Data integration methodology for an office environment

L Marinos, M P Papazoglou* and D Christodoulakis**

The typical requirements for integrating disparate relational database systems residing at personal workstations in an office environment are examined and a concise system architecture is proposed. The architecture is based on a set of logical front-end interfaces that facilitate a loose coupling of the databases. These logical extensions provide for effective interconnection of the individual database management systems, and guarantee a uniform and integrated access to remote databases in the network. General directions are given on how to achieve logical schema integration in terms of a global schema which unifies schema information dispersed across the databases in the network. The proposed system is oriented towards databases requiring moderate resources which makes schema integration cost effective. Personal or small group databases in an office environment are likely to benefit most from the proposed approach.

Keywords: systems integration, database systems, office environment, architecture

As organizations and businesses become more automated they are dramatically increasing their reliance on data processing facilities to store and manipulate data that are critical to their everyday operations. This leads to an increased demand for data availability and reusability. As organizations evolve and merge, the need to combine information stored in pre-existing autonomous data repositories grows. On the other hand, the increasing power of micro/mini computers leads to the fact that a greater number of dispersed database sites can afford local workstations which have sufficient power and storage to meet many of their local requirements without resource to a central mainframe system. Reliable data transfer between individual workstations can be achieved either locally over local area networks (LAN), or remotely over wide area networks (WAN). These facilities have provided a firm basis for the development of distributed database management systems in an office environment, and triggered research into ways of establishing them.

We view an automated office as comprising a set of databases residing at diverse personal workstations. There is no doubt that such workstations are useful for a wide spectrum of applications that are logically decentralized (autonomous). Obviously, any cooperative activity that presumes a loose integration of specialized but related application domains administered by different groups of office employees is not supported.

There are several good reasons why the data relevant to a specific enterprise is often stored and maintained by systems that are either completely unintegrated, or badly integrated, the main reason being that many databases are normally developed to provide support for a very specific class of applications. The choice of the data model and the actual conceptual schema for the database is strongly influenced by both these applications and the state of current database management system (DBMS) technology. An evolving enterprise may gradually obtain many variants of a particular database system. Therefore, applications must be developed so as to be in the position to cooperate with already existing databases, while they have to conform to the access provided by the existing databases. Clearly, such new applications cannot rely on conventional data management techniques to achieve integration and semantic congruence with the existing data repositories; they simply demand advanced forms of data management. To support such advanced features we require the provision of mechanisms and concepts which guarantee that each application

The German Institute for Computer Science (GMD), F2 - Institut fur Systemtechnik, Schloss Birlinghoven, D-5205 St Augustin 1, FRG
* Department of Computer Science, Australian National University, GPO Box 4, Canberra, ACT 2601, Australia
** Department of Computer Science, University of Patras, Patras 26500, Greece

0267-6192/91/030143 -- 09

'sees' a single database system where, in fact, data is maintained by several databases.

Owing to the growing popularity of relational systems the office environment of the future will most certainly comprise a potential mix of workstations connected by means of a common communication network, supporting a variety of databases with relational dialects. Such a composite system will have different groups of users, each of them being familiar with and preferring to use a different query language. Relative preferences can conveniently be satisfied by developing an appropriate transaction methodology for the diverse relational query constructs in the system. The provision of the above features is of immense commercial significance: institutions which put their data into strategic use as well as companies that provide others with the knowledge to do so will have clear advantages on the market.

The environment proposed here can be thought of as a 'virtually' integrated office system comprising many, and possibly diverse, personal workstations, each supporting some relational DBMS. It is assumed that the workstations are connected by means of a common LAN network, and databases in the network are developed autonomously. Furthermore, databases may contain semantically associated and possibly overlapping information. Database integration in this environment is viewed as a loose form of logical integration, since the intention is not to bind existing databases into a fixed global database, but rather to provide the users with uniform access to a number of databases, some of which may contain related information. The term *logical integration* is reserved to indicate the need for organizing information without having to bother about their physical location. The result of this loose integration is an integrated office system, an automated system that combines distributed data management and processing facilities for independently developed local applications. The prime purpose of such an integrated environment is to facilitate both the intensive communication required between employees within a department, as well as the sparse communication between employees in different departments[1].

Some of the concepts presented here were inspired by and present certain similarities to relevant concepts developed by the authors for an integrated environment of heterogeneous databases[2]. The data processing requirements of an integrated office environment and those of heterogeneous-homogeneous database environments have a lot in common. In particular, we are convinced that an office environment* could greatly benefit from such an approach. Our aim is to elaborate on the architectural concepts of a system that supports cooperative work and facilitates the office workers to gather information in a well defined and disciplined manner.

SYSTEM ARCHITECTURE

A variety of serious problems arise in an integrated office environment when a large organization decides to integrate disparate databases or wishes to upgrade its structure by introducing a new relational system to which all existing databases should adhere. A suitable relational query language inter-translator can be used in both cases to give transparent access to the corporate data. This assumes the use of a global schema (for modelling the composite application) and a universal query language powerful enough to manage the internal complexities of a relational database system.

The fundamental observation made here is that all the DBMSs that have been developed for workstations claim to be relational, although most of them are not truly relational[3], either by not supporting integrity mechanisms or by not being transparent as far as indexing is concerned. In the present study, we assume the existence of a common standard relational query language having the capabilities of SQL[4] to serve as the universal or *global query language.*

The data intensive needs of an integrated office environment pose a series of requirements which have an impact on both the architecture as well as the functionality of the composite database system. These requirements can be summarized as follows:

- The entire office environment, can be viewed as a network of geographically dispersed database systems. Actually, the proposed system consists of a set of workstations connected to a LAN, with one particular workstation acting as a server (see Figure 1). Each workstation contains a database, which is private to its user (Local sites in Figure 1). Nevertheless, the system as a whole gives the impression of a single *shared database* whose unifying global schema is stored in the server workstation** (see Global site in Figure 1). Consequently, the system must provide the means of interrogating the global schema by users located in remote workstations.
- The system offers a suitable universal query language into which all the individual databases of the system (i.e. local databases) should be translated. The universal query language is used for the development of a global schema which meets the user needs of the composite database application. This language must allow for future extensions and experimentation. In this case a language like OBE (Office by Example)[5], which is the standard language for advanced office applications, may not prove to be appropriate because of its rigid structure. Nevertheless, considering it as a potential candidate for a particular local database will increase the generality of the proposed approach. Moreover, the provision of a browser system such as Baroque[6], where attributes can be accessed by value (instead of the traditional way of providing only attribute names) could prove to be highly

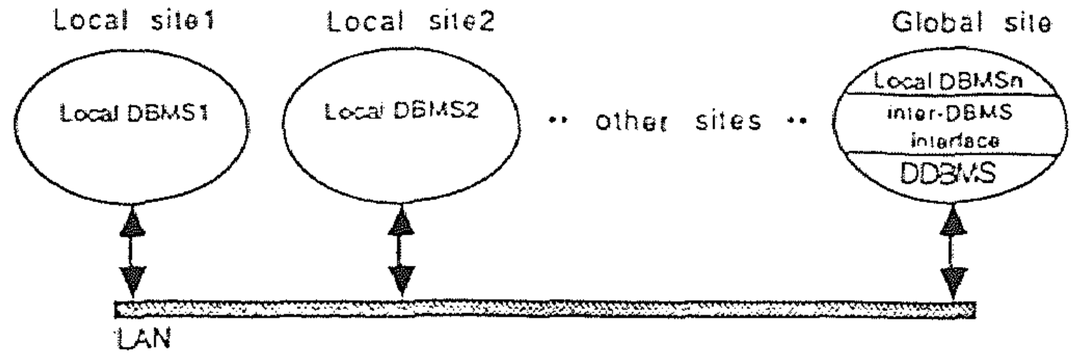


Figure 1. Interconnecting disparate databases in a distributed office

* We assume an office environment in which moderate amounts of data are stored and processed.

** In this context we use the terms server and global site interchangeably to refer to the server workstation.

to the naive user.

- The system provides special concurrency control mechanisms to cater for long lived transactions. These transactions are non-atomic and involve constant interaction with the user[7] as he may need some time to work out his input, or may wish to consult somebody else before completing a transaction. In the envisaged office environment, we can then distinguish between local and global transactions: the combination of all statements issued locally and concerning a local database is assumed to constitute a single local transaction[8]; on the contrary, if transactions are directed to the global database, then the system is required to perform both local and global operations on data.

To facilitate the integration of sites around the LAN, software components have to be introduced which undertake the control of the message traffic in the network. Two basic assumptions characterize the integrated office environment:

1 Databases in personal workstations in the office environment are likely to be small, hence the schema integration process will be cost effective.

2 Most of the queries issued at local sites refer to local entities, global queries are infrequent and serve to satisfy the requirements of local users with respect to entities or attributes stored in remote databases. In this way, there is a very small likelihood of the server site becoming a bottleneck.

Under the assumption that all local DBMSs are relational, we have to introduce a series of local front-end extensions to cope with the requirements of local transparency and autonomy. From what has been mentioned above it becomes evident that within an integrated office environment one has to discriminate between the characteristics and the functionality of local sites *versus* those of the global site.

Local sites

Each workstation can be used as a stand alone system, and also as a remote system with respect to the network's server. Each database can function either as an autonomous database or as a client of the global database. Under this perspective, the server does not need to contain the entire data in the system; rather, it contains a virtual schema and a concise directory (global directory) where information concerning the databases of the system are deposited.

To achieve local transparency we require both uniform and integrated access to remote DBMSs, a fact that implies the existence of a common means of describing local counterparts of globally known information; a common universal language; and unified local directories[9]. Thus, we need to introduce the following additional components:

- *Local Directories*
  Each local site contains a directory maintaining information about the relations and data held locally. Directories also contain information (descriptions) concerning the global schema, so that each local DBMS can decide about the correctness of global queries.

- *Global query language support*
  At each local site, the user can discriminate between its local query language and the global SQL-like language. The coexistence of these two languages in each local site requires translating between these two languages in case of local queries formulated in the global language*. Such inter-language transformations can be performed along the lines described by Howells *et al.*[10] where it is shown how the translation process between a family of relational query languages can be automated. As is discussed below, we also propose the decomposition of the locally issued queries to a low-level network transfer language[11] which contributes to the reduction of the amount of data that has to be transferred across the network.

Global site

The global site contains the global schema which captures the basic functional relationships and constraints between local data. The global site handles the entire volume of meta-data** and at the same time coordinates the operation of the entire system by means of the transaction manager. Although the DBMS of the global site has features common to those of the local DBMSs, it additionally contains appropriate logical interfaces to guarantee the overall system functionality. The supplementary components provided by the global DBMS, or Distributed DBMS, are as follows:

- *The global directory facility*
  This global directory can be thought of as a distributed data directory that comprises a series of system defined tables which include the meta-data pertaining to all local entities participating in the formation of corporate data. This component has at its disposal all the appropriate mechanisms that carry out all necessary global-to-local or local-to-global directory management facilities.

- *Conflict manager*
  One of the main problems in integrated information systems modelling is the handling of name conflicts. With name conflicts we mean situations where semantically equivalent data have different names, or alternatively semantically different data have the same name in different local sites. The conflict manager treats name anomalies in a uniform manner by consulting the appropriate information stored in the global directory. It is the task of the system administrator to update the conflict resolution tables of the global directory with the aid of an interactive semi-automated tool, and by taking into account such issues as the scope or semantical equivalence of locally defined attributes[12, 13].

- *Global transaction manager*
  This global manager is responsible for the coordination of the sub-queries directed to the local sites: it keeps track of the user interactions with the global server, and guarantees optimal query decompositions while ensuring consistent access to the corporate information.

---

* It is assumed that every local query language in the network should conform to the global query language.

** In this context the term meta-data is used to imply information about the entities and relationships that collectively simulate the composite universe of discourse being modelled.

As mentioned earlier, the distributed database management (DDBMS) facility in the integrated office environment resides at the global site only. The functionality of the DDBMS relies on front-end extensions which operate on top of each local DBMS, and the DBMS at the global site which actually implements the DDBMS. The purpose of these extensions is to integrate disparate DBMSs (by surmounting the problems of overlapping or inconsistent data), and to provide the user with a unified language to manipulate the data residing in one or more databases in a transparent manner. The advantage of this topology is that each local DBMS can operate autonomously on its local database when it is not participating in a global session. Consequently, there may be complete local control over access, and resource allocation.

LOCAL AND GLOBAL QUERY PROCESSING

The coupling of diverse local databases implies a partitioning of the global schema into a set of virtual fragments each corresponding to logical portions of the global schema. Such virtual fragments can be located in several local sites of the database network. Schema fragmentation is driven by appropriate information supplied by the database administrator during the process of global schema design.

The objective of schema fragmentation is to evaluate the expedience of partitioning the data entries into horizontal and vertical fragments (i.e. collections of records and collections of attributes, respectively), and to determine subsequently the allocation of these fragments over the various local database sites[14]. Virtual fragments can be collectively assigned to the particular site to which they correspond, and are considered to be the local schema at this specific site[8]. This design methodology aims to provide a sound basis for the minimization of excess processing overhead in a distributed database system, and thus has a direct bearing on the performance of the system. Feedback to this design methodology can be estimated only when the system first becomes operational. As a consequence, a finer tuning of the distribution design may be achieved while several optimization algorithms are examined for improved efficiency by using more precise sample input data.

Schema integration methodology

In a distributed office environment, it is desirable to enhance the conceptual modelling capabilities of the global system, without investing any additional effort on restructuring the representation of local entities. To achieve a proper representation of the global information the designer may have to specify complicated views in terms of the local schemata at the global level of the distributed office environment. One can increase substantially the effectiveness and cohesiveness of the database network by incorporating the generalization mechanism at the global level. Generalization is an abstraction mechanism that permits a set of local schemata belonging to the very same database context to be viewed and represented as a concise generic schema at the global level, and can be thought of as corresponding to a mixed form of fragmentation, i.e. a combination of horizontal and vertical fragmentation[8]. An additional reason for the adaptation of generalization is directly related to improving the semantic expressiveness of the global schemas.

The concept of generalization is associated with the description of the *schema integration* rules. Schema integration rules define broadly how global schema entities (i.e. relations, attributes) are derived (composed) from corresponding local ones. These are in fact the mapping rules that record the ranking of locally defined relations in terms of the global schema. Previous work on schema integration[12, 13, 15] has demonstrated the need of generalization and resolution of conflicts at the attribute level for connecting local schemata into a single integrated database view. By employing generalization at the global level, one can effectively represent the replicated (overlapping) information of the underlying data.

Schema integration rules which lead to the development of the global schema are more complex than those required for the integration of relational views (external schemata) into a single conceptual schema. This is due to the fact that global schema mapping rules have to resolve all possible modelling inconsistencies and naming conflicts among the underlying local databases, whereas the relational view mapping rules do not face such problems. A global schema cannot be thought of as the disjoint union of all the local schemata in the database network, without considering any particular integration rules specifying the equivalent local information portions. After the identification of the semantically equivalent information, various integration operations can be applied leading to a uniform representation of the global information.

An implementation policy for the generalization mechanism at the global level should fulfill the following important requirements:

- Automatic generation of system identifiers for global relations.
- A method to determine the composition of attributes of a global relation in terms of the attributes of the local relations.
- A facility to correlate the global key values with the primary key values in local relations.
- An update mechanism for global directories which also reflects the required domain conversions between semantically equivalent attributes at the local level.

Performing an integration of local schemas can be a very difficult, tedious and error-prone process. It is suggested that an interactive tool is used to assist database designers and administrators in integrating local schemas[16]. It is the purpose of this tool to collect information required for integration from the database administrator, perform the essential bookkeeping (e.g. directory updating), and integrate local schemas according to the semantics provided by the database designer. The schema integration tool essentially facilitates the automatic definition of generalization hierarchies. This tool must also automatically undertake the creation and maintenance of the required global directories.

In the following we elaborate on the specification and use of generalized entities at the global level. For the sake of simplicity we consider just two distinct relation types, *Employee* and *Person* (see Figure 2). It is assumed that these two relations reside in two different locations in the

Rel.Type EMPLOYEE

#emp_no	emp_name	sex

Rel.Type PERSON.

#pers_no	pers_name	qualificat.	sex

Rel Table

rel_name	key_attr.
EMPLOYEE	#emp_no.

Rel Table

rel_name	key_attr.
PERSON	#pers_no.

Attr Table.

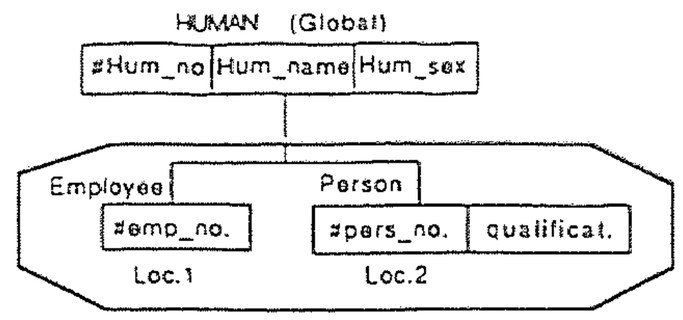
attr_nam.	rel_nam.	type
#emp_no.	EMPLOYEE	num.
emp_name	EMPLOYEE	char
sex	EMPLOYEE	char

Attr Table.

attr_nam.	rel_nam.	type
#pers_no.	PERSON	num.
pers_name	PERSON	char
qualificat.	PERSON	char
sex	PERSON	char

Location 1. Location 2.

Figure 2. Definition of local relation-types and contents of local directories



Key composition Table.

#Hum_no = (#emp_no OR #pers_no) IF (#emp_no = #pers_no)
#emp_no IF #emp_no ≠ #pers_no AND #emp_no IN Employee
#pers_no IF #emp_no ≠ #pers_no AND #pers_no IN Preson

Attribute composition table

Human_name = (emp_name OR pers_name) if Human_id IN (Employee OR Person) OR
(emp_name if Human_id IN Employee) OR
(pers_name if Human_id IN Person)

Human_sex = (sex OR sex) if Human_id IN (Employee OR Person) OR
sex if Human_id IN Employee OR
sex if Human_id IN Person

Figure 3. Global type formation, and the Global directories

database network and are composed of the attributes *emp_no#, emp_nam,sex* and *pers_no#, pers_nam,-qualification,sex*, respectively. Attributes with the # suffix represent the locally declared primary key attributes for each relation-type. Figure 2 also depicts the directory entries which correspond to these two relation-types. These directory entries contain detailed information about the attribute composition of the two relation-types, together with information about key-attributes and attribute-types. The *Rel_Table* and *Attr_Table* directory tables are maintained both at the global and local levels as they contain the complete knowledge about the constituents of the local relations. This information will be used at the global level for consistency and integrity checks and as a guideline for correlating the local data and executing global queries.

It is assumed that the relations *Employee* and *Person* have some common meaning, i.e. they refer to semantically related entities and may contain overlapping values. To integrate these two local types the database administrator must define a new global entity, called the composite entity. The role of this entity is to specify the composition rules (also called 'merging condition rules' by Dayal[17]) which identify and correlate the common attributes of local relations. For example, if the designer determines that some attributes of the *Employee* and *Person* entities have a common semantic meaning, then these attributes have to appear in the definition of the composite entity only once. When a query is evaluated against a generalized entity, then the system has to specify on which attribute the local entities must be joined*. Actually, the join attributes of the local entities will be the primary keys of these entities as they uniquely identify each tuple occurrence in the local entities. The join predicate is automatically constructed by the schema integration tool which also establishes the correspondences between the attributes defined in the composite entity and those at the local relation type levels.

In the following we assume that the local relations *Employee* and *Person* are enrolled in a generalization hierarchy *Human* which is composed of the following attributes (see Figure 3):

- *Human_id*: this attribute is the primary key of the generalized relation. It will establish the merging condition for the derivation of the global relation type *Human* from the local relation types. In other words, this attribute will indicate on which fields the two local relations will be joined in order to acquire the information pertinent to the generalized relation. Accordingly, for each unique value of *emp_no* in *Employee* and *pers_no* in *Person*, there will be a unique value for *Human_id* in the *Human* global relation. The declaration of *Human_id* at the global level must differentiate between two cases:

  - the local keys *emp_no* and *pers_no* have overlapping values, in which case the global key *Human_id* is formed as follows:

    Human_id = *emp_no* **OR** *pers_no* **if** *emp_no* = *pers_no*

  - if, however, the keys *emp_no* and *pers_no* are disjoint, then global queries must be evaluated in terms of either the employee type or person type domains. Consequently, the definition of the composite key will be as follows:

    Human_id = (*emp_no* **if** *emp_no* = *pers_no* **AND** *emp_no* **IN** *Employee*) **OR** (*pers_no* **if** *emp_no* = *pers_no* **AND** *pers_no* **IN** *Person*)

- *Human_name*: this attribute is the synonym of the two locally defined attributes *emp_name* and *pers_name*. This attribute derives its value from local data depending on which domain the key attribute *Human_id* draws its values from. The derivation of the value of *Human_name* in terms of the local attributes *emp_name* and *pers_name* comes about as follows:

  - The local keys *emp_no* and *pers_no* have overlapping values. In this case the global attribute *Human_name* is formed as follows:

    Human_name = (*emp_name* **OR** *pers_name*) **if** *Human_id* **IN** (*Employee* **AND** *Person*)

* Due to the fact that null value problems may appear when joining relations, the approach presented assumes the use of the outerjoin operation[3].

- if the keys *emp_no* and *pers_no* are disjoint then as previously explained, global queries must be evaluated in terms of either the employee type or person type domains. Accordingly, the definition of the global attribute *Human_name* can be formed as follows:

Human_name = *(emp_name* **if** *Human_id* **IN** *Employee)* **OR** *(pers_name* **if** *Human_id* **IN** *Person)*

- *Human_sex*: this attribute also reflects the semantic equivalence of both local sex attributes at the global level. As in the previous case, the derivation rules for the values of the *Human_sex* attribute will be as follows:

  - Overlapping *sex* attribute values at the local levels:

Human_sex = *(Employee.sex* **OR** *Person.sex)* **if** *Human_id* **IN** *(Employee* **AND** *Person)*

  - Disjoint *sex* attribute values at the local levels:

Human_sex = *Employee.sex* **if** *Human_id* **IN** *Employee* **OR** *(Person.sex* **if** *Human_id* **IN** *Person)*

The next step in the design phase of the global system consists of the automatic definition and automatic generation of the directory tables. Figure 3b depicts the *Key Composition Table* for the global relation *Human*, i.e. it illustrates the indispensable table entries depicting the proliferation of the key attribute of the super-type relation *Human* to the key attribute level in terms of the key attributes of the local relations. Accordingly, Figure 3c shows the composition rules for the remaining attributes of the relation *Human*, in terms of local attributes. It is worth mentioning that all the four table types presented in this section (see Figure 3) will be maintained at the global level. These tables contain information which controls the acquisition of data for the global relations. This process may also require certain transformations on the format of the local data, depending on the compatibility of the potentially equivalent attributes. This is the result of the fact that locally declared attributes may use the same identifiers for attributes which have totally incompatible definitions (e.g. different types, names, scopes) at the local levels.

ARCHITECTURE OF THE SERVER-SITE ENVIRONMENT

It is suggested that the distributed office environment is designed in a highly modular fashion, so that its architecture can prove to be highly conducive to initial and continuing development, and should facilitate future work and potential extensions or rearrangements. The encapsulation of all manipulation functions in dedicated modules permits experimentation with different relational structures (both local and global) to meet an adequate compromise between physical space utilization and processing efficiency. Accordingly, the overall objective of the distributed office architecture is twofold:

- to provide a versatile office environment which is general enough to support a variety of user applications and offers a reasonable amount of portability.
- to support new interfaces which can be added at a later development stage, at a minimum cost, by reusing existing software components wherever possible.

As explained earlier, to implement the distributed office environment we require major front-end extensions on top of the DBMS at the server site, while all local DBMSs require only restricted extensions to their present structure (see Figure 4). Extensions at the local sites are absolutely identical to associated extensions at the global site. Extensions at local sites deal with the translation of the local query language to the global data language, and a subsequent translation of the Universal Query Language (UQL) constructs into an Internal Network Query Language (INQL) before being decomposed and distributed to the other sites in the network. The INQL provides the means by which different DBMSs may communicate with each other, and its structure is based on the relational approach. It is worth mentioning that costly translations between query languages are performed locally, with the global DBMS left to control the globally issued transactions. This fact eliminates the possibility of having a system bottleneck at the global site, as the preprocessing of global queries is performed at the local databases.

As depicted in Figure 4, applications in the proposed integrated office environment translate their global queries into an INQL. Translated queries are transferred through the network to the server site and subsequently processed by a Low Level Query Evaluator (LLQE). These assumptions suggest the introduction of several other software components, apart from the LLQE, centred around the INQL. This approach presents a variety of benefits such as the localization in one component, i.e. the LLQE, of all the

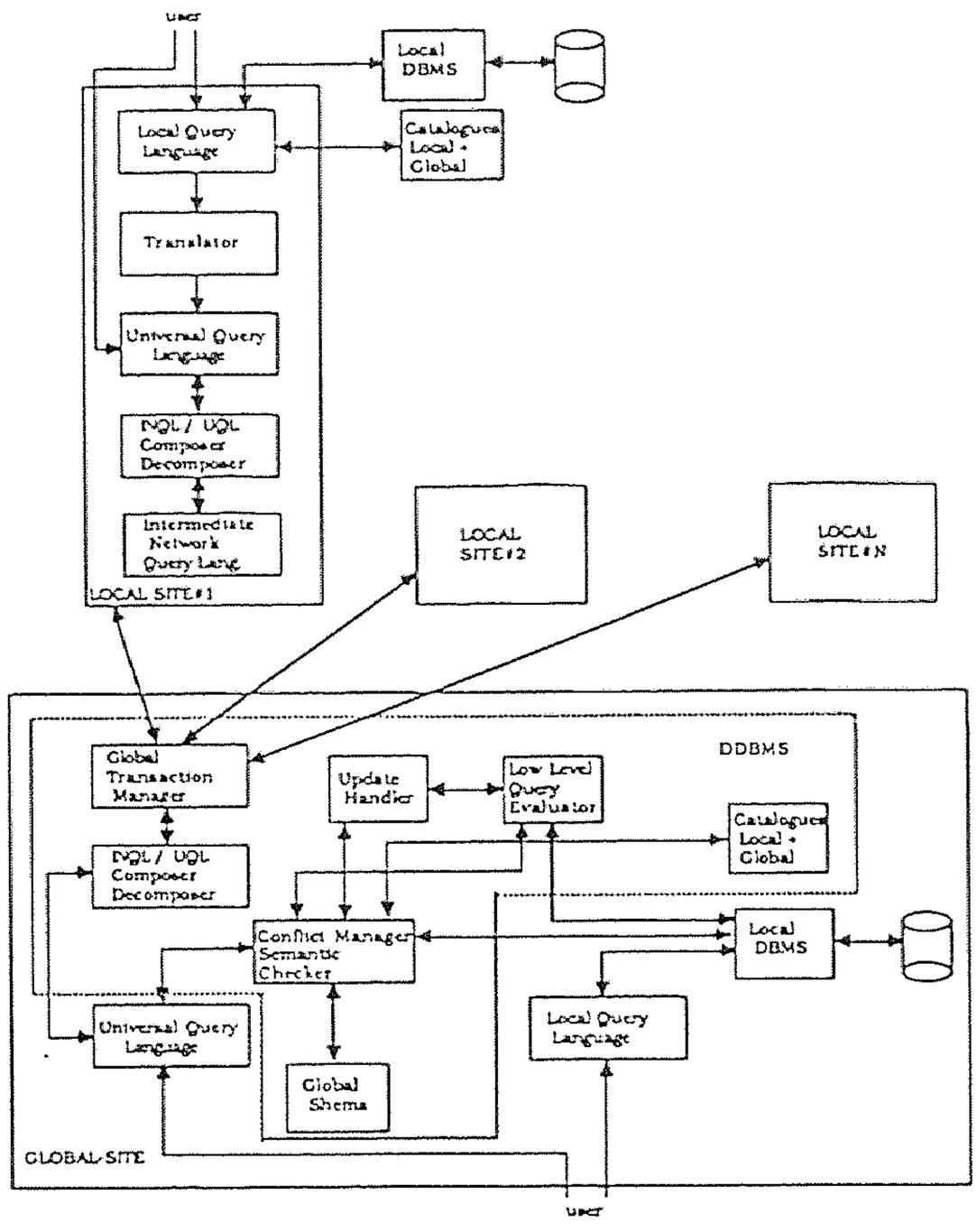


Figure 4. Concise system architecture

query processing methods. To improve performance this component should have knowledge of the structures and the function manipulated by the storage handling component of each local DBMS.

As depicted in Figure 4 the Conflict Manager Semantic Checker (CMSC) serves as the central component of the server site environment. The main purpose of the CMSC is to receive all global information requests from local users, perform semantic checks and resolve all conflicts (e.g. name conflicts, partial information overlapping). In addition, it decomposes them into the necessary sub-queries; forwards the sub-queries via the Global Transaction Manager to the appropriate locations; and finally assembles the partial answers to formulate a composite answer corresponding to the original query. In doing so, the CMSC redirects the sub-queries to the Local/Global Query Translator situated at each local site, (see Figure 4). Queries formulated on the basis of the global schema are forwarded to the CMSC. The CMSC facilitates the system to restructure a relation (i.e. change or add identifiers, if necessary) provided that the consistency and integrity rules (constraints) specified by the Update Handler are satisfied. The CMSC translates user-specified names into system internal identifiers, and ensures that the entities referred to by them already exist. To achieve this the CMSC must have access to both the global schema definitions and directory facilities. The existence of the CMSC isolates internal components from user interfaces, simplifying thus internal references to entities by utilizing system specified identifiers.

Any attempt to modify the database by the user is checked for consistency and privileges in a special purpose component named the Update Handler (UH). The UH enforces the specified integrity constraints by taking the appropriate actions if an access violation is attempted (e.g. an operation may be rejected, violations are recorded and reported to the user, etc). The UH adheres to the entity integrity and referential integrity rules for relations as described by Date[3]. Relation integrity rules concern the admissibility of a given tuple as a candidate for insertion into a given relation, or the relationship between tuples of one relation and those of another.

The component required to facilitate query formulation is the LLQE. The provision of a common LLQL for a system supporting diverse relational, or quasi-relational dialects, requests the definition of a common INQL. The view is taken that the INQL should be a stack-based query lang-uage capable of compressing relational query statements while retaining the functionality of the relational algebra. The provision of such concise language statements can speed up the transfer of data by reducing transfer costs. Moreover, as local catalogues contain adequate descriptions of the global schema (see above), INQL requests can represent the subqueries in an optimized form to result in an efficient execution of global queries. In the following we explain briefly how queries can expressed in INQL, and what types of messages can be exchanged in an integrated office environment.

Internetwork message exchange facilities

All query language constructs requiring global processing should be mapped to the INQL and subsequently be transferred to the server site. Such a stack-oriented query language has already been implemented as an internal query language for the distributed system Proteus[18]. The structure of INQL is specifically amenable to computer processing, and its form has been chosen to balance the following general characteristics:

- a simple subset of the query facilities can be implemented easily;
- its basic structure is extensible to meet any new processing needs; and
- a sophisticated query processor or query optimizer is feasible.

A Network Transfer Language (NTL) is used to transfer messages across the common communication network. A message typically consists of a message header and a message body. The message header contains information pertaining to the source and destination sites of the message, and an indication of the type of message. It is suggested that at least the following four types of messages are supported: mail, query, error and response. If, for example, a query is evaluated successfully, then the result of this query is transferred to the appropriate location in a response message. However, if query evaluation should fail for some reason - such as a constraint violation or site unavailability - then an error message is generated. (More information on the format of such messages can be found in References 11 and 19.) Of particular interest to us are the query messages of NTL. As explained earlier, all query messages expressed either in a local query language or in the universal (global) query language are mapped into INQL. Consequently, the body of an NTL query message will consist only of an INQL query.

INQL consists of a set of one and zero address instructions, intended for a hypothetical stack machine. Each instruction is indicated by a mnemonic and includes a type. The types that the instructions can assume are Boolean (b), integer (i), real (f), string (s) and relation (r). It is beyond the scope of this paper to discuss the INQL constructs in detail; instead, a sample query is explained in outline.

As an example, consider the simple schema:

SUPPLIER < SUPPLIER_ID#, SUP_NAME, PRODUCT_NO, LOCATION >.

and the query "List the *supplier_id*, and *name* of all suppliers located in London". This query can be expressed in INQL as follows:

```
    /* principal part of query */
$begin                  /* begins interpretation of the query */
rload   SUPPLIER        /* load the relation identifier */
rselect  L1             /* select tuples using the label L1 */
rproject L2             /* project attributes using the label L2 */
rput temp               /* result is a new relation on top of the stack */
rresult res             /* result is the list of entities at label res */
$end                    /* end of interpretation at main query body */

    /* selection procedure */
$L1: sload   "London"   /* load the string London onto the stack */
     sattr   Location   /* load attribute value of type string */
     sequal             /* compare two top stack items for equality */
     bresult            /* result is a Boolean on top of the stack */

    /* projection list */
$L2:sattr    sup id     /* load the string attribute sup_id onto
```

```
                        the stack */
    sattr  sup_name  /* load the string attribute sup name onto
                        the stack */
    end              /* end of projection list */

      /* assignment operations */
$temp: rvar          /* declare result relation a variable 'temp' */

$res:   rget temp    /* result list */ end
```

with each operation which defines what particular processing steps should be performed. In particular, the relation SUPPLIER is first loaded onto the stack, and a selection expression is carried out on it as indicated by the instructions at label L1. Then, the selected tuples from the relation SUPPLIER are projected as specified by the projection operation at the label L2. Finally, the result of the query is specified as being the list of objects at the label **res** (in this example just the transient relation **temp**).

The set of resulting INQL statements is passed to the LLQE as a tree and not as a sequence of instructions. Each query expressed in INQL is decomposed into a tree where each node contains a relational operator and its parameters, which are the instructions under the label of the operation as shown in the previous example. Reconstruction of the original tree is more straightforward than parsing some bracketed text using a precedence grammar. Furthermore, the conversion of INQL from a tree shape to a sequence of instructions, and *vice-versa*, is a rather trivial process. The sequential form of instructions is used in the distributed environment for transporting a given query or subquery to other sites.

CONCLUDING REMARKS

The purpose of this paper was to investigate the feasibility of applying distributed database technology, at a moderate cost, in a potentially expanding spectrum of applications in an office environment. In particular it was suggested that the considerable increase in power resulting from the transition from a centralized office environment to a distributed office environment can be achieved with few and relatively simple extensions to the already existing database management systems. Integration in this kind of distributed office environment can be only achieved if all local database descriptions are made relative to a single global schema, and all accesses to the database are made relative to that schema.

The proposed system is oriented towards databases requiring moderate resources; personal or small group databases in an office environment are likely to benefit most from the proposed approach. However, the approach can be easily generalized to fit the requirements imposed by a large office environment which, as a rule, divide their tasks among various departments. A department can thus control its private cluster of workstations and databases through a central server according to suggestions given. Such a cluster may in turn cooperate with other clusters and exchange valuable information via a master cluster unit.

It is worth mentioning that the directions outlined here are general, and could be applicable in most cases with small effort. Generally, proposing a distributed office system, entails the provision of a series of special purpose tools to improve productivity by assisting office workers in locating accurate and valuable information (two things which are not feasible in a centralized office environment).

REFERENCES

1 **Papazoglou, M P and Marinos, L** 'Requirements for coupling small scale DBMSs in a distributed office environment' *Microprocess. & Microprogramm.* Vol 19 No 5 (1987)

2 **Marinos, L, Papazoglou, M P and Norrie, M** 'Towards the design of an integrated environment for distributed databases' in **Chiricozzi, E and Amico, A D (eds)** *Parallel Processing and Applications* North-Holland, Amsterdam, Netherlands (1988)

3 **Date, C J** *Selected Writings* Addison-Wesley, NY, USA (1986)

4 **Chamberlin, D D and Boyce, R F** 'SEQUEL: a Structured English Query Language' *Proc. 1974 ACM SIGMOD Workshop on Data Description, Access & Control* (1974)

5 **Zloof, M M** 'Office by Example: A Business Language that Unifes Data and Word Processing and Electronic Mail" *IBM Syst. J.* No 21 (1982)

6 **Motro, A** 'Baroque: a browser for relational databases' *ACM Trans. Offlce Info. Syst.* Vol 4 No 2 (1986)

7 **Kim, W, Lorie, R, McNabb, D and Plouffe, W** 'A transaction mechanism for engineering design databases' *10th VLDB Conf.* Singapore (1984) pp 355-362

8 **Ceri, S and Pelegatti, G** *Distributed Databases: Principles and Systems* McGraw-Hill, NY, USA (1984)

9 **Gligor, V D and Fong, E** 'Distributed Database Management Systems: An architectural perspective' *J. Telecommun. Networks* Vol 2 No 3 (1983)

10 **Howells, D I, Fiddian, N J and Gray, W A** 'A source-to-source meta-translation system for relational query languages' *13th VLDB Conf* Brighton, UK (1987) pp 227-231

11 **Atkinson, M P** *et al.* 'The Proteus distributed database system' *Proc. 3rd Nat. Conf. on Databases* UK (1984)

12 **Effelsberg, W and Mannino, M V** 'Attribute equivalence in global schema design for heterogeneous distributed databases' Info. Syst. Vol 9 Nos 3/4 (1984)

13 **Mannino, M and, Karle, C R** 'An extension of the general entity manipulator language for global view definition' *Data & Knowl. Eng.* Vol 1 (1985)

14 **Navathe, S and Ceri, S A** 'Comprehensive approach to fragmentation and allocation of data in distributed databases' *IEEE COMPCON* (1983)

15 **Navathe, S, Elmasri, R and Larson, J** 'Integrating user views in database design' *IEEE Comput.* (1986)

16 **Sheth, A P, Larson, J A, Cornelio, A and Navathe, S** 'A tool for integrating conceptual schemas and user views' *4th Int. Conf. Data Eng.* Los Angeles, CA, USA (1988) pp 176-183

17 **Dayal, U** 'Query processing in a multidatabase

system' in **Kim, W, Reiner, D and Batory, D S** (eds) *Query Procesing in Database Systems* Springer-Verlag, Berlin, FRG (1985)

18 **Atkinson, M P, Gray, P MD and Hepp, P** *Message Formats for Intersite Communication Proteus Working Paper E2 (version 5)* University of Edinburgh, UK (August 1984)

19 **Norrie, M** *The Edinburgh node of the Proteus distributed database system* Internal Report, CSR 191-85, University of Edinburgh, UK (1985)