LINDA: A System for Loosely Integrated Databases

Antoni Wolski

Laboratory for Information Processing
Technical Research Centre of Finland (VTT)
Lehtisaarentie 2A, 00340 Helsinki, Finland

ABSTRACT

LINDA is an experimental system for database access in heterogeneous environments. The goal is to achieve maximum site autonomy and as much database access homogenization as possible. We start by discussing the problems to be solved and justifying design priorities. Major heterogeneity problems are sketched and the applied techniques are briefly described. We give an overview of the implementation of the system. We conclude with a discussion of the system's applicability.

1. INTRODUCTION

Several heterogeneous distributed database management systems have been implemented. Among them are experimental systems like XNDM [Kimbleton&al.79], MULTIBASE [Landers&Rosenberg82], SIRIUS-DELTA [Litwin82], [Ferrier&-Stangret83] and PROTEUS [Stocker&al.84], and a few systems built with operational usage in mind, like ADDS [Breitbart&al.86] and Aida/Mermaid [Templeton&al.86], [Templeton&al.87a].

The main stress in research has been on executing global queries in a heterogeneous database environment. This involved intermediate (global, pivot) languages, language conversions and schema conversions.

Most of the systems mentioned, if moved to operational environments, would be cumbersome. One reason for this is the way the global schema is handled. A global schema is necessary for execution of distributed global queries. However, the systems do not address the problem of automatic schema integration and updating. Consequently, the global schema is to be hand-built (either at each site or at a centralized data dictionary site) following any change at any participating node. This may be operationally unacceptable in a dynamic, diversified environment and, additionally, it violates the site autonomy which is of high value in such environments.

On the other hand, the commercial technology is not addressing the problem of heterogeneous database access properly. Currently available distributed database products or database server products, e.g. SQL*STAR (Oracle Corp.), INGRES/STAR (RTI), SQL Server (Sybase Inc.) and VAX Data Distributor (DEC), do not support heterogeneity at the DBMS level. They are built to interoperate with the DBMS software of the same make at all nodes of a distributed database system. This is reflected in the design of the corresponding database protocols, and the fact publishing of the protocols does not change this characteristic.

Demands for access to heterogeneous databases may be satisfied, in such systems, by using *gateways*. Typically, a gateway is a software subsystem enabling to accommodate a "foreign" DBMS as a database server in an otherwise homogeneous distributed system.

Vendors of distributed DBMS currently offer such gateways. However, because of heterogeneity problems, functionality of the foreign nodes is often limited. Additionally, as the global protocols originate from a specific distributed DBMS product, the solution is vendor dependent.

We shall focus our attention on *pre-existing* databases now. This notion not only stresses the chronology of events – that the databases had been created before we wanted to integrate them – but it includes the assumption that the databases were created in an uncoordinated way, without a common design history. The objective is, then, to *integrate* the databases – in contrast to a process of designing and creating a distributed database.

However, semantically correct integration of participating schemata may be impossible without an overall conceptual model which may be unavailable in cases of pre-existing databases.

At this point, a *federated database architecture* [Heimbigner&McLeod85] seems to be much more viable. A federated architecture does not insist on creation of a global schema. *Export* and *Import Schemas* are means for constructing required user (external) views in this case. Although no central authority is required in federated databases, a great deal of cooperation is necessary among the participating systems.

A slightly different approach is taken in multidatabases as defined in [Litwin&Abdellatif86]. Here, no special requirement for cooperation are stated. On the other hand a user is presented with a *multidatabase language* enabling the user to perform operations related to different databases at the same time, and to define interdatabase dependencies. See [Litwin&Zeroual88] for the comparison of the two above approaches.

We are defining a set of capabilities that is more limited than any of the above. Shortly, it includes a *homogenized* (i.e. unified by appearance but not integrated) view on participating schemata, and a common database language. A system having these characteristics, together with full site autonomy, will be

called here a *Loosely Integrated Database System* (hence: LINDA).

LINDA is an experimental system of the above type. It was developed during the 4-year FINPRIT research programme funded by the Technology Development Centre of Finland (TEKES), and completed in the early 1988.

This paper describes the implementation of LINDA. The following section deals with the problem description and LINDA objectives. Section 3 covers main principles of the LINDA design and Section 4 is solely devoted to database system heterogeneity and techniques to handle it. Section 5 contains a more detailed presentation of the implementation. In the Summary we discuss our experience and possible enhancements of the system.

2. LINDA OBJECTIVES

2.1. Organization and users

The target environment for LINDA is a medium-size or large business organization where databases had proliferated in a way typical for *decentralized computer systems* [Gray86].

Database applications are envisaged to be run at workstations. A common programmatic interface to the databases is required to be available at the workstations. The interface should embody a single set of data types, a single query language (SQL) and a unified data dictionary service. An interactive interface for ad-hoc usage should also be available.

2.2. Levels of database system heterogeneity

Decentralized systems are inherently heterogeneous. Databases in such an environment bring a new dimension to the heterogeneity. We shall discuss some aspects of database system heterogeneity here.

In our understanding, the databases are heterogeneous if they differ in any of the following:

- data model
- database language
- DBMS

and possibly (especially true for pre-existing databases):

- conceptual framework used for defining the database schema.

A truly successful solution to the problem has to deal with all the above aspects in some way. In LINDA, we decided to focus our attention on a subclass of the problems as described below.

First, we shall distinguish between *semantic* and *syntactic* levels of heterogeneity.

The semantic level concerns the meaning of database objects in terms of a common *universe of discourse* (UOD). We encounter this kind of heterogeneity if the databases do not share the same conceptual framework. A simple example is the same table or column name having different meanings in different databases (homonyms) or different names having the same meaning (synonyms).

Solving of semantic problems may be done in one of the following way:

A. Integrate schema. This is required if distribution transparency is to be provided, and a global schema is to be built. Schema integration is difficult if a common conceptual model can not be found for participating databases.

B. Leave it to be sorted out by the users. In order to make this easier, any local system should be able to provide as much information on the local schema and local concepts as possible.

C. Apply a multidatabase language which enables the user to specify the inter-object mappings and which may be utilized for execution of global queries.

As schema integration is not feasible here, the approach B, with the provision to extend it to C in the future, was chosen in LINDA.

All the other problems may be considered to be at the syntactic level because they have to do with representation of commands, data and metadata. In the first phase we have concentrated on relational systems only. The related heterogeneity problems and the corresponding LINDA techniques are presented in Section. 4.

There are other heterogeneity issues resulting from different hardware, operating systems and such. They are addressed by general-purpose interworking techniques (like *Open Systems Interconnection*) and are not within the scope of this paper.

2.3. Site autonomy

The requirements for site autonomy may be broken down in the following way. We may say that the following characteristics of a site should be unaffected by the process of database integration:

- database definition and the freedom to change the definition,
- access authorization policy and the way the authorization is performed,
- user identification and authentication techniques and security of authentication (e.g. protection of passwords),
- resource usage accounting,
- sovereignty of action – a site must not be forced to perform any action on behalf of another site or central authority,
- locality of action – no administrative duties need to be performed at a site for the sake of the "outer" system.

All of the above are required in LINDA.

An obvious characteristic of site autonomy is that a local DBMS can not be altered in any way for the sake of a system like LINDA (this would not be possible with most commercial products anyway).

3. THE PRINCIPLES OF LINDA DESIGN

3.1. LINDA units

The LINDA system is composed of LINDA *units* which operate at computer sites. A unit is characterized by the fact that it is associated with *one* (and only one) DBMS. All LINDA units have unique names within the LINDA environment.

There are two types of units: *Client Units* and *Server Units* (Fig. 1).

Users and applications (jointly referred to as "users" in the following) interface with *Client Units*.

A Client Unit provides the user with the service for accessing remote or local databases. The unit is associated with a local DBMS which is a "home" DBMS for the user. The user may insert query results into a database managed by this DBMS. This database is called the *Client Database*.

At the Client Unit, the query passed through a dynamic, SQL-based interface, is parsed, syntactically validated, transformed to a transfer syntax format and sent to the serving node for compilation and execution. The received results are transformed into an appropriate format and passed to the requesting program, or stored in the Client Database. There is always a Client Unit at a user's local site.

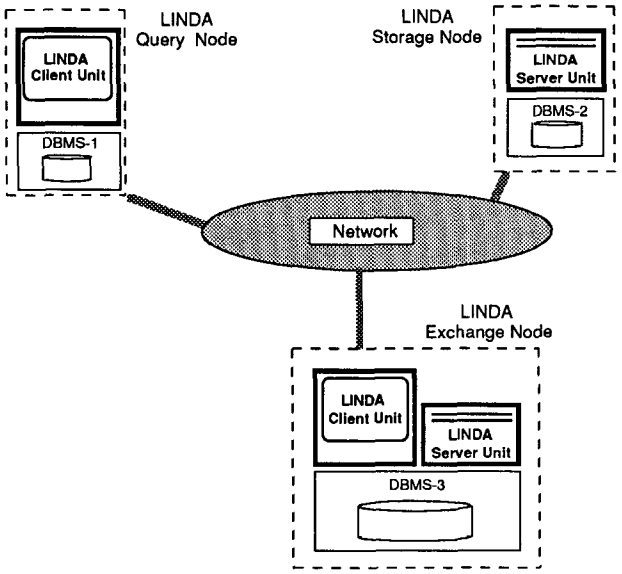


Fig. 1. The types of components in the LINDA system

A *Server Unit* is designated to satisfy database service requests originated at local or remote Client Units. One or more *Server Databases*, managed by a certain DBMS, are associated with each Server Unit. The Server Units are accessed remotely by means of a database protocol or locally by means of the related function call based interface.

Units of both types may be combined at a site over the same DBMS. The units operate independently of each other and the corresponding Client and Server Databases may be either disjoint or intersecting.

3.2. Levels of transparency

LINDA supports the following two levels of transparency:

Location transparency: a user is aware of the existence of distinct databases but he/she need not be aware of where they reside (at a local or remote site and which one).

DBMS transparency: a user is able to manipulate any database in the same way regardless of the DBMS managing the database.

As the schema is not integrated, a *distribution transparency*, meaning hiding of the fact that there are distinct databases, is not provided.

3.3. Database operations

In the first LINDA prototype, a user may:

a) perform a single site data retrieval,

b) store the result of a single site retrieval in the Client Database,

c) perform a single site data dictionary retrieval.

Capability b) enables the user to assemble integrated snapshots of various databases under the user's control.

LINDA is designed in a such a way that multiple site queries can be implemented easily.

The system does not support server database updates at the moment but single site updates can be easily added.

Multiple site updates will be required in some environments. This would require application of global concurrency control and error recovery mechanisms in the system. We are going to work on the problem in the near future.

3.4. Application of the RDA protocol

In order to achieve expandability of a network in a heterogeneous environment, the *open systems approach* is needed, i.e. every node should absolutely comply with the agreed global protocols applied to various levels of data exchange abstraction.

In LINDA, we are using the RDA (Remote Database Access) protocol [ISO/RDA/88], [ISO/RDA/87] which is under development at ISO*.

RDA enables one to access remote databases on a point-to-point basis. The RDA protocol and the service definitions are presently based on the SQL language as defined in the standard [ISO/SQL/87]. The RDA service primitives reflect the semantics of the SQL host language embedded interface. The protocol transfers the commands and data using a special, efficiency-tuned transfer syntax. The database commands are transferred in a parsed, tree-structured form.

RDA goes slightly beyond SQL in that it supports an asynchronous database interface and allows for transfer of multi-row results.

The LINDA nodes communicate with each others *solely* by means of the RDA protocol.

A consequence of the above principle is that a LINDA Server Unit is an implementation of a standard RDA server. Also, any standard RDA server may be used in place of a LINDA Server Unit. This becomes significant once DBMS vendors incorporate the RDA server capability into their products (as shown for Node 3 in Fig.2.). As this has not happened yet, the Server Units had to be developed for the LINDA system.

* RDA has the status of Draft Proposal (DP) at the time of writing. It is expected to be finally accepted in 1990.

4. COPING WITH SYNTACTIC HETEROGENEITY

4.1. Database language syntax

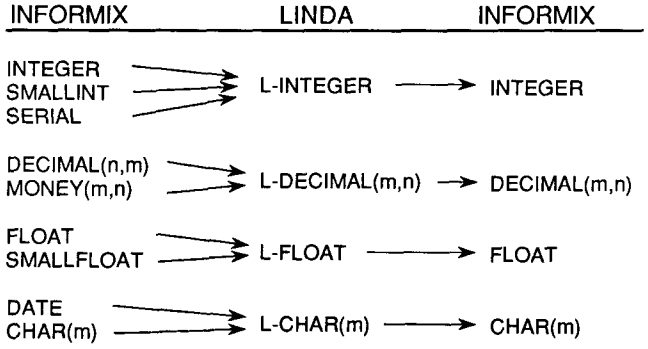
There is a considerable amount of syntactic difference among relational languages. Even different products based on "standard" SQL are (at the moment), practically speaking, totally incompatible with each other, with respect to their programmatic interfaces. The following list highlights the differences:

- notion of a database – not all DBMS's allow for creation and usage of distinct databases identifiable by a name;
- data types – there is a plethora of data types and the number of them varies from 4 to about 11;
- data representation and resulting precision;
- object naming conventions – ranging from strict, COBOL-like, to very liberated ones.

In LINDA, the above problems are solved according to the open systems approach.

The global protocol makes provision for an additional database name specification, and this is reflected in the global database interface.

Diverse data types are mapped to global, RDA-standard data types which originate from the SQL standard [ISO/SQL87]. For each specific DBMS a mapping table is made, like the following one prepared for INFORMIX:



The left part of the table is used when data are retrieved from a Server or Client database. The right part is used when the results are being stored into a Client database. The round trip mapping ambiguity is unavoidable as there are fewer global types than specific types.

Data type mapping between two different specific systems is achieved using LINDA types as a "pivot".

For each LINDA data type, the data transferred between Server and Client units are converted to the corresponding RDA transfer syntax. The advantage of the RDA syntax is that it deals with variable length representations so that no precision is lost in transfer. If the precision is lost in the Client Unit because of its limitations, the appropriate warning is generated.

The data type and data representation mappings are definable by means of internal tables used by the Database Access Modules — the components of LINDA that are responsible for specific to global data transformations.

4.2. Query processing

If commercial RDA servers were available, processing of a query would be reduced to just scanning and parsing the query, possibly validating it and, subsequently, converting it into the RDA transfer syntax – all at a Client Unit.

Because LINDA Server Units are built on top of existing DBMS's, the difficult part is to feed the query from the Server Unit to the specific DBMS. The requirement is to be able to execute an arbitrary query dynamically.

Commercial products support some of the following interfaces:

A. Interactive terminal query interface. This is a clumsy solution and has obvious problems: precision of results may be lost while converting to encoded form, unnecessary overhead is induced by the encoding and additional overhead is caused by handling of files that are necessary to accommodate intermediate results.

B. "Singleton"(i.e. one-row result) dynamic SQL. Useless for arbitrary queries.

C. Generalized dynamic SQL with memory-based multi-row result allocation. Best for this purpose but only few systems have it.

D. Cursor-based, pre-processed SQL. This can be used in the following way: for each query command received at a Server Unit, generate a source program (e.g. in C language) containing the corresponding cursor definition and fetch commands; pre-process and compile the program; activate it as a separate process and pipe the results to the Server process.

In LINDA, the interface types A, C and D can be chosen for connecting of a specific DBMS. The code corresponding to the selected method is then installed in the Database Access Module in question.

Another difficult part of the query language homogenization are slight semantic variations among different implementations. LINDA, generally, does not try to sort out such differences. The examples are:

- existence and treatment of null values (e.g. different results may be obtained while calculating aggregates allowing null values);
- different treatment of duplicate rows in result tables
- different extent of integrity constraints supported (keys, referential integrity and user-defined integrity constraints) that may result in a rejection of a query in one system and accepting it in another.

4.3. Authorization, identification and authentication

Any LINDA user who accesses a site is, from the point of view of the site, identified, authorized and authenticated in the same way as any local user. In this respect the scheme is similar to that of Mermaid [Templeton&al.87b]. In addition to this, the authentication process may be performed at two levels: the node level and, optionally, the unit (database) level.

At the node level, the user name (user ID) and the password are supplied by the Client Unit (the user had stored them there). The encryption scheme is simpler then that of Mermaid which uses a simulated Enigma machine. In LINDA, the node pass-

words are encrypted using the system supplied encryption functions and the user's special LINDA password as an encryption key. This way, only the user (or the Client Unit on behalf of the user) may decrypt them. The passwords are transmitted decrypted, assuming a secure network.

At the unit level, the Server Unit site may request the use of a second user ID and password to access the database. In this case, the ID is again supplied by the Client Unit but the password has to be entered on-line by the user (for maximum security).

Thus, the identification and authentication process, together with the establishment of a data transfer association, is invoked by the following global interface primitive:

dbopen (*unit-name*[, *db-name*][, *db-password*])

where the following parameters are optional: *db-name* (for facilitating distinct databases within a unit) and *db-password* (for unit/database level authentication).

We assume that most sites will not use the unit level authentication. This will leave the user with the need to log-in to his or her "own" workstation only and then enter once the LINDA password.

Following the identification and authentication phase, any LINDA activity taking place at the serving node performed on behalf of the user, is accountable to the user and is subject to all authorization limitations imposed locally.

4.4. LINDA data dictionary

The LINDA Data Dictionary concepts serve the following goals:

- location transparency
- DBMS transparency with respect to identification and authentication,
- DBMS transparency with respect to retrieval of metadata.

The first two goals are served by the *Client Dictionary* which is a dedicated data structure in the Client Database, containing information about the LINDA environment accessible by a user. This includes Server Unit names and their network addresses. It also contains the user's identification and authentication information (ID's and passwords) to be used while accessing remote nodes and Server Databases. This information is used by the Client Unit on behalf of the user. The information in the Client Dictionary is supposed to be maintained by a local user or administrator at a Query Node.

The goal of DBMS transparent retrieval of metadata is supported by means of *Server Dictionaries* which are virtual data structures located at Server Units and accessible via regular global data manipulation commands. The metadata are used by Client Units for query validation and may also be used by applications (like a graphical data dictionary browser that was implemented in LINDA).

The Server Dictionary describes the Server Database in terms of a full SQL schema. Essentially, it is a set of relational views defined for each Server Database to map the global dictionary structure to the product specific system catalog. Any changes made locally to a Server Database schema will be automatically reflected in the global view of the dictionary.

The global structure of LINDA Server Dictionary is based on the specifications found in SQL2 [ISO-ANSI/SQL2/88]* . As some of the metadata covered by the LINDA dictionary is not maintained and enforced in current systems (notably the referential integrity information), additional dictionary tables may be added to the Server Database. These have to be maintained manually in order to keep up with the changes in database definition. The additional tables, however, are not mandatory for correct functioning of the global database interface, and so the site autonomy is preserved.

The access to the dictionary is performed on behalf of a user and only the data pertaining to the user's view of the database are accessible.

4.5. Error message handling

Formats of error messages generated by specific systems seem to be the least consistent of all, giving a potential for great confusion.

In LINDA, error messages originating at Server Units are dealt with in a special way. There are global classes of error conditions. Many errors can be mapped directly to global error types enabling orderly error processing at the Client Unit. Other, unanticipated errors are classified as "open" and the entire error messages are transmitted to the Client Unit.

5. NOTES ON IMPLEMENTATION

5.1. General architecture

LINDA software is built in the form of layers. The top layer, present in the Client Unit only, is composed of LINDA applications. An example is GRAFER – a visual query facility for Sun workstations that was developed in the project [Tikkanen88].

The second layer deals with global processing (i.e. processing of global formats) and telecommunications, and is very portable. This layer offers a global programmatic database interface at Client Units.

The third layer is responsible for interfacing to local database systems. The modules here are tailored to specific products and are portable only within a given DBMS product.

The major components of the LINDA software are shown in Fig. 2. For simplicity, an arrangement of a Query Node and a Storage Node only is presented. In the following we shall describe the system architecture in terms of two main program interfaces present in the system.

5.2. Global Query Interface (GQI)

The *Global Query Interface* is a callable function library for advanced database access. It implements the SQL data manipulation commands and may be characterized in the following way:

- A unified interface to all the databases within the LINDA environment.

---

* A new SQL standard under preparation. The standard is expected to be finalized about 1991. SQL2 includes, among others, standard schema tables.

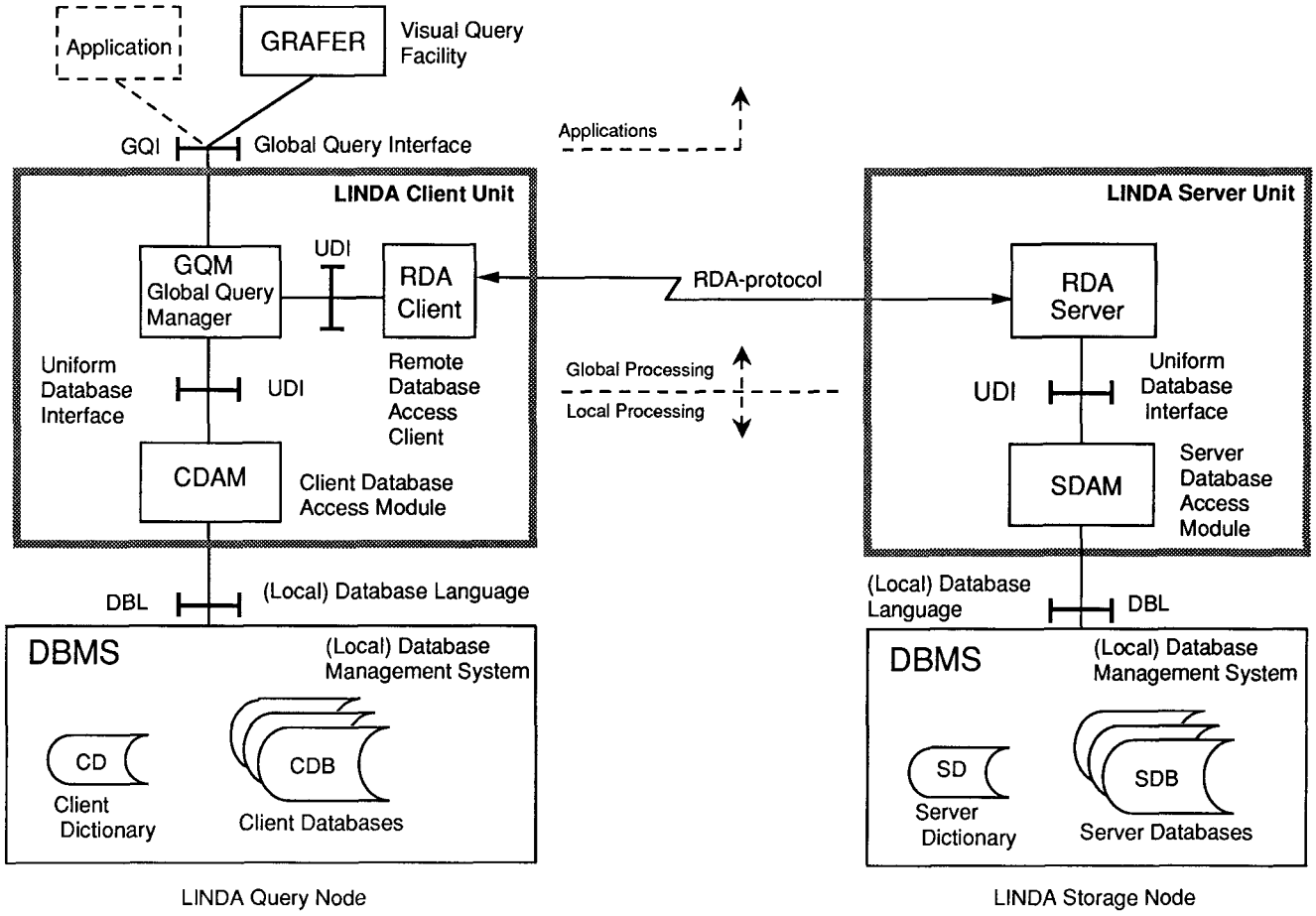


Fig.2. Simplified software architecture of LINDA.

- A dynamic, subroutine call based interface.
- The semantics of the data manipulation calls follows the semantics of the cursor-based embedded SQL.
- SQL query expressions are submitted as character strings.
- Database opening and closing commands are added. LINDA units and databases are seen as logical entities at this level
- Global LINDA data types (which correspond to standard RDA types) are used regardless of which database is accessed.

Global Query Manager (GQM) is responsible for routing the data manipulation commands and query results in the system. Using the Client Dictionary, it appends network and authentication information to the commands that are meant for remote nodes.

GQM performs scanning and parsing of the incoming queries and translates them into the RDA-like structures (parse trees) used by the next interface.

GQM also performs the translation of the "depth of retrieval". This means that it translates the row-oriented GQI cursor commands into the multi-row UDI (and consequently RDA-SQL) commands. It does this by buffering the query results so that they are passed at a row-at-a-time basis to an application using GQI. This step is necessary in order to make efficient use of the multi-row RDA protocol minimizing number of messages exchanged for a query.

5.3. Unified Database Interface (UDI)

The *Unified Database Interface* is a callable function library for implementing efficient database access, local or remote, within LINDA. The characteristics of the interface are as follows:

- The multi-row data manipulation. The semantics is similar to that of a cursor-based SQL but it enables to obtain result tables of a specified number of rows. The syntax follows the RDA-SQL syntax.

- The level of abstraction of the interface is lower that of GQI as the network nodes have to be dealt with explicitly, and the network error conditions have to be passed as well (to the GQM).
- The interface is composed of two intersecting sets of functions. A smaller set (retrieval only) is used by the RDA Server in the Server Unit, and a larger set of functions supporting inserting of the query results into a Client Database, is used by the GQM in the Client Unit.

The *Database Access Modules* perform the translation between the LINDA UDI calls and the corresponding commands of a local DBMS. The data type conversion is dealt with as well. Although some generalized heterogeneity information is maintained in program tables (installation parameters), the modules have to be somewhat tailored to a specific DBMS as well.

To submit the query to a local DBMS for compilation and execution, one of the techniques outlined in section 4.2 is used.

The *Client Database Access Module* (CDAM) is used in the Client Unit and it performs, in addition to database retrieval, inserting of query results into the Client Database.

The *Server Database Access Module* (SDAM) has the capability of database retrieval only and it is used in a Server Unit.

The *RDA Client* is responsible for translating between the UDI commands and the RDA protocol transfer syntax. It maintains the association with an RDA Server for the time of the remote access session. It is invoked at the beginning of the database session and terminated at closing of the session.

The *RDA Server* provides the RDA service accessible via network. While active, it is expecting requests for associations from the remote Client Units. Once the association is established, it translates between the RDA protocol and the UDI commands, in the Server Unit. It is implemented in a way enabling it to serve many associations at the same time.

5.4 The prototype

The LINDA prototype was built at the Laboratory for Information Processing of Technical Research Centre of Finland (VTT/TIK), in Helsinki. The prototype is confined to the UNIX environment with a local area network (Ethernet). A Storage Node is implemented in a MicroVAX II computer running EMPRESS DBMS, and Sun workstations act as Query and Exchange Nodes. INFORMIX DBMS is used at the Sun computers. The RDA protocol is implemented on top of the BSD UNIX socket service.

6. CONCLUSIONS

The LINDA system as presented here and implemented represents the minimum functionality required if heterogeneous, pre-existing databases are to be utilized productively in an enterprise. With LINDA, the users are able to access databases residing at different sites in a consistent way. At this time, a single query is limited to one database only but concurrent query sessions are possible. The sites preserve their autonomy and consider any user a local one.

The single site update capability is most obvious possible extension. It is easy to implement, simply by adding new tokens to interfaces and the protocol.

A multidatabase language could also be considered. Addition of this capability would affect the upper layers of the Client Unit.

We believe that distribution transparency (both for retrieval and update) is not justifiable in the envisaged environment. It would lead to serious loss of site autonomy, database availability, processing efficiency and would be a difficult and costly undertaking. The overall loss would outweigh possible benefits easily.

ACKNOWLEDGMENTS

Writing of this paper was possible thanks to contributions by the LINDA project members: Jukka Aakula (data communications), Maaret Karttunen (global processing and interfaces), Leena Sivola (heterogeneity and local interfaces), Matti Tikkanen (user interface) and Heikki Tiihonen (project management).

REFERENCES

[Breitbart&al.86]
Yuri Breitbart, Peter L. Olson, Glenn R. Thompson, "Database Integration in a Distributed Database System". *Proc. IEEE Int. Conf. on Data Engineering*, Feb. 5-7, 1986.

[Ferrier&Stangret83]
A. Ferrier, C. Stangret, "Heterogeneity in the Distributed Database Management System SIRIUS-DELTA". *Proc. 8th VLDB Conf.*, Mexico City, 1983.

[Gligor&Popescu-Zel.86]
Virgil Gligor, Radu Popescu-Zeletin, "Transaction Management in Distributed Heterogeneous Database Management Systems". *Information Systems*, Vol. 11, No. 4 (1986), pp. 287-297.

[Gray86]
Jim N. Gray, "An Approach to Decentralized Computer Systems". *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 6 (June 86), pp. 684-692.

[Heimbigner,McLeod85]
D. Heimbigner, D. McLeod, "A Federated Architecture for Information Management". *ACM Trans. on Office Inf. Sys.*, Vol. 3, No. 3 (1985), pp. 253-278.

[ISO/SQL/87]]
ISO 9075, "Information processing systems - Database language SQL". International standard, first edition, 1987. Ref. No. ISO 9075 : 1987 (E). Also available as: ANSI X3.135-1986, Database Language SQL.

[ISO/RDA/88]
ISO DP 9579, "Information Processing Systems - Remote Database Access". Draft Proposal (revised), July 1988. Also available as ANSI X3H2-RDA-88-56.

[ISO/RDA/87]
"ISO Remote Database Access, Tutorial". ISO/TC97/SC21/N1927, July 1987.

[ISO-ANSI/SQL2/88]
"ISO-ANSI Database Language SQL2" (working draft). ISO/IEC JTC1/SC21/WG3 DBL SYD-2, July 1988. Also available as ANSI X3H2-88-259.

[Kimbleton&al.79]
S. R. Kimbleton, P. S. C. Wang, E. Fong, "XNDM: An Experimental Network Data Manager". *Proc. Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, Aug. 1979. pp. 3-17.

[Landers&Rosenberg82]
Terry Landers, Ronni L. Rosenberg, "An Overview of MULTIBASE". In: *Distributed Data Bases*, H.-J. Schneider, ed. (Proc. 2nd International Symposium on Distributed Data Bases, Berlin, Sep. 1982). North-Holland, 1982.

[Litwin82]
Witold Litwin et al., "SIRIUS System for Distributed Data Management". In: *Distributed Data Bases*, H.-J. Schneider, ed. (Proc. 2nd International Symposium on Distributed Data Bases, Berlin, Sep. 1982). North-Holland, 1982.

[Litwin&Abdellatif86]
Witold Litwin, Abdelaziz Abdellatif, "Multidatabase Interoperability". *IEEE Computer*, Vol. 19, No. 12 (December 1986), pp. 10-18.

[Litwin&Zeroual88]
W. Litwin, A. Zeroual, "Advances in Multidatabase Systems". In: *Research into Networks and Distributed Applications* (Proc. EUTECO '88, Conf.), R. Speth (ed.), North-Holland 1988, pp. 1137-1151.

[Stocker&al.84]
P. M. Stocker et al., "PROTEUS: A Heterogeneous Distributed Database Project". In: *Databases - Role and Structure: An Advanced Course*, P.M. Stocker, P.M. Gray, M. P. Atkinson (eds). Cambridge University Press, 1984.

[Templeton&al.86]
Marjorie Templeton, David Brill, Arbee Chen, Son Dao, Eric Lund , "Mermaid - Experiences with Network Operation". *Proc. IEEE Int. Conf. on Data Engineering*, Feb. 5-7, 1986.

[Templeton&al.87a]
M. Templeton, D. Brill, A. Chen, S. Dao, E. Lund, R. MacGregor, P. Ward, "Mermaid – A Front-end to Distributed Heterogeneous Databases". *Proc. IEEE*, May 1987.

[Templeton&al.87b]
Marjorie Templeton, Eric Lund, Pat Ward, "Pragmatics of Access Control in Mermaid". *Quarterly Bull. IEEE Tech. Committee on Database Engineering*, Vol. 10, No. 3 (September 1987), pp. 33-38. Also in: W. Kim et al. (eds.): *Database Engineering*, Vol. 6, IEEE 1987, pp. 157-162.

[Tikkanen88]
Matti Tikkanen, "LINDA User Interface". In: J. Aakula et al., "LINDA - Loosely Integrated Databases", Technical Report, VTT, Lab. Inf. Proc., Helsinki 1988.