Languages for Multi-database Interoperability*

Frédéric Gingras
Department of Computer Science
Concordia University, Montreal, Quebec, Canada
gingras@cs.concordia.ca

Laks V.S. Lakshmanan
Department of Computer Science
Concordia University, Montreal, Quebec, Canada
laks@cs.concordia.ca

Iyer N. Subramanian
Department of Computer Science
Concordia University, Montreal, Quebec, Canada
subbu@cs.concordia.ca

Despina Papoulis
Department of Computer Science
Concordia University, Montreal, Quebec, Canada
papouli@cs.concordia.ca

Nematollaah Shiri
Department of Computer Science
Concordia University, Montreal, Quebec, Canada
shiri@cs.concordia.ca

1 Introduction

Database system technology has reached a stage now in which there is a proliferation of independent systems storing and manipulating enormous amount of data. Unfortunately, these systems typically have their own data models, communication processing protocols, query processing systems, concurrency control protocols, consistency management, and other similar aspects of database systems. There is also an increasing need for *Interoperability* among these systems. Though considerable amount of research has been done in the area of database interoperability, most of it has resulted in solutions that are ad-hoc and procedural. We have developed a declarative environment in which multiple heterogeneous databases interoperate by *sharing, interpreting,* and *manipulating* information, in a uniform way.

An important criterion for Interacting with multiple databases is the ability to query them in a manner independent of the discrepancies in their structure and data semantics. In this demo, we exhibit two languages for querying across multiple databases which store semantically similar data using heterogeneous schema, as well as for restructuring the queried data: (1) *SchemaLog* – a language that has its foundations in logic and (2) *SchemaSQL* – a language based on a principled extension to SQL.

*This research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Fonds Pour Formation De Chercheurs Et L'Aide À La Recherche of Quebec.

SIGMOD '97 AZ,USA

Part I: The SchemaLog System

2 About SchemaLog

Our approach is logic based – we believe that a logic based approach for interoperability would bring the advantages of clear foundations, sound formalism, and proof procedures thus providing for a truly declarative environment. Conventional database query languages are based on predicate calculus and are useful for querying the data in a database. But as seen in many applications, interoperability necessitates a functionality to query not only the data in a database but also its schema or meta-data. This calls for a higher-order language which treats "components" of such meta-data as "first class" entities in its semantic structure. In such a framework, queries that manipulate data as well as their schema "in the same breath" could be naturally formulated. In this vein, in [LSS93] we have proposed a language called *SchemaLog* that is syntactically higher-order but semantically first-order, and which facilitates multi-database interoperation. The language has its foundations in logic and has been studied in detail in [LSS96]. It also has syntactic features for explicitly referring to database, relation, and attribute names. In our implementation, users can write programs in *SchemaLog* to interoperate among multiple databases.

3 System Architecture

In this section, we outline the implementation strategy adopted for realizing our system. The system presently facilitates interoperability among multiple INGRES relational databases that contain semantically similar, but schematically dissimilar information.

Two aspects of our language that are of most significance are *(i)* higher-order features to access the schema information from multiple databases, and *(ii)* deduction. Our implementation handles these two aspects separately: The INGRES embedded SQL (ESQL) is used to manipulate the meta-information, and the deductive database CORAL is used for deduction. Figure 1 shows the architecture of our

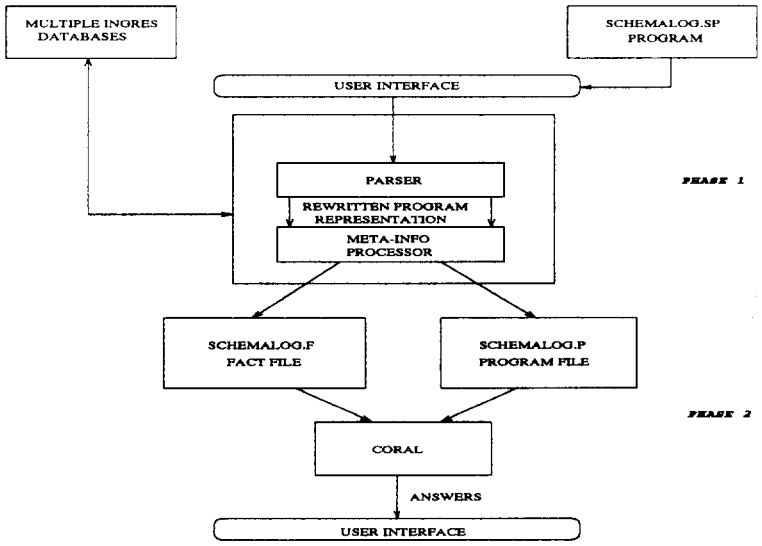

Figure 1: SchemaLog System Architecture

implementation. Since user programs can involve complex interactions between schema manipulation and deduction, our implementation integrates these two functionalities from ESQL and CORAL closely.

In Phase 1 of the implementation, the following activities are performed. The input *SchemaLog* program is rewritten to a first-order form using rewrite rules. The meta-information queried in the program is fetched and stored in appropriate intermediate tables using ESQL. During this process, various optimization strategies are employed to minimize the cost of fetching the meta-information as well as to reduce the amount of information that needs to be stored in the intermediate tables which in turn would affect the cost of (recursive) query processing by CORAL [Pap94]. Phase 2 of the implementation uses CORAL to process the rewritten program along with the meta-information fetched in Phase 1 and generates the answers.

The system sports a pleasant user interface capable, among other things, of a schema browsing facility.

4 Future Work

Various extensions to the existing implementation are currently under investigation. We would like to extend the scope of our system beyond relational databases to other non-traditional information systems such as spreadsheets, file systems, and mail servers. The syntax of SchemaLog can be interpreted against these systems as well. Other planned extensions involve incorporating disparate DBMS like Oracle, Sybase, and O_2 in the current implementation.

Part II: The SchemaSQL System

5 About SchemaSQL

SchemaSQL is a principled extension to SQL recently proposed by Lakshmanan et al. [LSS96b]. SchemaSQL possesses the features necessary for interoperating among a federation of (relational) databases and has the following properties:

Language	ESQL, C
DBMS	INGRES
Deductive System	CORAL
User Interface	UIM/X
Platform	Sun 3/50 workstations Under UNIX

Figure 2: Implementation Particulars of SchemaLog

- It solves the syntactic conflict problem between component databases. In particular, it solves the well-known attribute/value (more generally, data/schema) conflict problem;
- It has an expressive power which is independent of the specific schema with which a database is structured;
- It allows for restructuring, glossing over the difference between what, at a given time, is data or schema;
- It allows to perform complex block aggregations. In particular, its higher-order expressive power, achieved in a simple framework, allows aggregations and restructuring (including slice and dice) in the spirit of OLAP style computations. It can also express the well-known CUBE operator [GBLP96].

This implementation will demonstrate the features of the SchemaSQL language, how it can query across heterogeneous schemas, manipulate meta-data, restructure information at the user's will, and perform complex aggregations.

6 SchemaSQL System Architecture

In this section, we outline the implementation strategy adopted for realizing our system. In our architecture (see Figure 3), the user's SchemaSQL queries are submitted to the SchemaSQL server, which determines a series of local SQL queries to be dispatched to the appropriate local databases.

The answer to those queries are returned to the SchemaSQL server. The server now has the information necessary to answer the initial query. It "simply" has to use its own resident SQL engine to properly query and restructure the information it obtained from the local sites. The SchemaSQL server also maintains system tables containing information about the local databases (e.g. relations in each databases, attributes in each databases, etc.) to reduce processing costs.

The system is built using MS Access as DBMS for all databases in the federation. The SchemaSQL server also uses MS Access as its resident SQL engine. The code for the SchemaSQL server is written using Visual Basic and Visual C++ and uses OLE2 to interact with MS Access. The user interface is an integral part of the server's code. The whole system runs on IBM PCs under Windows 95.

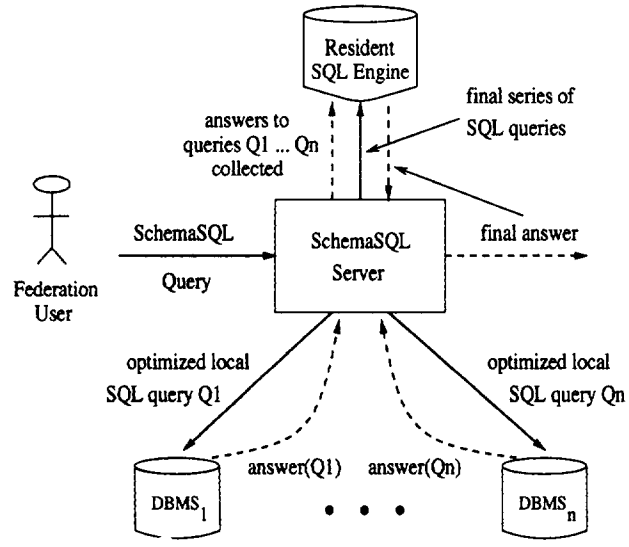

Figure 3: SchemaSQL System Architecture

The processing of a query is divided into two main phases. First, the query is posed by the user using the Graphical Querying Interface. The query is then parsed and validity checks are made. Local SQL queries are prepared to fetch all useful instantiations of the variables. At this stage, optimization techniques are used to minimize the quantity of information transmitted over the network and the number of connections to the local databases. The prepared queries are then submitted to the local sites and the results are returned.

The second step is the rewriting of the original SchemaSQL query as a series of SQL queries over the tables of results returned from the local queries. This step is where restructuring takes place. The user also has the possibility of creating views for those restructurings that would be repetitively useful. The server would then store, along the view definition, the processing steps to be taken (so as to minimize processing the next time around).

Our SchemaSQL server sports a semi-graphical interface. A future version will have a completely graphical interface that will let users query using the server without the need for them to know much about the syntax of the language.

Language	VBA, VC++
Tools	MS Access
Platform	IBM PC
	Under Microsoft Windows 95 running OLE 2.0

Figure 4: Implementation Particulars of SchemaSQL

References

[GBLP96] Gray, J., Bosworth, A., Layman, A. and Pirahesh H. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Proc. of the 12th Intl. Conference on Data Engineering

[LSS93] Lakshmanan, L.V.S., Sadri, F., and Subramanian, I. N. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proc. 3rd Intl. Conference on Deductive and Object-Oriented Databases (DOOD '93)*, pages 81–100. Springer-Verlag, LNCS-760, December 1993.

[LSS96] Lakshmanan, L.V.S., Sadri, F., and Subramanian, I. N. Logic and algebraic languages for interoperability in multidatabase systems. Technical report, Concordia University, Montreal, Feb 1996. Accepted to the Journal of Logic Programming.

[LSS96b] Lakshmanan, L.V.S., Sadri, F. and Subramanian, I. N. SchemaSQL – A Language for Querying and Restructuring Multidatabase Systems. Proc. IEEE Int. Conf. on Very Large Databases (VLDB'96), Bombay, India, September 1996.

[Pap94] Papoulis, Despina. Realizing SchemaLog. Tech. report, Dept. of CS, Concordia Univ., Montreal, Canada, 1994.