View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

1999

Harp: A distributed query system for legacy public libraries and structured databases

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Ying LU

DOI: https://doi.org/10.1145/314516.314521

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Databases and Information Systems Commons, and the Numerical Analysis and Scientific Computing Commons

Citation

LIM, Ee Peng and LU, Ying. Harp: A distributed query system for legacy public libraries and structured databases. (1999). *ACM Transactions on Information Systems*. 17, (3), 294-319. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/133

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Harp: A Distributed Query System for Legacy Public Libraries and Structured Databases

EE-PENG LIM
Nanyang Technological University
and
YING LU
Kent Ridge Digital Labs

The main purpose of a digital library is to facilitate users easy access to enormous amount of globally networked information. Typically, this information includes preexisting public library catalog data, digitized document collections, and other databases. In this article, we describe the distributed query system of a digital library prototype system known as HARP. In the HARP project, we have designed and implemented a distributed query processor and its query front-end to support integrated queries to preexisting public library catalogs and structured databases. This article describes our experiences in the design of an extended Sequel (SQL) query language known as HarpSQL. It also presents the design and implementation of the distributed query system. Our experience in distributed query processor and user interface design and development will be highlighted. We believe that our prototyping effort will provide useful lessons to the development of a complete digital library infrastructure.

Categories and Subject Descriptors: H.3 [**Information Systems**]: Information Storage and Retrieval; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms: Design, Languages

Additional Key Words and Phrases: Digital libraries, Internet databases, interoperable databases

Authors' addresses: E.-P. Lim, Center for Advanced Information Systems, School of Applied Science, Nanyang Technological University, N4-2A-12, Nanyang Avenue, 639798, Singapore; email: aseplim@ntu.edu.sg; Y. Lu, Kent Ridge Digital Labs, Heng Mui Keng Terrace, Kent Ridge, 119597, Singapore; email: luying@krdl.org.sg.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1999 ACM 1046-8188/99/0700–0294 $5.00

1. INTRODUCTION

1.1 Objectives and Motivation

In digital library research, one of the main objectives is to provide an integrated query service to digital library applications and users so they are able to remotely access legacy library collections as well as other archival information on the Internet. Typically, the kinds of information include:

—*Bibliographic data:* Bibliographic data can be found in every legacy public library. These data, also known as library catalog data, provide the references, e.g., stack numbers, to the shelves on which the library books or materials can be found. The creation of bibliographic records for library material is usually done by professional catalogers. Hence, bibliographic data represent an important class of information provided by the public libraries. Lately, new forms of bibliographic data have emerged due to the need to index publications on the Internet. For example, the Unified Computer Science Technical Report Index (UCSTRI) [Van Heyningen 1994] and many other online bibliographic resources [Ley 1995; Jones 1996] have been created to help Internet users to locate published computer science literature. Nevertheless, these new index servers may not adopt a common query interface to their bibliographic data and they also do not capture the large bulk of bibliographic data maintained by the public libraries.

—*Structured data:* Structured data have traditionally been used to store business and organization information. As SQL database systems become inexpensive, they are becoming widely used. Since modern SQL database systems can also be used to store text data and provide sophisticated text query features, we expect many of the SQL databases will be used as components of digital libraries to store information related to digital libraries. For example, SQL databases may be used to store information about interlibrary loan requests and books that have been borrowed or reserved.

—*Text and multimedia document data:* While many public libraries are now in the process of digitizing some or all of their library collections, library users have been gaining interest in the document collections found on the Internet. These document data can exist in a variety of formats. Some of them are totally unstructured, e.g., plain text files. Others may be semistructured. They can be represented by some mark-up languages such as SGML [ISO 1986], HTML [Graham 1995], etc. At present, the most popular way to obtain remote document files is through a web browser. A large number of document files can also be obtained from FTP (file transfer protocol) and WAIS [Kahle and Medlar 1991] (wide-area information servers) sites.

In this article, we address the important query-processing problem involving both legacy library catalogs and SQL databases containing structured data. This type of integrated querying is often desired in situations where SQL databases have been used to store some bibliographic information. Consider the following scenario. To perform an interlibrary loan, a library user has to first register a request with his or her affiliated library. Assume that all loan requests are stored in an SQL database and that they are processed in batch by a librarian. Having approved a number of registered requests, the librarian will attempt to locate the requested books in neighboring public libraries. In this process, it would be useful to provide integrated queries to the remote library catalogs and the SQL database.

We have developed an SQL-like query language known as HarpSQL for querying legacy public library catalog(s) and SQL server(s). The Z39.50 information retrieval protocol [NISO 1992] has been adopted to access the existing library catalog databases. We have also designed and implemented a query evaluation strategy for queries that involve legacy library catalogs and SQL databases. The query evaluation strategy includes the processing of joins between SQL tables and bibliographic data. We have also developed a distributed query processor that incorporates the tuple substitution join strategy. A graphical query formulation tool has been created.

1.2 Research Issues

A number of research issues have to be considered in the design of a distributed query system for legacy public libraries and structured databases:

—*Modeling of legacy public library catalogs:* Most existing public library catalogs contain bibliographic records represented in **MARC**[1] format [Crawford 1984]. Each MARC record consists of multiple fields representing bibliographic elements, e.g., title, author, subject, ISBN, etc. These bibliographic elements are identified by unique tag numbers (defined by the MARC standard). A selected set of MARC fields and their tags are shown in Figure 1. A bibliographic record formatted in MARC is shown in Figure 2. Sometimes, a bibliographic element may occur more than once in the same record, and sometimes each occurrence contains a different value (e.g., the example MARC record has multiple fields with the tag 650). Furthermore, a field may be composed by one or more subfields each carrying a specific meaning and a subtag (e.g., $a). In this article, we will present an extended SQL model known as HarpSQL to query these existing bibliographic data. Note that MARC is able to accommodate references to Internet resources by keeping Universal Resource Locators (URLs) in the MARC records (using fields with tag 856) as shown in Figure 1.

[1]MARC is the abbreviation of MAchine-Readable Cataloging.

Tag No.	Field name
001	Control number
020	ISBN number
040	Cataloging source
092	Call number
100	Main entry - personal name
110	Main entry - corporate name
111	Main entry - conference or meeting
245	Title statement
250	Edition statement
260	Publisher
600	Subject added entry - personal name
610	Subject added entry - corporate name
650	Subject added entry - topical heading
856	Universal Resource Locator (URL)

Fig. 1. Selected MARC fields and tags.

—*Query capabilities of remote access protocols:* To ensure that our distributed query-processing strategy is applicable to the present and future public library catalogs, we have adopted the Z39.50 protocol to query these databases [NISO 1992]. Z39.50 is an application layer information retrieval protocol drafted by ANSI/NISO. It has been widely used to support remote accesses to the library catalogs maintained by public libraries. At present, the queries supported by most Z39.50 servers are restricted to boolean searches (or selection queries) which at most consist of predicates on the MARC fields connected by boolean operators (AND, OR, NOT). Projection and join operations are not supported. In other words, our distributed query system has to accommodate the query restriction imposed by Z39.50. For example, our query system has to ensure that only selection queries are submitted to Z39.50 servers. Moreover, the query system must support join and projection operations which are not part of Z39.50 query support.

—*Merging different kinds of data:* Apart from retrieving data from remote SQL and bibliographic databases, our query system has to be able to merge the retrieved SQL and bibliographic data. In our approach, the distributed query system supports extended predicates to be used in joining the two kinds of data.

1.3 Article Outline

The rest of this article is organized as follows. In Section 2, we discuss related work. Section 3 describes an extended query language (HarpSQL) for writing integrated queries to bibliographic and SQL databases. Section 4 presents our distributed query-processing architecture. Processing

Bibliographic record:

Senn, James A. Information technology in business:
principles, practices, and opportunities.
Annotated instructor's ed.
Englewood Cliffs, N.J.: Prentice Hall, c1995.

Corresponding MARC record:

```
020  $a 0134849086 (Instructor's ed.)
020  $a 0134843045 (Student ed.)
040  $a DLC $c DLC $d DLC
100  $a Senn, James A.
245  $a Information technology in business :
      $b principles, practices, and opportunities
      $c James A. Senn.
250  $a Annotated instructor's ed.
260  $a Englewood Cliffs, N.J. : $b Prentice Hall,
      $c c1995.
650  $a Business $x Data processing.
650  $a Information storage and retrieval systems
      $x Business.
650  $a Information technology.
650  $a Local area networks (Computer networks)
```

Fig. 2. A bibliographic record example formatted in MARC.

HarpSQL queries will be given in Section 5. Implementation of our distributed query processor is described in Section 6. The design and implementation of a graphical query formulation tool is presented in Section 7. Discussions on possible HarpSQL extensions and conclusions are given in Sections 8 and 9 respectively.

2. RELATED WORK

Querying multiple heterogeneous library databases simultaneously over the network has been the research focus of several digital library projects recently. In the Alexandria Digital Library project [Smith 1996], a new metadata standard combining MARC and Federal Geographic Data Committee (FGDC) standards was proposed to unify the metadata of heterogeneous geographic information collections. In the University of Michigan Digital Library project [Atkins et al. 1996], software agents that search remote catalog and index databases via a Z39.50 interface have been developed. A distributed search protocol known as Dienst has also been developed at the University of Cornell to conduct multiple bibliographic and full-text searches on different document databases [Lagoze and Davis

1995]. Nevertheless, these research efforts have not addressed integrated queries to both structured and bibliographic databases.

In this project, remote structured and bibliographic databases can be seen as distributed heterogeneous databases. Our work is therefore related to the current research in multidatabase query processing [Lim et al. 1995; Liu and Pu 1996; Salza et al. 1994]. Similar to multidatabase systems, digital library systems have to accommodate different types of autonomous and heterogeneous databases. However, most multidatabase research has focused on querying structured databases only [Salza et al. 1994]. The multidatabase query evaluation strategies have to be further modified before they can be used to query bibliographic data. Multidatabase query processing can also be very complicated when the semantic conflicts between participating databases have to be resolved [Lim et al. 1995].

As bibliographic data are semistructured, our research is related to some ongoing work in modeling and querying semistructured data [Blake et al. 1995; Papakonstantinou et al. 1995; Quass et al. 1995]. To model and query all kinds of semistructured data, Quass et al. have proposed a flexible data model and query language known as OEM and LOREL respectively [Papakonstantinou et al. 1995; Quass et al. 1995] . In a similar effort, Blake et al. have extended SQL to query semistructured data and their metadescription. Both approaches assume that every record instance has its own structure or metadescription. Bibliographic data in the library catalogs, however, are stored as MARC records sharing a fixed set of MARC fields. Hence, it is simpler to model the library catalogs as relations and to extend SQL to query them.

Although the distributed query-processing problem has been well studied in the domain of relational databases, there is very little research in processing distributed queries which involve both structured and bibliographic databases. In Chaudhuri et al. [1995], several join techniques have been proposed for queries which involve an external text data manager loosely coupled with a relational database system. These techniques include (a) *naive tuple substitution*, (b) *relational text processing*, (c) *semijoin*, and (d) *probing*.

The naive tuple substitution technique requires a join between relation and text to be translated into a set of selection queries to the text database. This is done by evaluating the relational query followed by substituting relational attributes in the join predicate by actual column values. This technique is usually undesirable because a large overhead will incur when numerous selection queries are sent to the text database. The relational text-processing technique assumes that the relational database system can handle the join predicates between relational attributes and text attributes. This assumption, however, does not hold in our context because the extended predicates and functions in our integrated queries cannot be handled by ordinary relational database systems. In Chaudhuri et al. [1995], the proposed semijoin technique is actually a variant of tuple substitution. Semijoin reduces the overhead of tuple substitution by combining all selection queries generated by tuple substitution into one selection query. The probing technique, designed to work together with either naive tuple substitution or relational text processing, further improves the two techniques by not sending queries that return empty results to the text system.

In our project, we deal with queries involving multiple external SQL databases and library catalogs. The distributed query-processing problem is therefore more complex. We have adopted a tuple substitution approach similar to semijoin to evaluate subqueries on library catalogs. The probing method is not chosen because we currently do not maintain the statistics the probing method requires for the external databases.

3. HARPSQL QUERY LANGUAGE

To enable digital library users and application developers to query existing SQL and bibliographic data, we have extended the SQL language in a number of ways and called it **HarpSQL** [Lu and Lim 1996]. In the current HarpSQL design, we have assumed that the users are familar with the MARC attribute set and the bibliographic attribute set adopted by the Z39.50 protocol. The HarpSQL language can however be further extended to make it more useable by novice users (see Section 8). The unique features of HarpSQL include the following:

—*Foreign SQL and bibliographic tables:* In HarpSQL, a table (say `CourseTB`) from a remote SQL database (say `RefDB`) can be imported as a foreign SQL table (`CourseTB@RefDB`). Unlike SQL databases, remote library catalogs do not contain member tables. Hence, each remote library catalog is imported as a single foreign **bibliographic table** (or **BIB table**). Each imported BIB table is named `BibTB@⟨Library name⟩` where `⟨Library name⟩` is the public library that provides the BIB table.

—*MARCString data type:* Bibliographic data found in the public libraries are mostly formatted based on the MARC standard. To model the MARC fields, we have defined a new data type called **MARCString**. A MARCString value models multiple MARC fields sharing a common tag number in a MARC record. These MARC fields form the **elements** of the MARCString value. Each element may consist of multiple **subelements** modeling the subfields in the MARC fields. Hence, an imported BIB table consists of multiple MARCString attributes each with a unique tag number and attribute name `MAttr⟨tag number⟩`. For example, the `MAttr650` attribute value of the record given in Figure 2 is

```
(650, ('$a Business' '$x Data processing')
('$a Information storage and retrieval systems' '$x Business')
('$a Information technology')
('$a Local area networks(Computer networks)'))
```

—*Virtual bibliographic tables:* To support broadcasting of queries to multiple library catalog databases, HarpSQL allows a **virtual bibliographic (BIB) table** to be defined upon a number of imported BIB tables which

RefTB	*RefId*	*Title*	*Author*	*Course*
CourseTB	*CourseId*	*Cname*	*Year*	*Lecturer*

Fig. 3. Schema of `RefDB`.

are also known as the **members BIB tables**. Apart from having the same MARCString attributes found in any BIB table, every virtual BIB table contains an extra `location` attribute to indicate where its records come from.

—`Contain` *and* `Extract` *predicates and functions:* With the new data type MARCString, a new predicate called `Contain` has been defined to apply different kinds of selection criteria on the MARCString attributes, and to allow BIB tables to be joined with SQL tables by comparing the MARCString attributes with the text attributes in the SQL tables. Unlike the usual regular-expression predicates, the `Contain` predicate caters for a wide variety of string comparison methods by supporting different search modes for different bibliographic elements [Lu and Lim 1996]. Each search mode is a quadruple of four submodes represented by $\langle position, structure, truncation, completeness \rangle$. The specification of the search mode is given in the Appendix. In order to query remote library catalogs using the Z39.50 protocol, all `Contain` predicates on the bibliographic attributes must adopt a standard attribute set known as Bib-1. Bib-1 attributes can be seen as standard surrogates for the MARC attributes. A mapping between the Bib-1 attribute set and the MARC attribute set is given in Table V, in the Appendix. Like MARC attributes, every Bib-1 attribute is assigned a tag number and a name denoted by `BAttr⟨Bib-1 tag⟩`. The `Extract` function, on the other hand, allows us to extract subelements from a MARCString value by supplying the subtags.

Example 1. Let `BibTB@NTU` and `BibTB@NUS` be two BIB tables imported from the NTU[2] library and NUS[3] library. Let `CourseTB@RefDB` and `RefTB@RefDB` be two SQL tables imported from `RefDB`, an SQL database containing some course information. `RefTB@RefDB` contains information about reference books adopted by different courses. `CourseTB@RefDB` contains information about the courses to be taken by computer engineering students. Their attributes are shown in Figure 3.

In the following, we show some query examples[4] demonstrating the HarpSQL features.

Example 2. (Q1): Retrieve the titles and authors of books with titles containing 'distributed database' from the NTU library.

[2]NTU is an abbreviation of Nanyang Technological University.
[3]NUS is an abbreviation of National University of Singapore.
[4]To simplify our explanation, some parameters to be used in `Extract` and `Contain` are not shown.

```
SELECT Extract(Mattr245,‘$a’), Extract(Mattr100,‘$a’
FROM BibTB@NTU
WHERE Contain(BAttr4,‘distributed database’,
⟨ANY_POSITION,IS_PHRASE,NULL,NULL⟩)
```

In the above HarpSQL query, MAttr245 and MAttr100 are the MARCString attributes containing the title and author information in their subelements with subtag $a. BAttr4 is the Bib-1 attribute for book title. The search mode ⟨ANY_POSITION,IS_PHRASE,NULL,NULL⟩ in the Contain predicate indicates that only those titles containing ‘distributed database’ as a phrase are wanted. The Extract functions are used to obtain title and author text from the MARCString attributes MAttr245 and MAttr100 respectively.

Example 3. (Q2): Retrieve the course titles, call numbers, titles, authors, and locations of reference books used by courses held in the academic year 95/96 from NTU and NUS libraries.

```
SELECT c.Cname, Extract(a.MAttr092, ‘$a’),
  Extract(a.MAttr245,‘$a’), Extract(a.MAttr100,‘$a’),
    a.location
FROM VL_NTUandNUS a, RefTB@RefDB b, CourseTB@RefDB c
WHERE c.Year = ‘95/96’ AND b.Course = c.CourseId AND
  Contain(a.BAttr4,b.Title,
    ⟨FIRST_IN_SUBFIELD,IS_PHRASE,NULL,NULL⟩) AND
  Contain(a.BAttr1003,b.Author,⟨NULL,IS_NAME,NULL,NULL⟩)
```

The MARCString attribute MAttr092 contains the call number information. BAttr1003 is the Bib-1 attribute for author. The above query specifies a join between two SQL tables and a virtual BIB table VL_NTUandNUS defined on the BIB tables imported from NTU and NUS libraries. Note that the Contain predicates have been used to join RefTB with VL_NTUandNUS.

4. DISTRIBUTED QUERY-PROCESSING ARCHITECTURE

As shown in Figure 4, our distributed query-processing architecture consists of two types of processes, namely **query managers** and **query agents**. Each HarpSQL query is handled by a query manager and a number of query agents, one for each remote database server to be accessed by the query. Given a HarpSQL query, a query manager is first created and in turn creates the appropriate query agents. The query manager first parses the query into a query graph that is later decomposed into a number of subqueries to be processed by its query agents. Having collected all the subquery results, the query manager combines them together and returns the final query result to the digital library application. Since the combination of subquery results cannot be performed by the individual query agents, a **HarpSQL server** is included into the query manager to provide the capabilities to store and process intermediate results.

Figure 4 also depicts the remote SQL and Z39.50 servers managing existing structured data and bibliographic data respectively. The SQL and

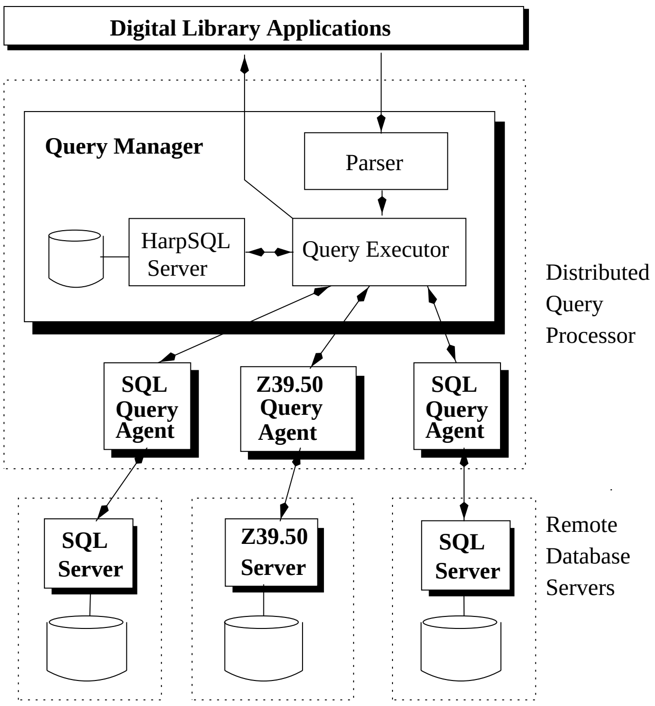


Fig. 4. Architecture of the distributed query processor.

Z39.50 query agents act as wrappers that support subqueries to remote SQL and Z39.50 servers which are members of the integrated digital library environment. A query agent receives subqueries from the query manager, sends them to its remote database server for processing, and returns the result to the query manager. The interaction between the query agents and their remote servers is governed by the specific remote access protocols supported by the servers. By using the query agents, the query manager is able to execute queries without knowing much about the complex protocols and query interfaces adopted by the remote servers. Furthermore, the query agents are designed to process their subqueries concurrently, thus reducing the query response time.

5. HARPSQL QUERY-PROCESSING STRATEGY

In this section, we describe the query-processing strategy adopted by our HarpSQL distributed query processor which has been developed based on the architecture given in Section 4. Although query optimization is not the prime focus of this research, our processing strategy has been designed to reduce the subquery results by performing selection and projection as early as possible and by avoiding cartesian products in the subqueries to be evaluated by the external servers. By reducing the subquery results, we are able to minimize the overhead of shipping data from the external servers to the distributed query processor. Upon receiving the subquery results, the

HarpSQL server will combine them together by performing some interdatabase joins or cartesian products.

5.1 Restricting Bibliographic Queries Using Tuple Substitution

As Z39.50 disallows bibliographic queries that do not carry any selection predicate,[5] our query-processing strategy requires all BIB tables involved in HarpSQL queries to be restricted by either selection or join with other SQL tables. For those BIB tables that are only restricted by join, we can derive the subqueries to their Z39.50 servers by performing **tuple substitution**.[6] In tuple substitution, a join predicate used in the join between a BIB table and an SQL table is transformed into a disjunctive set of selection predicates by first evaluating the SQL table, followed by instantiating the SQL attribute in the join predicate by the corresponding attribute values in the SQL subquery result.

For example, to process the query (Q3) below, we first evaluate the SQL subquery to obtain the various reference title values from `RefTB@RefDB`.

Example 4. (Q3): Retrieve the call number and title of books that are listed in the reference table.

```
SELECT Extract(a.MAttr092,'$a'),a.MAttr245
FROM BibTB@NTU a, RefTB@RefDB b,
WHERE
  Contain(a.BAttr4,b.Title,⟨FIRST_IN_SUBFIELD,IS_PHRASE,NULL,NULL⟩)
```

Suppose the titles returned from `RefTB@RefDB` are "Digital Design," "Computer Networks," etc. By tuple substitution, we obtain the following selection subquery for `BibTB@NTU`:

```
SELECT *
FROM BibTB@NTU
WHERE Contain(BAttr4,'Digital Design', ···) OR
  Contain(BAttr4,'Computer Networks', ···) OR ···
```

5.2 Distributed Query-Processing Steps

To process a HarpSQL query, we first represent it using a query graph [Wong and Youssefi 1976]. In a query graph, each node represents an SQL table, BIB table, or virtual BIB table. An edge between a pair of nodes represents a join. For example, the query graph representing Q2 in Section 3 is shown in Figure 5. Given a query graph, the query-processing steps performed by our distributed query processor are as follows:

—*Step 1 (SQL subgraph extraction):* When a query graph involves some SQL tables, we first derive the subqueries to these tables by extracting **SQL subgraphs** from the query graph. A SQL subgraph is a connected subgraph of a query graph consisting of SQL tables that belong to the

[5]This prevents huge amounts of bibliographic data to be shipped across sites.
[6]This is similar to the semijoin technique mentioned in Chaudhuri et al. [1995].

same SQL database. To extract the SQL subgraphs, a depth-first search is performed on the query graph.

—*Step 2 (Processing SQL subqueries):* Once the SQL subgraphs are extracted, we generate an SQL subquery for each subgraph. All these SQL subqueries are submitted to the SQL query agents created for the target SQL database servers and are evaluated by the servers concurrently. Typically, the SQL subqueries involve select, project, and intradatabase join operations. A temporary table for each subquery result is created at the HarpSQL server when the subquery result is returned by the query agent.

—*Step 3 (Processing bibliographic subqueries):* In this step, we derive and evaluate the subqueries against the BIB tables. These subqueries can be obtained in two ways as described below:

Case (a): If a BIB table (or virtual BIB table) node in the query graph is restricted by some selection predicate(s), a bibliographic subquery against the BIB table (or virtual BIB table) with the selection predicate(s) is derived.

Case (b): If a BIB table (or virtual BIB table) node in the query graph is not restricted by any selection predicate(s), we have to derive the bibliographic subquery by performing tuple substitution. In tuple substitution, the subquery result of an SQL subgraph that is linked to the BIB table (or virtual BIB table) is chosen[7] to convert a join predicate between the SQL subgraph and BIB table (or virtual BIB table) into a disjunction of selection predicates.

For each subquery against a BIB table, we create a Z39.50 query agent to process it. The subquery result returned by the query agent is stored as a temporary table at the HarpSQL server with the necessary attribute projection. In both cases (a) and (b), a subquery against a virtual BIB table will be further replicated into subqueries against its member BIB tables. Multiple Z39.50 query agents, each corresponding to a member BIB table, will be created to process these subqueries. The results of all these subqueries are unioned and stored as a temporary table in the HarpSQL server with the necessary attribute projection. In the process of unioning the subquery results, the location attribute value is added to every record.

—*Step 4 (Final result generation):* A final query that joins all the temporary tables is evaluated by the HarpSQL server. Apart from the final attributes to be projected, the final query may consist of (1) join(s) between tables from different SQL database servers and (2) join(s) between the SQL table and BIB table (except the join used for tuple substitution). The final query may also involve any remaining `Contain` predicates and `Extract` functions to be applied on the query result.

[7]If there are multiple SQL subgraphs adjacent to the BIB table, we just choose one of them.

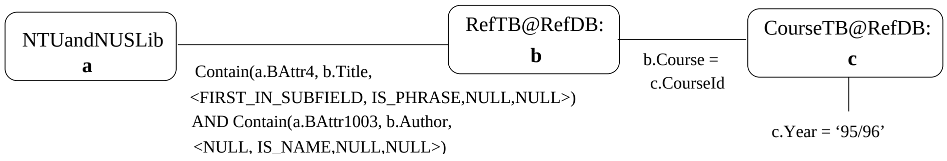

NTUandNUSLib
a
RefTB@RefDB:
b
CourseTB@RefDB:
c
Contain(a.BAttr4, b.Title,
<FIRST_IN_SUBFIELD, IS_PHRASE,NULL,NULL>)
AND Contain(a.BAttr1003, b.Author,
<NULL, IS_NAME,NULL,NULL>)
b.Course =
c.CourseId
c.Year = '95/96'

Fig. 5. A query graph example.

When the HarpSQL queries to be processed involve only SQL tables, only steps 1, 2, and 4 are required. On the other hand, if a HarpSQL query involves only a BIB table or virtual BIB table, we only need to perform steps 3 and 4. In steps 2 and 3, query agents are created to evaluate subqueries for remote servers. When any of the remote servers cannot be reached due to some network or server problems, or cannot return subquery results within a specified amount of time, the respective query agent will experience a timeout. The timeout in turn causes the query manager to return failure status for the given HarpSQL query.

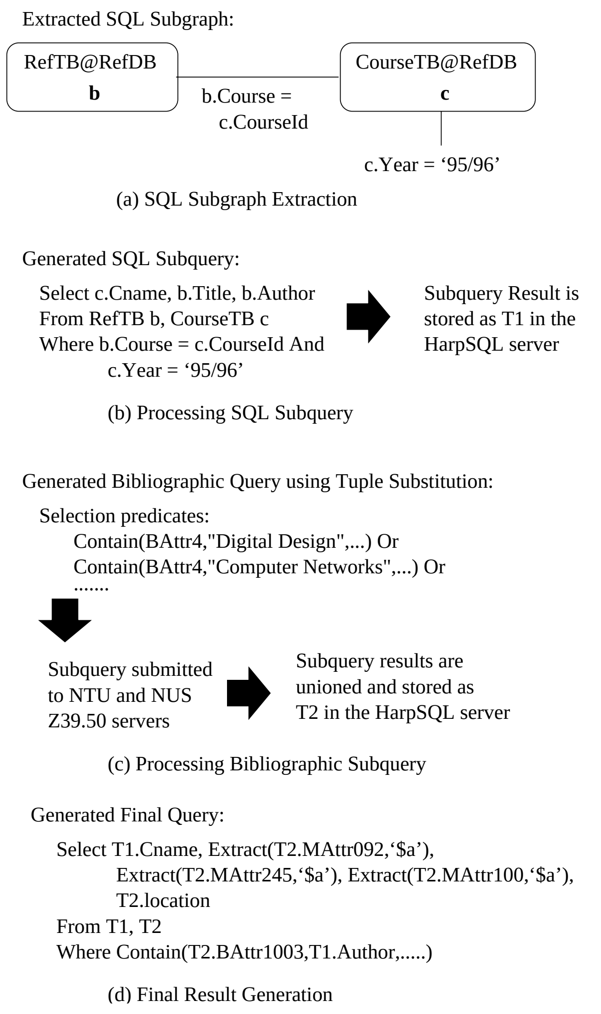
5.3 Query-Processing Example

Figure 6 shows the query-processing steps for our query example Q2. From the query graph (see Figure 5), we extract an SQL subgraph which is translated into an SQL subquery to be executed by an SQL query agent. The SQL subgraph is shown in Figure 6(a). From the subgraph, we generate an SQL subquery and send it to an SQL query agent as shown in Figure 6(b). A temporary table T1 is created at the HarpSQL server for the subquery result. Subsequently, we substitute the `Title` attribute in the `Contain` predicate by the corresponding attribute values in the previous SQL subquery result T1. A substituted BIB subquery is created and is submitted to the two BIB query agents for the NTU and NUS libraries as shown in Figure 6(c). The results from all the BIB query agents are unioned and stored in a temporary table T2 by the HarpSQL server. Finally, a query that joins T1 and T2 is evaluated by the HarpSQL server to obtain the final query result as shown in Figure 6(d).

6. IMPLEMENTATION ISSUES

As part of our research work, we have developed a distributed query processor that can handle HarpSQL queries over a collection of SQL and Z39.50 servers. Within the distributed query processor, the query manager and agents are implemented as separate processes. Message queues have been used for communication between the query manager and agent processes.

Since the query manager is responsible for storing and processing intermediate results collected from different remote servers, we need a HarpSQL server that can handle both SQL and bibliographic data on behalf of the query manager. In the following subsection, we describe how we realize the HarpSQL server by extending the POSTGRES database system.


Extracted SQL Subgraph:
RefTB@RefDB
b
CourseTB@RefDB
c
b.Course = c.CourseId
c.Year = ‘95/96’
(a) SQL Subgraph Extraction
Generated SQL Subquery:
Subquery Result is stored as T1 in the HarpSQL server
(b) Processing SQL Subquery
Generated Bibliographic Query using Tuple Substitution:
Subquery submitted to NTU and NUS Z39.50 servers
Subquery results are unioned and stored as T2 in the HarpSQL server
(c) Processing Bibliographic Subquery
Generated Final Query:
(d) Final Result Generation

```
Select c.Cname, b.Title, b.Author
From RefTB b, CourseTB c
Where b.Course = c.CourseId And
          c.Year = ‘95/96’
```

```
Selection predicates:
    Contain(BAttr4,"Digital Design",...) Or
    Contain(BAttr4,"Computer Networks",...) Or
    .......
```

```
Select T1.Cname, Extract(T2.MAttr092,‘$a’),
         Extract(T2.MAttr245,‘$a’), Extract(T2.MAttr100,‘$a’),
         T2.location
From T1, T2
Where Contain(T2.BAttr1003,T1.Author,.....)
```

Fig. 6. Query-processing steps for Q3.

6.1 HarpSQL Server Implementation

To play a role in processing distributed HarpSQL queries, the HarpSQL server supplementing the query manager must support the following features:

—Basic SQL data types and the MARCString data type

—`Contain()` predicate

—`Extract()` function

Instead of building the HarpSQL server from scratch, we base our implementation on the POSTGRES database system [Stonebraker and Rowe 1986]. One key difference between POSTGRES and standard relational systems is that POSTGRES captures extra information in its catalog,

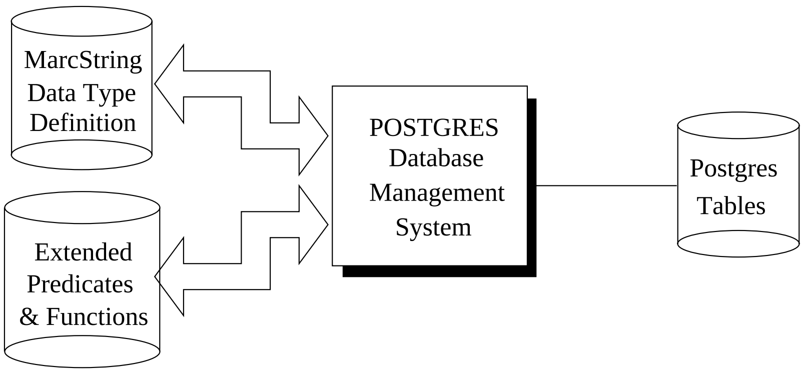


Fig. 7. HarpSQL server architecture.

which allows its processing and storage capabilities to be extended. This includes not only information about tables and fields, but also information about types, functions, access methods, and etc. This information can be modified by the user, and POSTGRES carries out its internal operation based on this information. The query language of POSTGRES is known as POSTQUEL. POSTGRES can also incorporate precompiled user-written code into its query processing through dynamic loading. In other words, the user can create an object file (e.g., a compiled (dot "oh") .o file or shared library) that implements new types and function in POSTGRES. A detailed description can be found in The POSTGRES Group [1994].

Figure 7 shows the architecture of the HarpSQL server. Our HarpSQL server is developed by augmenting POSTGRES with the MARCString data type and its extended predicates and functions. In order to support the MarcString data type, a C data structure is first defined and is used by the POSTGRES database system as an internal representation of a MARC-String value. After the MARCString datatype is defined, the `Contain` predicate and `Extract` function were implemented in C and incorporated into POSTGRES.

7. GRAPHICAL QUERY FORMULATION TOOL

In this section, we describe the design and implementation of the HarpSQL query formulation tool. The HarpSQL query formulation tool allows users to load and view schemas, to formulate HarpSQL queries, and to view the query results. It is designed for digital library application developers and sophisticated end-users to formulate and evaluate their HarpSQL queries. Using the query formulation tool, one can store queries for future reference or execution. Since library catalog records may contain references to Internet resources, our query tool can be configured to invoke a web browser to view the Internet resources directly.

In the design of our query formulation tool, the following criteria have been adopted:

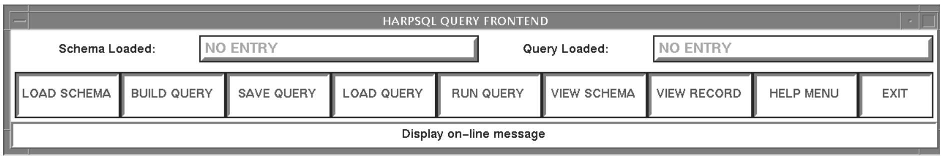


Fig. 8. MAIN window.

—*Online help subsystem:* An online help subsystem plays an important role in guiding new users. Two kinds of help information have been provided by our GUI design: (1) a HELP window that displays the relevant help information and (2) a message bar which dynamically displays brief help messages about the icons or dialog boxes on which the cursor is placed.

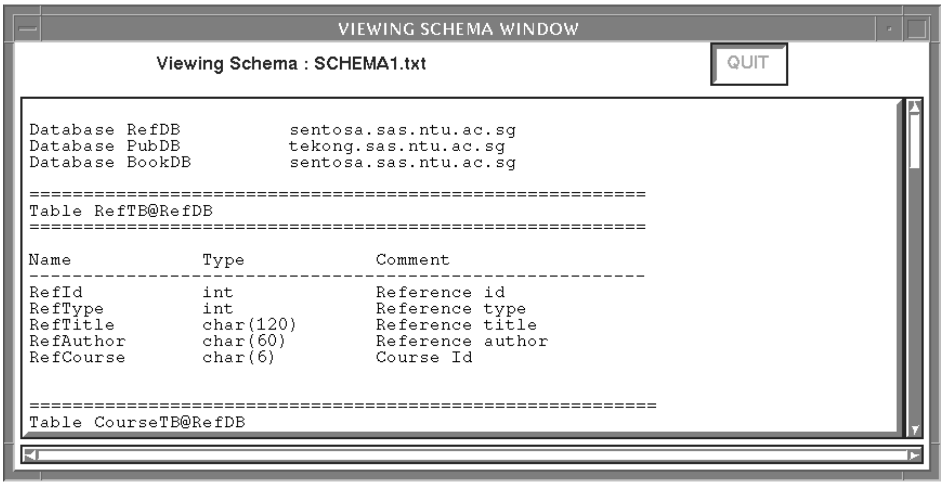
—*Multiwindow design:* Our query formulation tool involves multiple window objects that can be moved, minimized, and resized freely. However, the dependencies between these window objects are strictly maintained by our query tool. For example, an update to a window object is automatically propagated to the other window objects that are affected by the change.

—*User input assistance:* In order to minimize typographical errors, users are given as much assistance as possible during their query construction. Using the query tool, a user can easily formulate HarpSQL queries using the mouse. Keyboard input has been avoided as much as possible.

7.1 Interface Design and Functional Description

Our HarpSQL query formulation tool consists of a number of window objects. Among them, a MAIN window is designed to be an entry point to invoke query-related functions. As shown in Figure 8, The MAIN window consists of nine buttons: LOAD SCHEMA, BUILD QUERY, SAVE QUERY, LOAD QUERY, RUN QUERY, VIEW SCHEMA, VIEW RECORD, HELP MENU, and EXIT. To prevent users from choosing inappropriate buttons, some of the buttons may be disabled. For instance, the BUILD QUERY and VIEW SCHEMA buttons are disabled when the schema of imported library catalogs and structured databases has not been loaded. The online message bar at the bottom of MAIN window provides the dynamic help and status information during query formulation. The MAIN window also indicates the currently loaded schema and query. Once a schema is loaded, it can be browsed by invoking the VIEW SCHEMA button as shown in Figure 9.

7.1.1 *Query Construction Windows.* In our query tool, HarpSQL queries can be constructed in two ways. One can construct new HarpSQL queries from scatch using the BUILD QUERY option. Otherwise, new queries can be constructed by modifying prestored queries using the LOAD QUERY option. In both cases, the same three query construction windows will be activated. They are the TABLE LIST (FROM), TARGET LIST (SELECT), and PREDICATE LIST (WHERE) windows for specifying the `FROM`, `SE-`



```
Database RefDB          sentosa.sas.ntu.ac.sg
Database PubDB          tekong.sas.ntu.ac.sg
Database BookDB         sentosa.sas.ntu.ac.sg

========================================================
Table RefTB@RefDB
========================================================

Name            Type            Comment
--------------------------------------------------------
RefId           int             Reference id
RefType         int             Reference type
RefTitle        char(120)       Reference title
RefAuthor       char(60)        Reference author
RefCourse       char(6)         Course Id

========================================================
Table CourseTB@RefDB
```

Fig. 9. VIEW SCHEMA window.

LECT, and WHERE clauses respectively. These three windows are shown in Figures 10, 11, and 12.

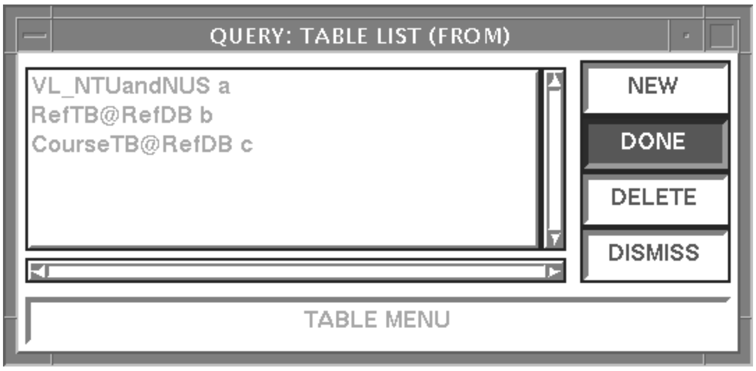
The TABLE LIST (FROM) window allows users to select the tables required by the query and to provide an alias for each of them. The TARGET LIST (SELECT) window allows users to select the target attributes of the tables included in the TABLE LIST (FROM) window. When an attribute of MarcString type is chosen, a TARGET FUNCTION window (see Figure 13) will appear, and the user can choose to apply the `Extract` or `MarcToText`[8] function on the MarcString attribute.

The PREDICATE LIST (WHERE) window allows users to construct the predicates based on the selected tables in the TABLE LIST (FROM) window. For each predicate, the OPERAND#1, OPERAND#2, and OPERATOR options must be specified. When the `Contain` function is chosen as the operator, a CONTAIN PREDICATE window (see Figure 14) will appear, allowing the user to specify the search mode.

When a query is built, it can be saved using the SAVE QUERY button. The LOAD QUERY button can later be used to load any saved query for modification or evaluation.

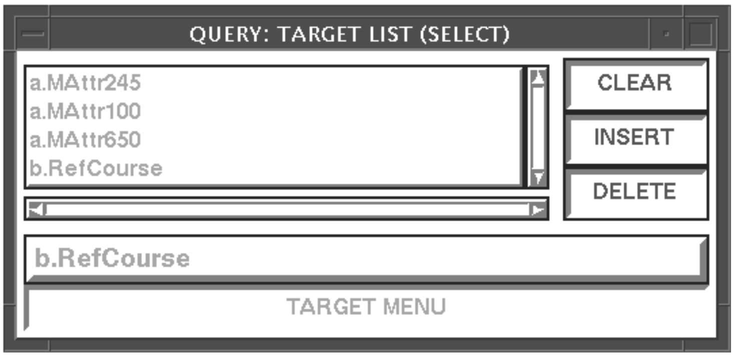
7.1.2 *Query Result Window.* A constructed HarpSQL query can be evaluated directly within the query formulation tool by invoking the RUN QUERY option. Once the query evaluation is completed, the result can be viewed within a QUERY RESULT window as shown in Figure 15. Apart from viewing the query result, one can also choose to save or print the result. The VIEW RECORD option in the MAIN window can later be invoked to reload any saved query result.

[8]`MarcToText` is a function that flattens a MarcString value into a plain text string. It is used mainly for debugging purposes.



```
VL_NTUandNUS a
RefTB@RefDB b
CourseTB@RefDB c
```

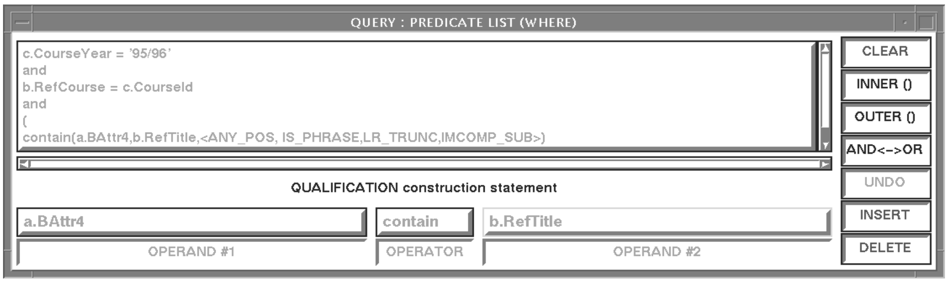
Fig. 10. TABLE LIST (FROM) window.



```
a.MAttr245
a.MAttr100
a.MAttr650
b.RefCourse
```

```
b.RefCourse
```

Fig. 11. TARGET LIST (SELECT) window.



```
c.CourseYear = '95/96'
and
b.RefCourse = c.CourseId
and
(
contain(a.BAttr4,b.RefTitle,<ANY_POS, IS_PHRASE,LR_TRUNC,IMCOMP_SUB>)
```

```
a.BAttr4    contain    b.RefTitle
```

Fig. 12. PREDICATE LIST (WHERE) window.

Figure 16 shows a QUERY RESULT window showing a query result that contains URL information as clickable objects. When any of the URL values in the window is selected, a web browser will be invoked to browse the corresponding Internet document.

7.2 Implementation of the Query Formulation Tool

The HarpSQL query formulation tool is built in the UNIX environment using Tcl/Tk [Ousterhout 1993]. Tcl/Tk provides a programming environ-

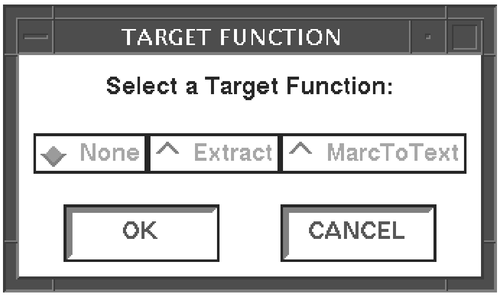


Fig. 13. TARGET FUNCTION window.

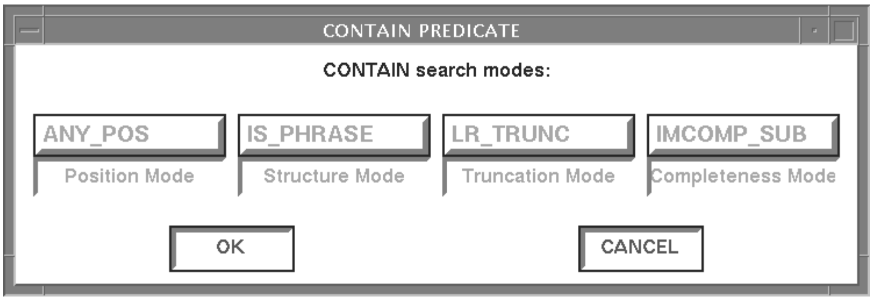
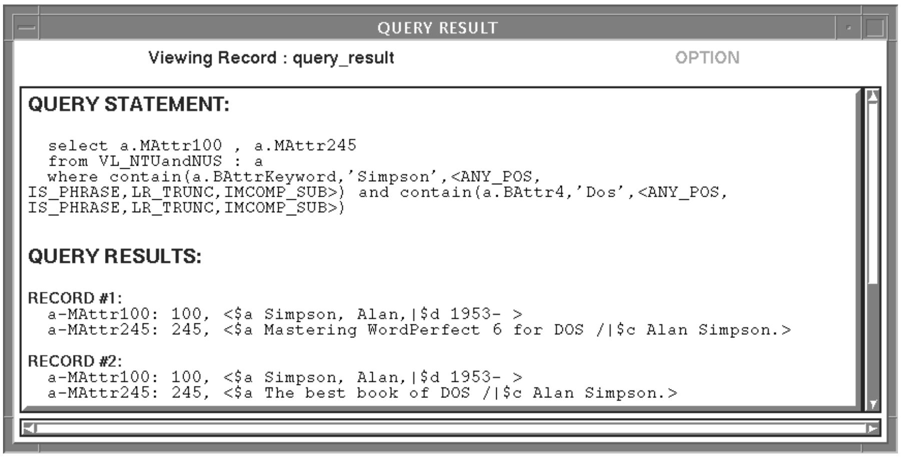


Fig. 14. CONTAIN PREDICATE window.



QUERY STATEMENT:

```
  select a.MAttr100 , a.MAttr245
  from VL_NTUandNUS : a
  where contain(a.BAttrKeyword,'Simpson',<ANY_POS,
IS_PHRASE,LR_TRUNC,IMCOMP_SUB>) and contain(a.BAttr4,'Dos',<ANY_POS,
IS_PHRASE,LR_TRUNC,IMCOMP_SUB>)
```

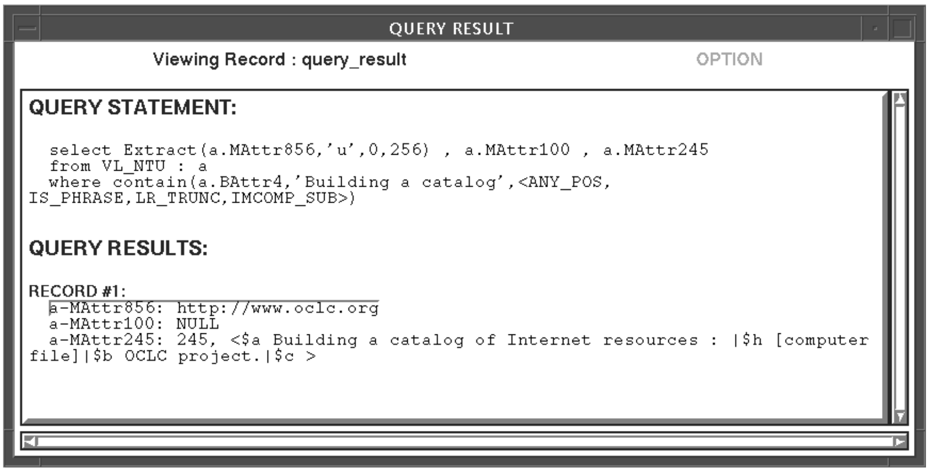
QUERY RESULTS:

```
RECORD #1:
  a-MAttr100: 100, <$a Simpson, Alan,|$d 1953- >
  a-MAttr245: 245, <$a Mastering WordPerfect 6 for DOS /|$c Alan Simpson.>

RECORD #2:
  a-MAttr100: 100, <$a Simpson, Alan,|$d 1953- >
  a-MAttr245: 245, <$a The best book of DOS /|$c Alan Simpson.>
```

Fig. 15. QUERY RESULT window.

ment for developing and using graphical user interface applications. Tcl is a simple scripting language that provides generic programming facilities. Tk, a toolkit for the X Window System, extends the core Tcl facilities with additional commands for building user interfaces, so that users can construct Motif user interfaces by writing Tcl scripts instead of C codes.



QUERY STATEMENT:

```
  select Extract(a.MAttr856,'u',0,256) , a.MAttr100 , a.MAttr245
  from VL_NTU : a
  where contain(a.BAttr4,'Building a catalog',<ANY_POS,
IS_PHRASE,LR_TRUNC,IMCOMP_SUB>)
```

QUERY RESULTS:

RECORD #1:

```
  a-MAttr856: http://www.oclc.org
  a-MAttr100: NULL
  a-MAttr245: 245, <$a Building a catalog of Internet resources : |$h [computer
file]|$b OCLC project.|$c >
```

Fig. 16. QUERY RESULT window with document URL.

8. DISCUSSIONS

The use of MARC tags to represent bibliographic fields has become a well-accepted practice in most legacy public libraries. The mapping between Bib-1 attributes and MARC tags further allows us to query bibliographic databases using the Z39.50 protocol. Based on the MARC and Bib-1 standards, the HarpSQL query language and distributed query system have been developed. Nevertheless, a number of problems may have to be dealt with when MARC and Bib-1 are used. In the following, we discuss some of these problems and present possible extensions to HarpSQL and our query system in order to handle them.

8.1 Knowledge about MARC and Bib-1 Attributes

Our current query system relies heavily on users knowing the MARC and Bib-1 attributes before they formulate HarpSQL queries. However, most users may not be able to remember the MARC or BIB tag numbers. To address this problem, our query system can be extended with an alias mechanism that allows a meaningful bibliographic field name to be defined as an alias of a MARC or Bib-1 attribute. For example, one can define the following MARC and Bib-1 aliases.

```
DEFINE ALIAS MTitle AS MAttr245;
DEFINE ALIAS MAuthor AS MAttr100;
DEFINE ALIAS BTitle AS BAttr4;
```

With the above aliases, the query example Q1 in Example 2 can be rewritten as follows:

```
SELECT Extract(MTitle,'$a'), Extract(MAuthor,'$a')
FROM BibTB@NTU
WHERE Contain(BTitle,'distributed database',
⟨ANY_POSITION,IS_PHRASE,NULL,NULL⟩)
```

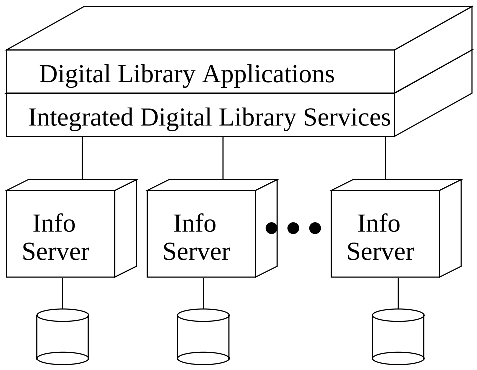

Digital Library Applications
Integrated Digital Library Services
Info Server
Info Server
Info Server

Fig. 17. Integrated digital library architecture.

Compared to its original form, the above query is clearly more readable. The MARC and Bib-1 alias mechanism can be incorporated into our distributed query system in a number of ways. For example, the query system could support a default set of MARC and Bib-1 aliases shared by all users. The query system could also be designed to allow users to define their own aliases.

8.2 Inconsistent Use of MARC Tags

We have so far assumed that MARC tags are consistently used across libraries during the cataloging process. This assumption may not hold for all libraries because tags may be used slightly differently due to different cataloging methods practiced by the libraries. Given a query, the inconsistent use of tags leads to unexpected query results from different bibliographic databases. For example, a book that exists in both libraries A and B may only be included in the query result from library A but not library B.

At present, our query system does not handle the inconsistency problem. Nevertheless, the problem can be overcome by the addition of metadata to each bibliographic database. The metadata could include information such as the cataloging techniques practiced by the library concerned. Based on the metadata of each bibliographic database, our distributed query system can be enhanced to modify queries accordingly before they are submitted to the bibliographic database server. When query results are returned from the server, further interpretation of the results has to be performed using the metadata information. In particular, individual bibliographic fields in the query results must be extracted correctly using the knowledge about how tags are used by the respective libraries.

9. CONCLUSIONS

In this article, we describe a distributed query system that allows users to formulate integrated queries to legacy public library catalogs and SQL databases. Our proposed query language, HarpSQL, supports a new data type, predicate, and function required for representing and manipulating

Table I. Position Submode

Option	Meaning
First in element	Search term must be the first data in the element.
First in subelement	Search term must be the first data in any subelement.
Any Position*	Search term may appear at any place.

Table II. Structure Submode

Option	Meaning
IS_WORD	A word search term contains no blanks. It specifies the exact text of the value to be searched.
IS_PHRASE*	A phrase search term consists of one or more words separated by blanks. It will be treated with respect to order and adjacency.
IS_WORD_LIST	A word list search term consists of one or more words separated by blanks. No order of the words is implied.
IS_NAME	The search term is treated as a person name.
IS_STRING	The entire term is to be treated as a string, rather than a sequence or set of individual words.

MARC-formatted bibliographic data. By accommodating MARC-formatted data and by adopting the Z39.50 protocol standard to access the bibliographic databases in public libraries, we achieve interoperability while not sacrificing the local autonomy of the existing library systems. HarpSQL further supports joins between SQL and bibliographic data.

To process HarpSQL queries over SQL and bibliographic databases at different locations, we have designed and implemented a distributed query processor which adopts some heuristics to reduce communication costs during query processing. To handle interdatabase joins including joins between SQL and bibliographic data, we implemented a HarpSQL server which provides the query-processing capabilities to the query manager. Moreover, we have also implemented a user-friendly graphical query front-end for users to formulate their HarpSQL queries. The distributed query system has been tested on the bibliographic database of the Nanyang Technological University Library. The bibliographic database consists of over 217,000 records. Using multiple query agents to access the same bibliographic database over the Internet, we managed to simulate multiple bibliographic databases using only one physical bibliographic database. By doing this, we reduced the amount of time spent on system testing significantly.

A digital library system typically consists of three layers of software, namely the **digital library applications**, **digital library services**, and **information servers** as shown in Figure 17. The work presented in this article represents an effort in the digital library service layer. Our distributed query-processing technique therefore represents an important step toward advanced query support for future digital library applications.

Table III. Truncation Submode

Option	Structure Option	Meaning
RIGHT_TRUNC*	Word/Phrase	Last word of term is right truncated.
	String	Entire term is right truncated.
	Word list	Each word is right truncated.
LEFT_TRUNC	Word/Phrase	First word of term is left truncated.
	String	Entire term is left truncated.
	Word list	Each word is left truncated.
LEFT_AND_RIGHT_TRUNC	Word/Phrase	First word of term is left truncated. Last word of term is right truncated.
	String	Entire term is left and right truncated.
	Word list	Each word is left and right truncated.
DO_NOT_TRUNC		No truncation is to be applied.
PROCESS_#		The search term contains "#" to show where truncation will take place.
REGULAR_EXP		The term is in the form of a regular expression.

Table IV. Completeness Submode

Option	Meaning
INCOMPLETE_SUBELEMENT*	Words other than those in the search term may appear in the element/subelement in which the term appears.
COMPLETE_ SUBELEMENT	No words other than those in the search term should appear in the subelement in which the term appears.
COMPLETE_ ELEMENT	No words other than those in the search term should appear in the element in which the term appears.

We are currently extending our work in several directions. First, we are considering the use of cost-based optimization techniques in processing the HarpSQL queries. Second, we plan to extend the HarpSQL to query other forms of data, e.g., Web pages, since the latter represents a fast-growing source of information on the Internet. Finally, we are attempting to build some advanced digital library applications, e.g., interlibrary loan, using HarpSQL and our distributed query processor.

APPENDIX

SEARCH MODE OF CONTAIN( )

In Tables I–IV, for each submode, the option marked with an asterisk is the default value. The mapping between BIB-1 and MARC attributes is shown in Table V.

Table V. Mapping between Bib-1 And MARC Attributes

Bib-1 id	Name	MARC Tag(s)
62	Abstract	520
1003	Author	100, 110, 111, 400, 410, 411, 700, 710, 711, 800, 810, 800
1000	Author-title	100/2XX, 110/2XX, 111/2XX, 400, 410, 411, 700, 710, 711, 800, 810, 811
1005	Author-name corporate	110, 410, 710, 810
1006	Author-name conference	111, 411, 711, 811
1004	Author-name personal	100, 400, 700, 800
13	Dewey classification	082
16	LC call number	050
55	Code-geographic area	043
56	Code-institution	040
54	Code-language	008, 041
9	LC card number	010, 011
12	Local number	001, 035
30	Date	005, 008, 260, 033, etc.
31	Date of publication	008, 260, 046, 533
1011	Date/time added to database	008
1012	Date/time last modified	005
7	ISBN	020
8	ISSN	022, 4XX, 7XX
1007	Identifier-standard	010, 011, 015, 017, 018, 020, 022, 023, 024, 025, 027, 028,030, 035, 037
1002	Name	100, 110, 111, 400, 410, 411, 600, 610, 611, 700, 710, 711, 800, 810, 811
57	Name and title	100/2XX, 110/2XX, 111/2XX, 400, 410, 411, 600, 610, 611, 700, 710, 711, 800, 810, 811
2	Corporate name	110, 410, 610, 710, 810
3	Conference name	111, 411, 611, 711, 811
58	Name geographic	651
1	Personal name	100, 400, 600, 700, 800
63	Note	5xx
21	Subject heading	600, 610, 611, 630, 650, 651, 653, 654, 655, 656, 657, 69X
24	INSPEC subject	600, 610, 611, 630, 650, 651
27	LC subject heading	600, 610, 611, 630, 650, 651
1008	Subject-LC children's	600, 610, 611, 630, 650, 651
1009	Subject name-personal	600
4	Title	130, 21X-24X, 400, 410, 440, 490, 600, 610, 611, 700, 710, 711, 730, 740, 800, 810, 811, 830, 840
43	Title abbreviated	210, 211, 246
37	Title added-title-page	246
38	Title caption	246
34	Title collective	243
36	Title cover	246
5	Title series	400, 410, 411, 440, 490, 800, 810, 811, 830, 840
6	Title uniform	130, 240, 700, 710, 711, 730

ACKNOWLEDGMENTS

We thank Nanyang Technological University and National Computer Board for supporting this research work. We also thank the anonymous referees for giving their helpful comments on this article which improved both the article's content and presentation.

REFERENCES

ATKINS, D. E., BIRMINGHAM, W. P., DURFEE, E. H., GLOVER, E. J., MULLEN, T., RUNDENSTEINER, E. A., SOLOWAY, E., VIDAL, J. M., WALLACE, R., AND WELLMAN, M. P. 1996. Toward inquiry-based education through interacting software agents. *IEEE Computer 29*, 5, 69–76.

BLAKE, G., CONSENS, M., DAVIS, I., KILPELAINEN, P., KUIKKA, E., LARSON, P.-A., SNIDER, T., AND TOMPA, F. 1995. Text/relational database management systems: Overview and proposed SQL extentions database prototype. Tech. Rep. 95-25. Centre for the New OED and Text Research, University of Waterloo, Waterloo, Canada.

CHAUDHURI, S., DAYAL, U., AND YAN, T. W. 1995. Join queries with external text sources: execution and optimization techniques. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (SIGMOD '95, San Jose, CA, May 23–25, 1995), M. Carey and D. Schneider, Eds. ACM Press, New York, NY, 410–422.

CRAWFORD, W. 1984. *MARC for Library Use: Understanding the USMARC Formats*. Knowledge Industry Publications, Inc., White Plains, NY.

GRAHAM, I. 1995. *The HTML Sourcebook*. John Wiley & Sons, Inc., New York, NY.

ISO. 1986. International Standard 8879: Information processing—text and office systems—Standard Generalized Markup Language (SGML). International Standards Organization. Ref. No. ISO 8879-1986(E).

JONES, D. M. 1996. The Hypertext Bibliography Project. Tech. Rep.. MIT Laboratory for Computer Science, Cambridge, MA. http://theory.lcs.mit.edu˜dmjones/hbp/.

KAHLE, B. AND MEDLAR, A. 1991. An information system for corporate users: Wide area information servers. *Online 15*, 5 (Sept. 1991), 56–60.

LAGOZE, C. AND DAVIS, J. R. 1995. Dienst: An architecture for distributed document libraries. *Commun. ACM 38*, 4 (Apr. 1995), 47.

LEY, M. 1995. DB&LP: A WWW bibliogrphy on databases and logic programming. Tech. Rep.. Informatik Universitat, Trier, Germany.

LIM, E.-P., SRIVASTAVA, J., AND HWANG, S.-Y. 1995. An algebraic transformation framework for multidatabase queries. *Distrib. Parallel Databases 3*, 3 (July 1995), 273–307.

LIU, L. AND PU, C. 1996. Issues on query processing in distributed and interoperable information systems. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications* (Kyoto, Japan, Dec.).

LU, Y. AND LIM, E.-P. 1996. On integrating existing bibliographic databases and structured databases. In *Proceedings of the IEEE International Computer Software and Applications Conference* (COMPSAC '96, Aug.). IEEE Press, Piscataway, NJ.

NISO. 1995. Information Retrieval (Z39.50): Application service definition and protocol specification. Tech. Rep. ANSI/NISO Z39.50-1995. NISO Press, Bethesda, MD. Available via http://lcweb.loc.gov/z3950/agency/.

OUSTERHOUT, J. 1993. *An Introduction to Tcl and Tk*. Addison-Wesley, Reading, MA.

PAPAKONSTANTINOU, Y., GARCIA-MOLINA, H., AND WIDOM, J. 1995. Object exchange across heterogeneous information sources. In *Proceedings of the IEEE International Conference on Data Engineering* (Mar.). IEEE Press, Piscataway, NJ.

QUASS, D., RAJARAMAN, A., SAGIV, Y., ULLMAN, J., AND WIDON, J. 1995. Querying semistructured heterogeneous information. In *Proceedings of the 4th International Conference on Deductive and Object-Oriented Databases* (Singapore, Dec.). Springer-Verlag, Berlin, Germany.

SALZA, S., BARONE, G., AND MOZRY, T. 1994. Distributed query optimization in loosely coupled multidatabase systems. In *Proceedings of the International Conference on Database Theory* (Prague).

SMITH, T. 1996. A digital library for geographicaly referenced materials. *IEEE Comput. 29*, 5 (May), 54–60.

STONEBRAKER, M. AND ROWE, L. A 1986. The design of POSTGRES. *SIGMOD Rec. 15*, 2 (June 1986), 340–355.

THE POSTGRES GROUP. 1994. The POSTGRES user manual. Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA.

VAN HEYNINGEN, M. 1994. The Unified Computer Science Technical Report Index: Lessons in indexing diverse resources. In *Proceedings of the 2nd International WWW Conference* (Chicago, IL, Oct. 17–20). http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/whitehead/whitehead.html.

WONG, E. AND YOUSSEFI, K. 1976. Decomposition—A strategy for query processing. *ACM Trans. Database Syst. 1*, 3 (Sept.), 223–241.

Received: July 1997; revised: March 1998 and June 1998; accepted: June 1998