfaces read the input WSQL query as mentioned above and provide an input-stream to the WSQL parser. They may convert (For example GUI), the input to the WSQL query language. In the following sections, we describe how a user can use any of the above-mentioned interfaces.

6.1 File

A user writes his query in a file and provides the file name as an argument to the client-side program “Discover”. (Do not confuse this name with the entire system name i.e. the Discover system). For example if query is written in file “f1.wql”, then at the shell prompt, user gives a command :

```
%    java  Discover   f1.wsql
```

6.2 Command line

An advanced user may write his query online as follows :

```
%   java   Discover
select a.href, a.label
from   document  d  such that "http://www.csa.iisc.ernet.in/" @> d ,
       anchor    a ,
       relinfon  r such that r.delimiter = "hr"
       where     r.text like "sTuDenT" ;
```

or as an output of some other application :

```
%   cat  query.file | java   Discover
```

6.3 GUI

We have designed a simple and dynamic GUI to input WSQL queries. Most of its features are automatic, like

1. variable names for table declarations
2. select list
3. previous anchor tables list
4. variable size

The user starts the Discover's user interface program from the shell prompt as :

```
%   java   Discover   gui
```

The GUI comes up and its look and feel is as shown in Figure 6.1. It has a control panel consisting of :

1. Next-Node Button : to create required components in the GUI, to specify next node-query. For the first node query, the user specifies the starting point, whereas for the subsequent node-queries the anchor type table names in the preceeding node-query are automatically generated and provided to a user.

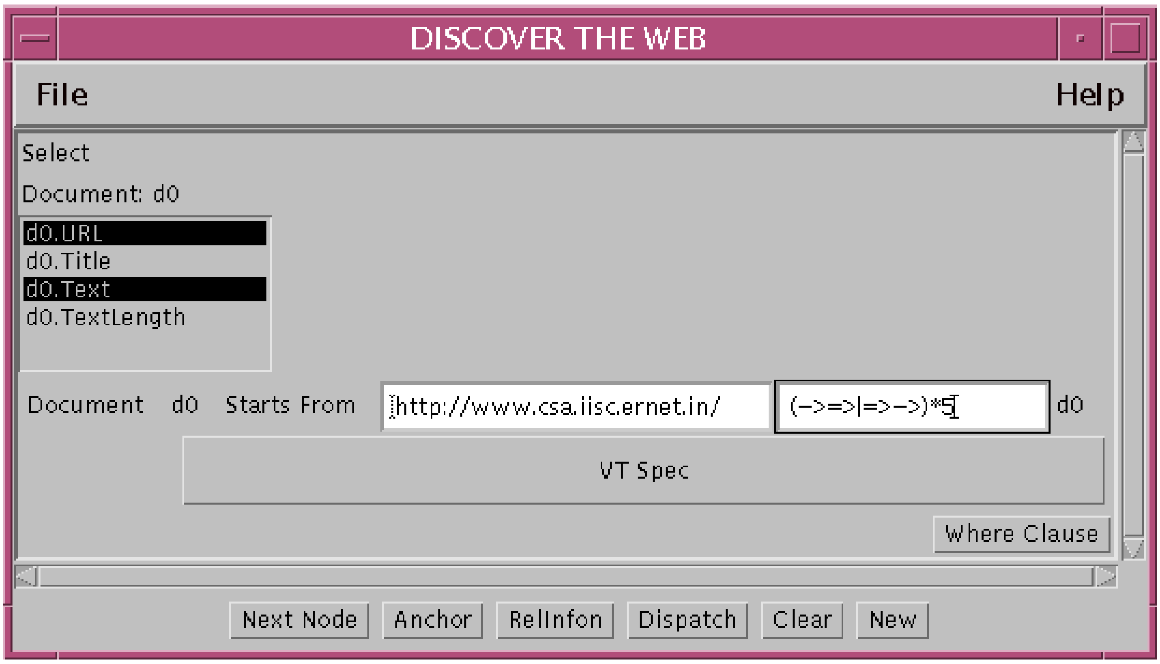

Figure 6.1: The GUI for the Discover

2. Anchor Button : to create an Anchor table declaration related components for the current node-query.

3. Relinfon Button : to create a Relinfon table declaration related components for the current node-query.

4. Dispatch Button : to submit the query to the web.

5. New Button : to clear and re-intialiatize the GUI for next web-query.

Apart from this, the file menu gives options of starting a clone of window or exiting from it. In Figure 6.1, it shows a declaration of only one document table named "d0" and asks for the starting point and the PRE . For the select-clause the list of all the tables declared are arranged horizontally, with the attributes they support. A user can make multiple selects of the attributes of a table.

A user can declare multiple number of Anchor and Relinfon tables in any order for this node, by clicking the mouse over the Anchor and the Relinfon buttons respectively. Multiple Anchor and Relinfon tables obtained for the first node-query is shown in Figure 6.2. This figure shows two anchor tables automatically named "d0_A0" and "d0_A1", plus a Relinfon named "d0_R0". Note that each has a prefix "d0", corresponding to the node-query they belong. By

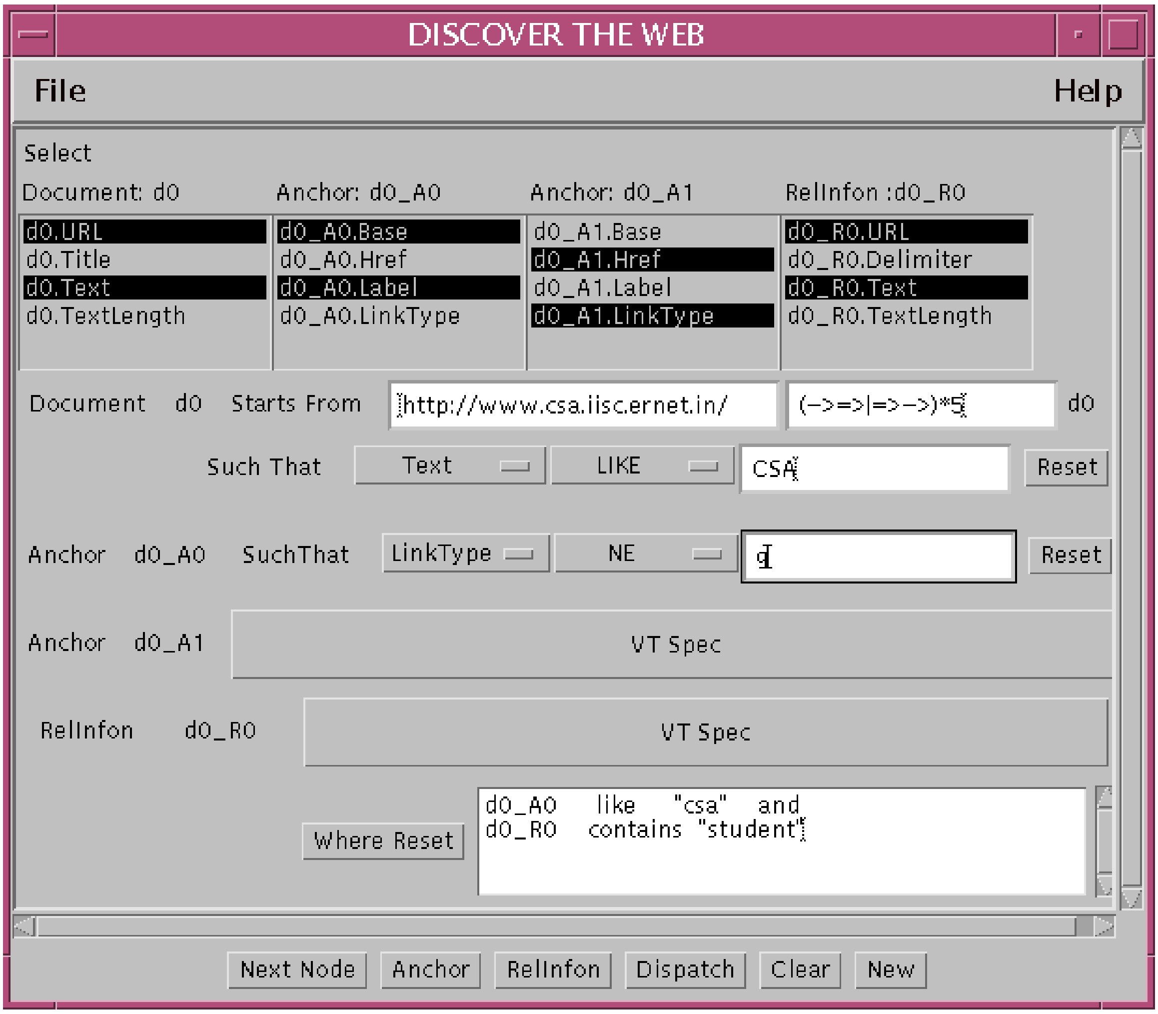

Figure 6.2: GUI with a single node-query

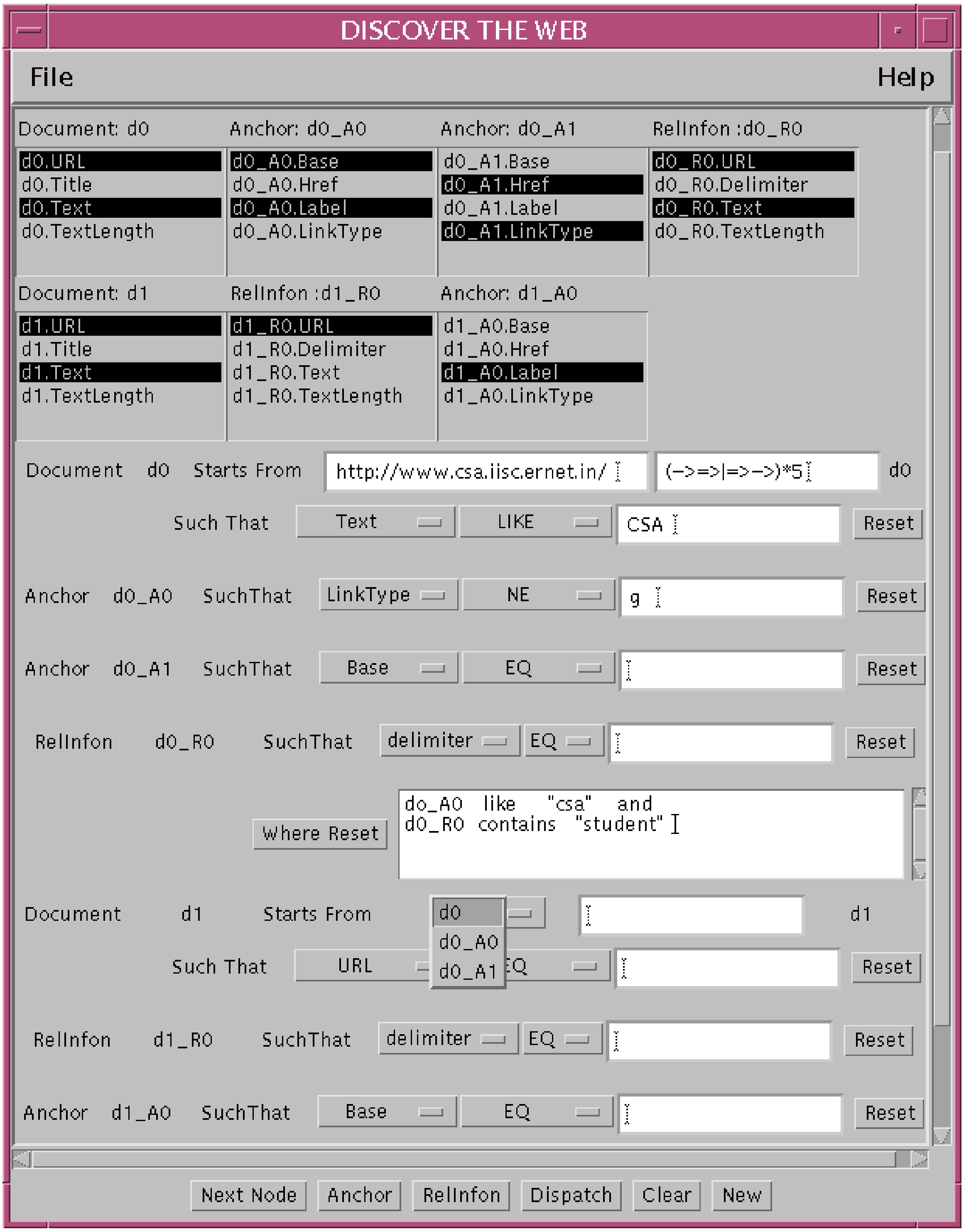

Figure 6.3: GUI with multiple node-queries

clicking on "Where-Clause" shown in Figure 6.1, we get a region as shown in Figure 6.2 to key in *predicates* for the where-clause of the current node-query. Also, note the horizontal addition of list corresponding to table d0_A0, d0_A1 and d0_R1, in the select-clause (list). A user can select multiple attributes of these which he wants back as result for the query.

In order to specify the next node-query, click the "Next-Node" button on the control panel. This action causes other node-query fields to be created and added to the GUI, as shown in Figure 6.3. The name of the Document table is "d1" and so on. Like for "d0", a user can declare any number of Anchor tables and Relinfon tables. It should be noted that the name given to this node is "d1" and it is used as a prefix for other table names. Also, note the options in VTSpec of "d1" – it provides the list of possible anchor tables declared in previous node-query (i.e. for "d0"). The user specify one of these anchor tables, which is used as set of hyperlinks at nodes corresponding to "d0" to forward the query.

6.4 Current Implementation Notes

The current implementation of the user interface of the Discover system supports submitting web-queries through *Command line* and a single query at a time through a *File*. The *GUI* is completely ready but yet to be hooked to the Query-Dispatcher.

Chapter 7

Sample Query Execution

In this chapter, we present the contents of the various tables that are constructed at various nodes, while processing a web-query. To demo this, we arranged a set of HTML files on our campus machines, midas in DSL, krithika in SERC and osiris in CSA. The query processor was started as a daemon process on all of them. The following query was submitted from the machine malhar at DSL :

```
select    rv.url, rv.text,
            ma.label, ma.href, ma.ltype
from      document v such that "http://144.16.65.83/~csagupta" =>|-> v
                                                 and v.text like "vidya",
          relinfon  rv such that  delimiter = "hr"
          where     rv.text like  "database" ,

          document  m    such that  v =>|-> m  and  m.text like "MJT",
          anchor    ma
          where     m.text like "arch" ;
```

This query is an example query mentioned in Section 4.5.4. The purpose of this query is to start from CSAGUPTA's home-page on krithika to find his class-mate "vidya", who did her project on Database Mining and took system programming course under Dr. MJT. Find all the anchors on the Dr. MJT's home-page, if he is still working in the field of computer architecture. Do not follow more than one local or global link from CSAGUPTA's home page, in search for Vidya's

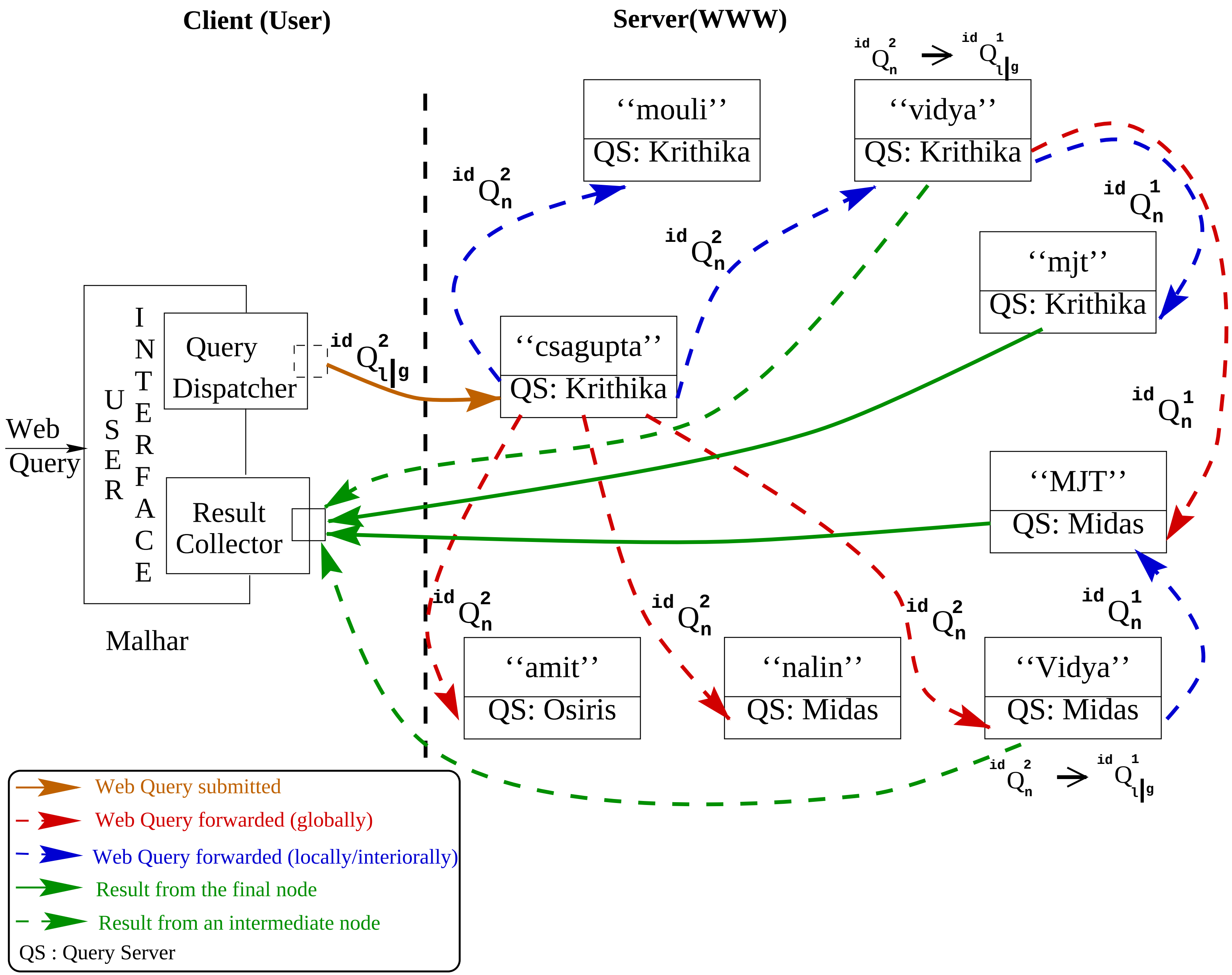

Figure 7.1: Web Traversal of the Test Query

URL	TITLE	TextLength
http://144.16.65.83/~csagupta	Nalin	294

Table 7.1: Super Document Relation : (SDR Document Table)

Href	Type	Label
http://144.16.65.83/~mouli/	l	Mouli @ KRITHIKA
http://144.16.65.83/~amit/	l	amit @ Osiris
http://144.16.65.83/~csagupta/vidya/vidya.html	l	Mining@krithika
http://dsl.serc.iisc.ernet.in/~nalin/vidya/VIDYA.html	g	Delta@dsl

Table 7.2: SAR Anchor Table (base = http: //144.16.65.83/~csagupta)

home-page. Similarly, do not follow more than one local or global link from vidya's home-page in search of her professor's home-page. Also get the name of her project.

The web traversal of the query is shown in Figure 7.1. The following sections show the contents of the tables contructed at the starting point (NonServer-router), a Server-router, and a destination. Also, the results sent by the Server-router and the destination are shown. Finally, we present the complete result obtained by the user in the last section.

7.1 At NonServer-router

According to the top most node-query, the Starting Node is :

"http://144.16.65.83/~csagupta"

on "krithika" (144.16.65.83). This node acts as a NonServer-router and constructs "SDR" (Table 7.1) and "SAR" (Table 7.2) tables. It uses "SAR" anchor table to forward the query.

7.2 At Server Router

As shown in Figure 7.1, and also in "SAR" Table 7.2, one of the next nodes the web-query reaches is :

"http://144.16.65.83/~csagupta/vidya/vidya.html".

Since PRE in the top most node query is totally consumed, this node acts as Server-router.

Therefore it processes the top most node query. Apart from "SDR" and "SAR" tables, the other tables constructed are 'v' as Document table (Table 7.5), 'rv' as Relinfon table (Table 7.6) with delimter 'hr'. Note from "SDR" or "v" table that this node satify the VTSpec condition. Table 7.7 shows the result it computed and sent back for the node-query.

7.3 At Destination

Finally, the query reaches the destination node :

http://144.16.79.150/~nalin/mjt.html .

It receives a single node-query in the web-query. Apart from "SDR" and "SAR" tables, the other tables constructed are 'm' as Document table (Table 7.10), 'ma' as Anchor table (Table 7.11). Note from "SDR" or "m" table that this node satify the VTSpec condition. Table 7.12 shows the result it computed for the node-query, and sends back. Since no more node-queries are left in the web-query, this node does not forwards the web-query.

7.4 Results Received

Figure 7.1 shows that there are two Servers-routers, "vidya at Krithika" and "Vidya at Midas", which send the result. The user gets null Result from "Vidya at Midas", because this node satisfies the VTSpec condition but does not contain any keyword like "database". The user also receives result from two destinations which has a word similar to "MJT" in their title. As shown in Figure 7.1 the query reaches the destination "MJT at Midas" twice, so the user receives the same result two times. It should be noted that such recomputation would not occur, if we maintain the log-records at the query-servers as described in Section 4.4. In the current prototype of the Discover system, the log-records maintenance has not been implemented. In end of the chapter, we included the result obtained by the user at malhar. The results are saved as text from the Netscape Browser, as the obtained results are embedded with HTML tags.

URL	TITLE	TextLength
http://144.16.65.83/~csagupta/vidya/vidya.html	vidya at krithika	266

Table 7.3: SDR Document Table

Href	Type	Label
http://midas.serc.iisc.ernet.in/~nalin	g	NALIN GUPTA
http://144.16.79.150/~nalin/mjt.html	g	mjt@midas
http://144.16.85.83/~csagupta/MJT.html	g	MJT@Krithika

Table 7.4: SAR Anchor Table (base = http://144.16.65.83/ ~csagupta/vidya/vidya.html)

URL	TITLE	TextLength
http://144.16.65.83/~csagupta/vidya/vidya.html	vidya at krithika	266

Table 7.5: ‘v’ Document Table)

text	textlength
vidya at krithika Shrividya’s TEST HOME PAGE FOR THE DISCOVER SYSTEM NALIN GUPTA	88
COURSES DONE COURSE INSTRUCTOR SYSPRO Dr. MJT mjt@midas and MJT@Krithika DBMS Dr. JRH	109
Project : Database Mining Tool (MINTO) Guide : Dr. JRH	56
Email Me	14

Table 7.6: ‘rv’ Relinfon table. (url = http://144.16.65.83/ ~csagupta/vidya/vidya.htm)

rv.url	rv.text
 http://144.16.65.83/ ~csagupta/vidya/vidya.html 	 Project : Database Mining Tool (MINTO) Guide : Dr. JRH

Table 7.7: First Node query result sent by URL: http://144.16.65.83/ ~csagupta/vidya/vidya.html

URL	TITLE	TextLength
http://144.16.79.150/~nalin/mjt.html	Dr. MJT's Home Page @ DSL	533

Table 7.8: SDR Table

Href	Type	Label
../acads/res-groups/ compsys.html/#perf-eval	1	Performance aspects of Computer Architecture
/homepage/acads/res-groups/ comp_arch/homepage.html	1	Computer Architecture
/homepage/personnel/mjt.html	1	click here

Table 7.9: SAR Table (Base : http://144.16.79.150/~nalin/mjt.html)

URL	TITLE	TextLength
http://144.16.79.150/~nalin/mjt.html	Dr. MJT's Home Page @ DSL	533

Table 7.10: 'm' Document Table

Href	Type	Label
../acads/res-groups/ compsys.html/#perf-eval	1	Performance aspects of Computer Architecture
/homepage/acads/res-groups/ comp_arch/homepage.html	1	Computer Architecture
/homepage/personnel/mjt.html	1	click here

Table 7.11: ma Anchor Table (Base : http://144.16.79.150/~nalin/mjt.html)

ma.label	ma.href	ma.ltype
Performance aspects of Computer Architecture 	../acads/res-groups/ compsys.html/#perf-eval	l
Computer Architecture	/homepage/acads/res-groups/ comp_arch/homepage.html	l
click here. 	/homepage/personnel/mjt.html	l

Table 7.12: The last Node query result from URL: http://144.16.79.150/~nalin/mjt.html

```
The Results Obtained By The User At Malhar For The Sample Query

----------------------------------------------------------------------------
From Node : http://144.16.65.83/~csagupta/vidya/vidya.html
For Query ID : Query ID : UserName :nalin QueryNumber :0
PortNumber(receiving) :18643
Result :
http://144.16.65.83/~csagupta/vidya/vidya.html Project : Database Mining
                                               Tool (MINTO) Guide : Dr. JRH
----------------------------------------------------------------------------
From Node : http://dsl.serc.iisc.ernet.in/~nalin/vidya/VIDYA.html
For Query ID : Query ID : UserName :nalin QueryNumber :0
PortNumber(receiving) :18643
Result :
----------------------------------------------------------------------------
From Node : http://144.16.79.150/~nalin/mjt.html
For Query ID : Query ID : UserName :nalin QueryNumber :0
PortNumber(receiving) :18643
Result :
 Performance aspects of
 Computer Architecture  ../acads/res-groups/compsys.html/#perf-eval        1
 Computer Architecture  /homepage/acads/res-groups/comp_arch/homepage.html 1
 click here.            /homepage/personnel/mjt.html                       1
----------------------------------------------------------------------------
From Node : http://dsl.serc.iisc.ernet.in/~nalin/mjt.html
For Query ID : Query ID : UserName :nalin QueryNumber :0
PortNumber(receiving) :18643
Result :
 Performance aspects of
 Computer Architecture  ../acads/res-groups/compsys.html/#perf-eval        1
 Computer Architecture  /homepage/acads/res-groups/comp_arch/homepage.html 1
 click here.            /homepage/personnel/mjt.html                       1
----------------------------------------------------------------------------
From Node : http://144.16.79.150/~nalin/mjt.html
For Query ID : Query ID : UserName :nalin QueryNumber :0
PortNumber(receiving) :18643
Result :
 Performance aspects of
 Computer Architecture  ../acads/res-groups/compsys.html/#perf-eval        1
 Computer Architecture  /homepage/acads/res-groups/comp_arch/homepage.html 1
 click here.            /homepage/personnel/mjt.html                       1
----------------------------------------------------------------------------
```

Chapter 8

Conclusion and Future Work

8.1 Summary

In this thesis, we presented the design and the prototype implementation of the Discover system. The Discover system is an distributed architecture with the purpose to perform web-queries and extract existing web resources (HTML files). It proposes a querying mechanism for the web using the SQL-like declarative query language WSQL. The Discover system inherits some basic feature from WebSQL [MMM96] and provides a new concept of processing web-query. The style of query processing can be viewed as the multi-threaded programming in the distributed environment of independent query-servers. To achieve this, we devised web traversal mechanism for the web-query by virtue of PRE and cloning at each node, with the notion of voluntary co-operation among the user and the query servers. We encountered new issues that typically do not arise in Distributed DBMS. We have addressed them by providing simple solutions. We suggested maintaining log-records of Query-State of the web-queries processed to avoid multiple computations of the same query at a node and in the following gossamer of the web. We also provided CHT protocol, so that a user can know when his query gets completely answered. Unlike WebSQL, it avoids following irrelevant documents by the concept of VTSpec and dead-end along a path of traversal. The temporary database at the node is constructed by a single parse of the HTML file. The entire Discover system has been implemented in platform independent O-O language Java, and related parsers have been generated using JavaCC. We exploit the feature of Serialization in Java for creating clones and transferring them across the web. Consequently, the Discover system is a new approach to search and retrieve from the web, which also implements features left as future work in WebSQL like query optimization, avoidance of irrelevant documents and constructing anchor table on the basis of VTSpec. Like the other related works, the purpose of this project is not to provide immediate solution to the naive users but as an engine to applications or programmers. It should be noted that conventional SQL is also not used by end-users but through some interface or by programmers. Applications or programmers can use the Discover system for :

- Selective Indexing based on the hypertext organization. Applications like Search engines, can generate the web-queries and use the Discover system to build their indices, rather than itself downloading the entire files and parsing them. They can restrict the query to either physical or logical hypertext organization.

- Link maintenance to determine that information referred by the links still exists, i.e. keeping links current and checking that they are not floating.

8.2 Future Works

The Web Database as well as the Discover system are emerging concepts, which leaves lot of scope in introducing new ideas and improvements. For the Discover system, in particular the query language WSQL requires lots of features to be added, For example aggregate operations. The current WSQL parser has a place holder through which it can push the WSQL queries into the stack, and then query the web. For aggregate operations such as **max**, the appropriate syntax and semantic need to be added in the select-clause. Currently by virtue of PRE, the web-query is restricted to logical hypertext organization, but by introducing a field in the Web Query object, we can restrict it with respect to physical hypertext organization. The aim of the Discover system is to keep minimum overhead over the query-servers. It would be interesting if techniques better than CHT protocol and log records could be devised. Also, the Discover system could be extended to perform more varieties of structural queries such as to query within any hypertext organization to determine how many hyperlinks point to a particular node.

Appendix A

Grammar for WSQL Query Language

The following is the BNF specification for the WSQL query language, generated using JavaCC document generator. Each Production has been given serial number, which is used in the thesis for explaination.

```
     NON-TERMINALS

0)   QueryReader              ::=  WsqlStatement <SEMICOLON>

1)   WsqlStatement            ::=  SelectClause FromClause

2)   SelectClause             ::=  <SELECT> Attrlist

3)   Attrlist                 ::=  Attr_VarFieldPair
                                 (<COMMA> Attr_VarFieldPair)*

4)   Attr_VarFieldPair        ::=  TableVar <DOT> Field

5)   Attr                     ::=  Field
```

```
6)  Field                     ::=  <IDENTIFIER> (<DOT> Field)?

7)  Table                     ::=  <IDENTIFIER>

8)  TableVar                  ::=  <IDENTIFIER>

9)  Node                      ::=  StrConst | Table

10) FromClause                ::=  <FROM> DomainSpec

11) DomainSpec                ::=  StartingNodeSpec
                                   (<COMMA> OtherVirtualRelationSpec)*
                                   (WhereClause)?
                                   (<COMMA> NextNodeSpec)*

12) StartingNodeSpec          ::=  <DOCUMENT> TableVar <SUCHTHAT>
                                   URLStringConst PathRegExpr TableVar
                                   (<AND> AttrPredicate)?

13) OtherVirtualRelationSpec::=  Table TableVar
                                   (<SUCHTHAT> AttrPredicate)?

14) NextNodeSpec              ::=  NextDocumentSpec
                                   (
                                     (<COMMA> OtherVirtualRelationSpec)*
                                     (WhereClause)?
                                   |
                                      WhereClause
                                   )

15) NextDocumentSpec          ::=  <DOCUMENT> TableVar <SUCHTHAT>
```

```
                                    Table PathRegExpr TableVar
                                    (<AND> AttrPredicate)?

16) PathRegExpr               ::=  PathRegExprTerm
                                   (<PIPE> PathRegExprTerm)*

17) PathRegExprTerm           ::=  PathRegExprFactor
                                   ((<DOT>)? PathRegExprFactor)*

18) PathRegExprFactor         ::=  (Link | (<LPAREN> PathRegExpr <RPAREN>))
                                   (<ASTERIK> (<INTEGER_LITERAL>)?)?

19) Link                      ::=  ("=" | "=>" | "#>" | "->")

20) WhereClause               ::=  <WHERE> SearchConditions

21) SearchConditions          ::=  BoolTerm (<OR> BoolTerm)*

22) BoolTerm                  ::=  BoolFactor (<AND> BoolFactor)*

23) BoolFactor                ::=  Predicate
                                |  (<LPAREN> SearchConditions <RPAREN>)

24) Predicate                 ::=  AttrPredicate

25) AttrPredicate             ::=  Attr
                                   (
                                      LikePredicateRHS
                                   |
                                      ComparisionPredicateRHS
                                   )
```

```
26) ComparisionPredicateRHS ::=  RelOpr
                                 (Attr | <INTEGER_LITERAL> | StrConst)

27) LikePredicateRHS        ::=  (<LIKE> | <CONTAINS>) StrConst

28) URLStringConst          ::=  StrConst

29) Constant                ::=  StrConst
                              |   IntegerConst

30) StrConst                ::=  <STRING_LITERAL>

31) IntegerConst            ::=  <INTEGER_LITERAL>

32) RelOpr                  ::=  <EQ> | <GT> | <LT> | <GE> | <LE> | <NE>
```

Appendix B

Grammar for HTML

The HTML Grammar written is based on the assumption that "lookup" tables are used. It is used to fill the database from a single parse of an HTML file. For that, the HTML parser acts as a reactive object, which generates event on finding either start or end tag.

```
--------------------------------------------------------------------------
HTMLDocument  :=  ( Tag | Decl | CommentTag | <Word> |
                      <Entity> |<Punct> | <Space>
                  )* <EOF>
--------------------------------------------------------------------------
Tag           :=  ( <TagName>
                      ( <ArgName> ( <ArgEquals> ArgValue )? )*
                    <TagEnd>
                  )
--------------------------------------------------------------------------
ArgValue      :=  <ArgName>
               |  <ArgQuote1> <Quote1Text> <CloseQuote1>
               |  <ArgQuote2> <Quote2Text> <CloseQuote2>
--------------------------------------------------------------------------
Decl          :=  <DeclName> ( ArgValue | <ArgEquals> )* <TagEnd>
--------------------------------------------------------------------------
CommentTag    :=  <Comment>
                      ( <CommentText> | <CommentExtra> | <CommentSpace>
                  )* <CommentEnd>
--------------------------------------------------------------------------
```

Appendix C

Grammar for PRE

PREparser : Path Regular Expression Parser. PREparser is used by a query-server to construct binary tree of the path regular expression received as string. The tree constructed is used to obtain set of modified PREs which will be of form : some ”linktype” followed by modified “PRE”. This will help query-server to consume the linktype and select a set of anchors of type same as linktype. Then query server can use these anchors to forward query with the corresponding modified ”PRE”

```
-----------------------------------------------------------------------
PathRegExpr          :=  PathRegExprTerm  (<PIPE> PathRegExprTerm)*
-----------------------------------------------------------------------
PathRegExprTerm      :=  PathRegExprFactor ((<DOT>)? PathRegExprFactor)*
-----------------------------------------------------------------------
PathRegExprFactor    :=  ( Link | (<LPAREN> (PathRegExpr) <RPAREN>))
                         ( <ASTERIK> (<CONST>)? )?
-----------------------------------------------------------------------
Link                 :=    <LOCALLINKSYM>
                      |    <NULLLINKSYM>
                      |    <INTERIORLINKSYM>
                      |    <GLOBALLINKSYM>
-----------------------------------------------------------------------
```

Bibliography

[BL93] T. Berners-Lee. Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Networkas used in the World-Wide Web. *RFC1630*, March 1993. Available at `ftp://ds.internic.net/rfc/rfc1630.txt`.

[BLC95] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. *RFC1860*, November 1995. Available at `ftp://ds.internic.net/ rfc/rfc1860.txt`.

[BLCG92] T. J. Berners-Lee, Robert Cailliau, and Jean-Francois Groff. The World Wide Web. *Computer Networks and ISDN Systems*, 25:454–459, 1992.

[BLCGP92] T. J. Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. World Wide Web: The Information Universe. *Electronic Networking: Research, Application and Policy*, 1(2):52–58, Spring 1992.

[BLFN96] T. Berners-Lee, R. Fielding, and H. Nielsen. Hypertext Transfer Protocol – HTTP/1.0. *RFC1945*, May 1996. Available at `ftp://ds.internic.net/ rfc/rfc1945.txt`.

[BLMM94] T. Berners-Lee, L. Masinster, and M. McCahill. Uniform Resource Locators. *RFC1738*, December 1994. `ftp://ds.internic.net/rfc/rfc1738.txt`.

[Bou] T. Boutell. World Wide Web FAQ. Web Page. Available at `http://www.boutell.com/faq/`.

[CGI95] "The Common Gateway Interface". University of Illinois, Urbana-Champaign, 1995. `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`.

[Con97] World Wide Web Consortium. HTML 3.2 Reference Specification. Web Page, January 1997. World Wide Web Consortium Recommendation. Available at http ://www.w3.org/pub/WWW/TR/REC-html32.html.

[D2W95] "DB2 WWW connection home page". http://service.software.ibm.com /pbin-usa-demos /getobj.pl?/demos-pdocs/wwwdb2dnld.html, 1995.

[FGM+97] R.T. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. *RFC2068*, January 1997. Available at ftp:// ds.internic.net/rfc/rfc2068.txt.

[Fie95] R. Fielding. Relative Uniform Resource Locators. *RFC1808*, June 1995. Available at ftp://ds.internic.net/rfc/rfc1808.txt.

[Fos94] J. Foster. A Status Report on Networked Information Retrieval: Tools and Groups. *RFC1689*, August 1994. Available at ftp://ds.internic.net/ rfc/rfc1689.txt.

[Gaf96] John Gaffney. Illustra´s Web DataBlade Module. *SIGMOD Record*, 25(1):105--112, March 1996.

[JAVa] Sun Microsystems. the Java Language : Programming for the Internet. http://java.sun.com.

[JAVb] Java Compiler Compiler, (JavaCC) Version 0.6(Beta). http://www.suntest.com /JavaCCBeta.

[JDB] The JDBC Database Acess API. http://splash.javasoft.com/jdbc/.

[Kos] Martijn Koster. The Web Robots Page:WWW Wanderers, Spiders, Robots, Worms. http:// www.webcrawler.com/mak/projects/robots/ robots.html.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World Wide Web. In *Proc. of VLDB'95*, pages 54--65, 1995.

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proc.*

of 6th. International Workshop on Reserach Issues in data Engineering, Ride'96. New Orleans, February 1996.

[MMM96] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proc. PDIS'96*, pages 80--91, December 1996.

[NS96] Tam Nguyen and V. Srinivasan. Accessing Relational Databases from the World Wide Web. In *Proc. of ACM SIGMOD'96*, volume 25, pages 529--540, June 1996.

[Per95] C. Pero. "HTML FORMS Tutorial". *University of Illinois, Urbana-Champaign*, 1995. http://robot0.ge.uiuc.edu/carlosp/cs317/cft.html.

[Wor] World Wide Web Consortium. Web Page. Available at http://www.w3.org/.