The Model-Assisted Global Query System for Multiple Databases in Distributed Enterprises

WAIMAN CHEUNG
The Chinese University of Hong Kong
and
CHENG HSU
Rensselaer Polytechnic Institute

Today's enterprises typically employ multiple information systems, which are independently developed, locally administered, and different in logical or physical designs. Therefore, a fundamental challenge in enterprise information management is the sharing of information for enterprise users across organizational boundaries; this requires a global query system capable of providing on-line intelligent assistance to users. Conventional technologies, such as schema-based query languages and hard-coded schema integration, are not sufficient to solve this problem. This article develops a new approach, a "model-assisted global query system," that utilizes an on-line repository of enterprise metadata—the Metadatabase—to facilitate global query formulation and processing with certain desirable properties such as adaptiveness and open-systems architecture. A definitional model characterizing the various classes and roles of the required metadata as knowledge for the system is presented. The significance of possessing this knowledge (via a Metadatabase) toward improving the global query capabilities available previously is analyzed. On this basis, a direct method using model traversal and a query language using global model constructs are developed along with other new methods required for this approach. It is then tested through a prototype system in a computer-integrated manufacturing (CIM) setting.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*distributed systems*; *query processing*; H.2.7 [**Database Management**]: Database Administration—*data dictionary/directory*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms: Design, Performance

This research was supported in part by National Science Foundation grants DDM 9015277 and DDM 9215620 and by Rensselaer's CIM and AIME Programs, both of which are sponsored by ALCOA, Digital Equipment, General Electric, CM, and IBM, and was conducted under the Center for Manufacturing Productivity and Technology Transfer.

Authors' addresses: W. Cheung, Faculty of Business Administration, The Chinese University of Hong Kong, Shatin, N.T. Hong Kong; email: wcheung@cuhk.hk; C. Hsu, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY 12180-3590; email: hsuc@rpi.edu.

Additional Key Words and Phrases: Enterprise integration, global query system, heterogeneous distributed information systems, metadatabase

1. GLOBAL QUERY SYSTEMS

The notion of information-based enterprises has become a reality. For various organizational and technological reasons, information systems in these enterprises are typically characterized by heterogeneous, distributed environments. For example, a computerized manufacturing enterprise may have a number of shop floor subsystems that are implemented with different file management systems on various platforms, while its business and engineering design subsystems are operating in relational and object-oriented environments distributed over wide-area networks. In all likelihood, this multiplicity will not disappear, nor be replaced by an all-encompassing standard any time soon. Therefore, a major objective of information integration for these enterprises is to provide a logical structure to integrate these islands of information resources for enterprisewide information sharing and management without relying on a fixed controlling hierarchy. A key requirement here is what might be called the "*on-line intelligence and assistance*" capabilities of the integrated systems for supporting enterprise users' (varying) needs in retrieving information at a global level from the multiple local systems, which may frequently change their operating rules as well as contents. Numerous research and commercial systems have evolved over the past decade toward providing these capabilities for single-site or multiple-site databases. However, a rigorous formulation of this requirement expressly for the global query needs of multiple systems has not been provided previously in the literature, nor has such a technology.

1.1 The Need for Enterprise Metadata Support

Metadata has been increasingly recognized as a key element in global query systems [Babin 1993; Bouziane 1991; Cheung 1991; Collet et al. 1991; Ferrara 1994; Hsu et al. 1991; 1992; Siegel and Madnick 1991; Wang and Madnick 1989]. The questions raised here are: how much and what metadata should a global query system process in order to effect on-line intelligence and assistance capabilities, and what architecture can manage and utilize the metadata to suffice these capabilities?

To illustrate the significance of these desired capabilities, we consider some basic tasks required of a global query system in a heterogeneous, distributed environment (e.g., see Cheung [1991], Krishnamurthy et al. [1991], Litwin et al. [1990], and Reddy et al. [1989] for a survey on these tasks). A typical global query operation involves two steps: global query formulation and global query processing. In the first step, the user's requests are articulated and represented in a way that the global query system understands. The query formulation is done primarily through the

user interface of the system. In the second step, accomplished by the system internally, queries are sent to appropriate local systems to retrieve pertinent information and are reassembled for the users.

Toward query formulation, on-line intelligence enables the user interface to provide assistance in the articulation as well as allow for a high-level, intuitive representation of queries. Specifically, for the *articulation* of queries, the system would utilize its knowledge on the enterprise information resources (which are referred to in this article as *metadata*, including information models, implementation models, and business and operating rules—see Section 2.2.2 for a formal description) to alleviate semantic ambiguity, facilitate logical analysis, and enhance adaptive construction during the formulation of queries. Similarly, the *representation* itself could accommodate heterogeneity in local models across the enterprise through, for example, the knowledge on enterprise SUBJECTs (i.e., applications, views, and objects) and the equivalence of data items among different local systems, without having to impose a single, fixed "integrated schema" on all databases.

In addition to supporting high-level and nonsyntax-based queries for enterprise users, the system would also handle all context-based interpretations and dynamic mappings between the globally formulated query and the locally implemented file structures or database schemata. Assisted with these capabilities, the second step—global query processing—would be accomplished in the following fashion:

(1) *query optimization*: the global query is first decomposed into local queries taking into account both semantics and performance;
(2) *query translation*: the local queries are then encoded in their respective data manipulation languages;
(3) *query execution*: the encoded local queries get dispatched to and processed at their respective systems; and finally
(4) *result integration*: results from local systems are assembled to answer the global query.

Each of these processing steps makes use of metadata as well. For instance, local database schemata, directory and network information, and the contextual knowledge of data are required for query optimization; local DBMS information for query translation; operating rules on information flows for query execution; and knowledge on equivalent objects, incompatibility, and data conversion for result integration.

Without the assistance from sufficient on-line knowledge in the form of metadata, all of the remaining information required above for both query formulation and processing would have to be provided by the users or application programs, such as the case in previous systems, regardless of the interfacing designs used [Afsarmanesh and McLeod 1989; Angelaccio et al. 1990; Azmoodeh 1990; Bernstein et al. 1981; Chung 1990; Greenberg and Witten 1985; Motro 1990; Shipman 1981; Shyy and Su 1991; Stonebraker 1988; Wang and Madnick 1989; 1990]. The problem of lacking

on-line metadata support is especially acute for multiple systems, where researchers have increasingly emphasized the use of enterprise metadata (e.g., see Bouziane [1991] and Hsu et al. [1991] for a survey on this topic). A basic reason is the significant differences in the CONTEXTs (i.e., processes and contextual knowledge) in which data are utilized, on top of the complexities in the varying data semantics (models), data manipulation languages, and data structures among local systems (a discussion of the contexts is provided in Hsu and Rattner [1990] and Hsu et al. [1993]). Either the users or the global system itself must abridge these differences for each query before the information can be shared. Since enterprise users generally do not possess the expertise in database technology, nor the technical knowledge about the local systems, they cannot truly benefit from a global query system that does not possess sufficient metadata to provide on-line intelligence and assistance. The notion of enterprise metadata is concerned with all metadata pertaining to the above SUBJECTs and CONTEXTs (see Section 3.1 for details).

1.2 User Interface for Query Formulation

User interface techniques are important to global query systems; however, do they supplant the need for enterprise metadata support? Techniques such as windows, icons, menus, graphics, visualization [Greenberg and Witten 1985; Gyssens et al. 1990; Hartson and Hix 1989; Hutchins et al. 1986; Nilan 1992; Robertson et al. 1993] and other forms of nontextual formalisms (e.g., see Hsu and Lee [1993] and Lodding [1983] for a survey) free the typical user from having to learn sophisticated programming languages. Therefore, graphical user interface (GUI) technologies have been employed, together with cognition-theoretic interface design principles and guidelines [Gardiner and Christie 1987; Gould and Lewis 1985; Greenberg and Witten 1985], to facilitate database query tasks. The results have improved significantly the commercially available database query systems (e.g., GUI add-ons to SQL for a few relational systems [Benjamin and Lew 1986]). Notwithstanding, these products still do not support users with on-line metadata on enterprise models (especially contextual knowledge of data) required for global query systems in heterogeneous environments. Users are still charged with the responsibility of furnishing much of the technical information mentioned above. Moreover, since these systems do not offer an on-line global model separate from the schemata, which are fixed, their user interfaces tend to be hard to change or to customize as the underlying systems or the users change.

The same observations are largely applicable to natural-language interfaces. In principle, any systems that use natural languages should not require the users to learn the artificialities of correct command formats or modes of interaction [Blaser 1988; Bunt 1988; Hirschman 1989; Norcio and Stanley 1989; Rich 1984]. Unfortunately, few systems (even research systems) have successfully achieved this goal, due mainly to insufficient results in linguistics and artificial intelligence to support this class of user

interface. However, even in the ideal case, a natural-language interface would not be able to assist on the query formulation itself for the same reasons as a GUI faces: it does not provide users with such knowledge as the local and global data resources, dictionary and directory knowledge, and business rules about the enterprise, all of which are needed before a user interface could ever support a user in a "natural" mode of query formulation.

Addressing the need of providing metadata to users, database browsers have been developed to help end-users look through the contents of a particular database. Most of these database interfaces are limited to browsing the data instances and support only single-site and single-tuple queries; moreover, few look beyond the simple database schema per se, which does not include other types of enterprise metadata [Herot 1980; Motro 1990; Smith et al. 1981; Stonebraker and Kalash 1982; Zloof 1977]. They, ironically, provide interesting evidence supporting the thesis that enterprise metadata can lead to a new breakthrough for the problem.

1.3 Integrated Schema for Global Query

Enterprise metadata as discussed above are complex in their own right. Thus, it is natural to expect an architecture devoted to them for the above-stated tasks. Is the conventional integrated schema technology sufficient?

A key issue is how to reconcile and consolidate the various views and representation methods across the enterprise and still retain local differences for autonomy and flexibility. Most previous efforts employ a solution strategy emphasizing the development of global architectures based on a schema integration (e.g., Batini et al. [1986]) approach. Although an integrated schema is a facility of metadata, its hard-coded nature tends to contradict or even nullify some of the basic promises of true local autonomy, such as openness and adaptiveness [Litwin et al. 1990]. Moreover, research efforts (e.g., Afsarmanesh and McLeod [1989], Chung [1990], Krishnamurthy et al. [1991], Siegel and Madnick [1991], and Wang and Madnick [1989]) have also revealed that additional enterprise metadata beyond the integrated schema are needed to facilitate the representation of global views and the management of query transactions among local databases. Therefore, these efforts have by contradiction shown an even bigger role for enterprise metadata, that is, minimizing the reliance on some fixed, hard-coded global schemata or controller to effect information integration.

Thus, the conventional approach of integrated schema does not seem to provide a solution to the enterprise metadata management problem. To bring the point into better light for our discussion, we refer to the concept of an independent system of enterprise metadata supporting the above purposes as a *Metadatabase* (which is first defined in Hsu and Skevington [1987]).

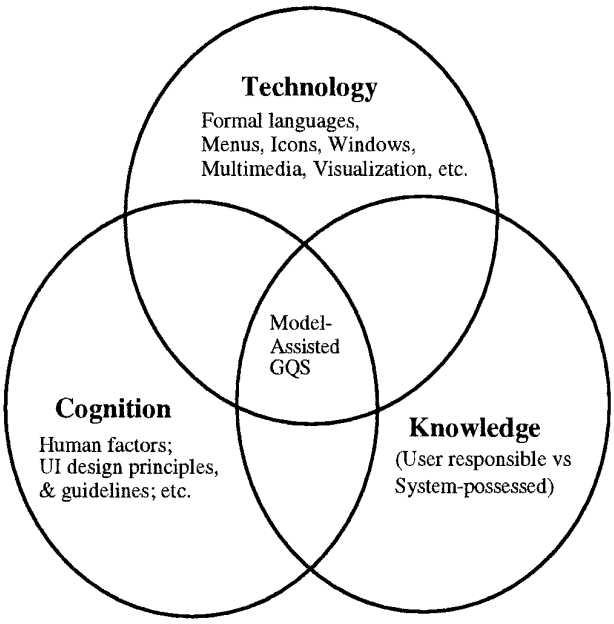


Fig. 1. Dimensions of global query user interfaces.

1.4 The Objective and Organization of the Article

A conclusion from the above discussion is evident: the man-machine interface of global query systems requires a combination of technology, cognition, and knowledge, as depicted in Figure 1. The one dimension that has long been neglected is "knowledge," which should be put on-line to provide intelligent assistance to users. We submit that the idea of enterprise metadata holds a key for effecting this dimension; that is, knowledge through enterprise metadata is elevated and explicitly formulated to play the central role in a new solution approach to solving the global query problem in this research. Since enterprise metadata are essentially information models, this approach is referred to as the *model-assisted global query system*. The original concept of such a system, along with its execution methods, is the major contribution of this article.

Specifically, the conceptual model formulates the fundamental needs for enterprise metadata in general terms, which other systems can adopt, with or without the Metadatabase. The execution model then avails a direct method using model traversal to assist end-users and a new high-level language using global model constructs, both of which are based on a Metadatabase enabling the assistance. Automatic derivation of required metadata, which the users do not provide, is a key element in this approach; thus, new methods, such as rule-based consistency checking and message-based search for a shortest solution path, are developed for the execution model. These results lead to a prototype developed at Rensselaer recently to test the validity of the approach and demonstrate the global query ability for a multiple-database environment.

The next section presents a conceptual model of this approach, defining the uses and requirements of metadata in a global query system. New methods that are required for the implementation of the approach are provided in Section 3, while the implementation prototype and the new query language using metadata are included in Section 4. The empirical verification of the Model-assisted Global Query System (MGQS) is discussed in Section 5 through using the prototype for an integrated manufacturing case. An analysis of this system vis-a-vis some representative systems (e.g., Multibase, MSQL, CIS, and KIM query system) in the literature is presented in Section 6. Section 7 concludes the article with further remarks and future directions of research. Although the prototype system has been reported elsewhere as a part of the Metadatabase system for certain industrial applications [Hsu et al. 1992; 1994], the complete model—both conceptual and execution—and methods are presented and discussed in this article.

2. THE MODEL-ASSISTED GLOBAL QUERY APPROACH

The basic logic of the model-assisted global query approach proceeds as follows. First, all classes of enterprise metadata are specified and structured through a metadata representation method. This metadata structure (abstraction) then serves as the basis for organizing and implementing enterprise information models into an on-line and shared Metadatabase facilitating all tasks of information integration. As such, the functional views, processes, data models, business rules, and other knowledge concerning the global query operation are readily available to both the users and the system through the Metadatabase. Therefore, on-line assistance on query formulation and processing becomes a feasible and fully characterized concept. Specific methods based on this knowledge can be defined and developed in terms of metadata requirements and utilized in each major task of the problem.

2.1 The Goals

The target of the model-assisted global query system is delineated in the goals below, which will later be used as the criteria for comparing the MGQS with the existing technologies.

(1) *Information sharing*: achieve information sharing in heterogeneous, distributed, and autonomous environments by means of global queries.
(2) *Subsystem transparency*: support a global model of the whole system and hide local implementations from the users.
(3) *Local autonomy*: maintain local control of its own applications and allow (potentially) for local differences. In addition, this implies that integration of local systems should not necessitate major conversions or merging of the existing systems.

(4) *Interoperability*: accommodate local heterogeneities while resolving conflicts in data equivalency, different data models, and different data manipulation languages.

(5) *Open-system architecture*: support the flexibility and adaptability for incorporating new application systems or updating old ones without imposing major recompilation of reconstruction efforts.

(6) *Direct query formulation*: provide sufficient enterprise metadata to facilitate the articulation and representation of a global query using the information models directly via a noncommand user interface (or minimum-syntax query language for program interface). The user or programmer is not responsible for providing the technical details of the local systems.

(7) *On-line assistance*: use enterprise metadata (including business rules and other contextual knowledge) and knowledge-processing capability to assist on difficult tasks for both query formulation and processing. These tasks include, but are not limited to, model traversal and semantic checks in direct query formulation, derivation of implied data items and operating rules in the query, context-based joint path optimization, and data equivalence in the assembly of local results.

2.2 The Definitive Model for Metadata Requirements

We first formally characterize the role of a Metadatabase as on-line knowledge for a global query operation. This characterization starts with a technical analysis of the major global query tasks and their basic metadata requirements. Since these tasks are generic and are not tied to any specific systems, the analysis applies to the general problem studied in this article.

2.2.1 *A Global Query Operation Algorithm.* Let GQ denote a global query characterized by a set of attributes/data items (A) that are involved in the query operation, a set of persistent, stored data objects (D) from which all the attributes are drawn, and an expression ⟨C⟩ that specifies the retrieval conditions. Expression ⟨C⟩ consists of subexpressions for selection conditions ⟨SC⟩ and join conditions ⟨JC⟩. Finally, all data items, objects, and expressions are subject to specification in terms of systems metadata ⟨M⟩. These metadata, which may be either supplied by the user or provided by the global query system, must satisfy a minimum scope required by each particular query, specifically,

$$
\begin{aligned}
GQ = (A, D, \langle C\rangle|\langle M\rangle) \text{ where}\\
A = A^u \cup A^s \text{ with } A^u \neq \emptyset,\\
D = D^u \cup D^s \text{ with } D \neq \emptyset \text{ and } D^u \cap D^s = \emptyset,\\
\langle C\rangle ::= [\langle SC\rangle|\langle JC\rangle|\langle SC\rangle \text{ AND } \langle JC\rangle],\\
\langle M\rangle ::= \{\langle M^u\rangle\} \cup \{\langle M^s\rangle\}.
\end{aligned}
$$

Although this formalism is constructed at the most general level possible, it does inspire the particular syntax of a query language for MGQS—see Section 4.2. The additional data subsets are explained below:

—A^u: this represents the set of data attributes selected by the user. It includes both the items requested directly for retrieval and the items indicated in the selection conditions. A global query must have at least one data item in A^u.

—A^s: the system may determine that additional data items are also needed or implied in the query, and hence fill in some data attribute (the set A^s) for the purpose of query processing.

—D^u: the user could also specify the set of data object(s) D^u that contain the selected item(s) (i.e., the set A^u).

—D^s: since D^u may not contain all items in A^u, the system will again determine the remaining data objects (D^s) that are involved in the global query operation.

—$\langle M^u \rangle$: this is the user-supplied metadata, which represents the technical knowledge that the user must possess about the enterprise information models and multiple systems in order to represent and specify a query sufficiently.

—$\langle M^s \rangle$: this is the system-supplied enterprise metadata. The set $\langle M^s \rangle$ is the complement of $\{\langle M^u \rangle\}$ with respect to the minimum metadata requirements for a particular query.

The global query operation can be described as a process consisting of the following steps:

Step 1. *Global Query Formulation.* Since the user is not necessarily required to specify all of the technical details (the remaining ones will be filled in by the query system automatically), the result of the formulation is likely to be an incomplete global query IGQ defined as

$$IGQ = (A^u, [D^u], [\langle SC \rangle]|[\langle M \rangle])$$

where the data objects D^u and selection conditions $\langle SC \rangle$ may or may not be required. For example, a global query "Find part ID and quantity completed for Jane Doe's order which has a desired date of 5/10/91" could imply the following sets in the formulation step:

A^u = {PARTID, NUM_COMPLETED, CUST_NAME, DATE_DESIRED}

D^u = {WORK_ORDER, CUSTOMER}

$\langle SC \rangle$ = CUST_NAME = "Jane Doe" AND DATE_DESIRED = "5/10/91"

$\langle M \rangle$ = the exact names and syntax used to specify the elements of A^u, D^u, and $\langle SC \rangle$.

A direct approach for end-user global query formulation may be employed to formulate the above query via model traversal where the user has the choice of picking as few as only some data items in A^u, or as much as other information she or he wants to include.

Model Traversal. Model traversal is a direct approach whereby users utilize the enterprise metadata to articulate the query directly in terms of information models. The technical details and semantics of the heteroge-

neous systems are provided interactively and maybe iteratively. The user will, for example, "pick" the data items and objects directly from the models as opposed to "enter" their names to the query. Every step along the way, on-line assistance is provided to the user to traverse as well as pick. Some common semantic errors due to syntax-based translation or interpretation of information models in conventional "indirect" approaches are avoided, as the user sees and deals directly with the comprehensive and unequivocal system models. The purpose is to allow the user to formulate a global query while traversing the information models.

The specific traversal method is designed according to the characteristics of the model constructs and logical associations among the information models stored in the Metadatabase. The following is a basic model traversal process:

Repeat (for each visit)

Step 1.1 Traverse to the data object identified at the ith visit (d_i^u) and select the data item(s) (a_i^u) from d_i^u for retrieval.

Metadata required: names of the applications, functional views, data constructs, attributes and their associations (i.e., the global data model).

Step 1.2 Specify selection condition(s) ⟨SC⟩ that will be imposed on a selected data item(s).

Metadata required: formats, domains, and operating constraints of the data items

Step 1.3 Resolve semantic ambiguity.

Metadata required: semantic constraints, such as functional dependencies and business rules that describe the intended use of the data.

Until no more intended data attributes (i.e., all elements of A^u are specified) $A^u = \bigcup_i (a_i^u)$ and $D^u = \bigcup_i (d_i^u)$.

Step 2. *Join Condition Determination.* A global query may involve multiple data objects that are stored in one or more local systems. Normally, the user has to specify the equijoin conditions between these data objects for a complete global query. The specification would require detailed understanding of the information model. To relieve the user of this burden, the system with an on-line Metadatabase can perform this job through an automatic join condition determination algorithm using enterprise metadata, as follows:

Step 2.1. Determine the set of data objects, O_k, which contain all the user-selected attributes, A^u and their equivalent data items. This step establishes the maximum space of data objects that the query involves. The metadata required are the associations between the entities/relationships and their data items, plus data equivalence information.

Step 2.2. Determine a minimum set of data objects (O_{min}) that contain all $a_k^u \in A^u$ and $D^u \subseteq O_{min}$.

Step 2.3. Identify the shortest path (SP_{min}) which connects all data objects $d_m \in O_{min}$. The metadata required are the associations between the entities and relationships (global data model).

Step 2.4. Insert join conditions for every connected pair of data objects in D. The results of step 2 for the earlier example would be:

A^s = {ORDER_ID CUST_ID CUST_ORDER_ID}
D^s = {ORDER}
⟨JC⟩ = ORDER_ID = CUST_ORDER_ID AND
ORDER.CUST_ID = CUSTOMER.CUST_ID

Step 3. *Global Query Processing.* A formulated global query GQ = (A, D, ⟨C⟩) is decomposed into a set of local queries {LQ_i} where i indicates the local system. The decomposition is based on the physical whereabouts of the intended data. Each local query LQ_i pertains to one and only one local system.

LQ_i = (LA_i, LF_i, ⟨LC_i⟩) where
LA_i is a set of local items with $\bigcup_i$ (LA_i) = A,
LF_i is a set of local files/base relations/record types/objects, and
⟨LC_i⟩ is a condition expression concerning only the data item(s) that is (are) contained in the local system i.

Step 3.1. Determine all data files which contain $a_j \in A$. The metadata required are the data equivalence, physical storage—such as files—relation tables, and their data items (implementation models).

Step 3.2. Determine a minimum set of data files (F_{min}) from which the query system retrieves data items (A).

Step 3.3. Formulate local query LQ_i for local system i, such that

(a) (lf_i) ($lf_i \in LF_i$) $\wedge$ ($lf_i \in F_{min}$),
(b) (la_i) (lf_i) ($la_i \in LA_i$) $\wedge$ (lf_i contains la_i)
(c) all items involved in ⟨LC_i⟩ are elements of LA_i.

The metadata required are the physical storage methods—such as files—relation tables, and their data items.

Step 3.4. Preserve global join conditions ⟨GJC⟩ such that

⟨GJC⟩ ::= ⟨j_condition⟩ [AND ⟨j_condition⟩]
⟨j_condition⟩ ::= item1 = item2
where item1 and item2 belong to two different systems.

The metadata required are the same as for Step 3.3.

Suppose two local systems—shop floor control and order entry system—were involved in the query in the previous example, then two local queries would result from this step:

LQ_{SFC}:
li_{SFC} = {PART_ID NUM_COMPLETED ORDER_ID}
lf_{SFC} = {WORK_ORDER}
$\langle LC_{SFC} \rangle$ ::= (no condition)

LQ_{OES}:
li_{OES} = {CUST_NAME CUST_ID DATA_DESIRED ORDER_ID}
lf_{OES} = {ORDER CUSTOMER}
$\langle LC_{OES} \rangle$ ::= CUST_NAME = "Jane Doe" AND
DATE_DESIRED = "5/10/91" AND
ORDER.CUST_ID = CUSTOMER.CUST_ID
$\langle GJC \rangle$ ::= WORK_ORDER.ORDER_ID = ORDER.ORDER_ID

Step 4. *Local Query Generation.* A language generator is needed for each distinctive data manipulation language used in the enterprise. A local query is generated using the local language for each LQ. For the earlier example, the query language LQ_{SFC} generated for the shop floor is in Oracle/SQL:

```
SELECT    WORK_ORDER.ORDER_ID, '|',
          WORK_ORDER.PART_ID, '|',
          WORK_ORDER.NUM_COMPLETED, '|'
  FROM    WORK_ORDER;
```

The query language LQ_{OES} generated for the shop floor is in Rdb/Rdo:

```
invoke database filename OES$DIR:OES
FOR A IN ORDER
  CROSS B IN CUSTOMER
    WITH A.DATE_DESIRED = "5/10/91"
    AND B.CUST_NAME = "JANE DOE"
  PRINT "@",
    A.CUST_ID, "|",
    A.CUST_ORDER_ID, "|",
    A.DATE_DESIRED, "|",
    B.CUST_NAME, "|"
END_FOR
```

metadata required: implementation models: local DBMS, local DML, and access path, and security metadata: user's access authority and password.

Step 5. *Query Execution.* A local query, LQ, must be sent via the network to the destination for processing by the local database, and then the result will be sent back.

Step 5.1. A message is produced for each local query generated from step 4 containing destination, priority, and other metadata, in addition to the LQ.

Step 5.2. These messages are transmitted to the appropriate local system by a network administrator/monitor.

Step 5.3. At the local system shell, a message is received by the network administration/monitor and dispatched to the DBMS where the local query is executed.

Step 5.4. Local result is sent for Result Integration by the network administration/monitor as a message. Please note that the above description assumes a networking system using the message methods and possessing local as well as global administration monitoring capabilities. These assumptions are consistent with virtually all network protocols such as TCP/IP, MAP, and TOP.

For a minimum network, the metadata requirements can be satisfied by the previous steps. For more advanced systems, metadata such as priority and alternate sources can be used for flows management and optimization.

Step 6. *Result Integration.* The results of local queries (LQs) must be interpreted and assembled according to the global join conditions (⟨GJC⟩) in Step 3.4. Logically equivalent data items may be implemented differently in terms of format, scale, and encoding in different local systems. In our example, ORDER_ID in WORK_ORDER and CUST_ORDER_ID in ORDER are encoded differently for local processing purposes. Therefore, data conversions must be performed on one or both of them before the results from these two systems can be joined and presented to the user. The metadata required are data equivalence and contextual knowledge: conversion rules and operation rules.

2.2.2 *A Basic Model for On-Line Knowledge.* The basic metadata requirements identified above are organized into a definitive model, as follows, to characterize the knowledge needed for the global query operation.

Definition. Enterprise metadata give rise to the knowledge required in global query formulation and processing for both end-users and application programs.

Knowledge for Global Query Formulation

—*Global data model*: a logical model representing the data resources of the enterprise. Specifically, the (names of) applications or functions, data constructs, and their relationships are needed for model traversal; format and domains of the data items are used for selection condition(s) specifi-

cation; functional dependencies are for integrity checking of the selection conditions, and primary keys and foreign keys are for implicit join conditions determination.

—*Data equivalence*: the knowledge needed in most steps to convert as well as identity/clarify multiple data definitions, including typing, semantic (interpretation of) presentation, and scale.

—*Contextual knowledge*: business rules describing the intended use of the data and the needs of the user, also used for ambiguity checking of the selection conditions.

Knowledge for Global Query Optimization and Decomposition

—*Data equivalence*.

—*Implementation models*: physical storage methods of the logical data items, including the size of the files or relation tables and the like used for query optimization and decomposition.

Knowledge for Global Query Generation

—*Implementation models*: metadata about local data language environments and access paths, used to determine the language generators to use and the heading of a query program.

—*Security metadata*: users' access privileges, used to determine whether the retrieval requests are legitimate, and the passwords are used for obtaining access permission.

Knowledge for Results Integration

—*Data equivalence*.

—*Contextual knowledge*: conversion rules and operating rules needed for resolving conflicting data definitions and computing derived data items.

2.3 The Approach: The Conceptual Model

The above analysis, which applies to the global query problem in general, also defines the overall algorithm and the contents of the Metadatabase for the model-assisted global query approach. Thus, a defining characteristic of the MGQS approach is $\langle M^s \rangle = \langle M \rangle$, i.e., the system provides all of the metadata required. The methods that are required to implement this approach are discussed next.

3. NEW METHODS: UTILIZING THE METADATABASE FOR ON-LINE ASSISTANCE

The execution methods envisioned in the model-assisted GQS are developed below. The Metadatabase can be designed in a number of ways, of course. The particular execution model we developed is based on the particular Metadatabase developed in the past several years, whose structure is detailed by Hsu et al. [1991]. The Metadatabase structure itself is outlined first.

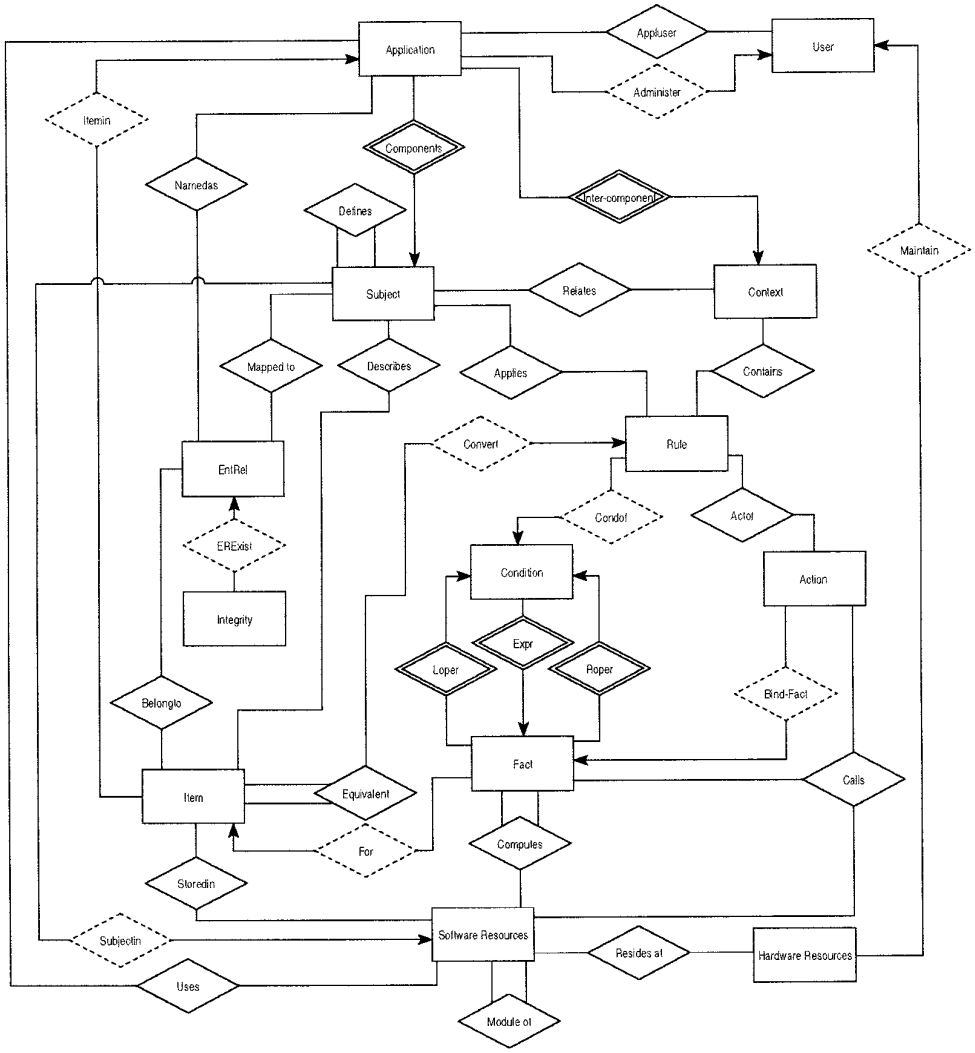


Fig. 2. The GIRD metamodel.

3.1 The Metadatabase Structure: The GIRD Model

The Metadatabase employs a generic metastructure, the Global Information Resources Dictionary (GIRD) model (see Figure 2 and Hsu et al. [1991] for details), for abstracting the enterprise metadata resources (including four categories: functional models, structural models, software and hardware resources, and enterprise/application families) contained in a metadatabase. The model in Figure 2 shows the types of the abstraction (e.g., all subjects from local data models are represented as instances into the SUBJECT metaentity, their attributes ITEM, and so on). Furthermore, all categories of medata are constructed in an integrated manner amenable for implementation using primarily relational technology (i.e., each type can be

mapped to and implemented as a base table). Thus, even though the local databases can be of a variety of classes, from relational to object oriented and flat files, the Metadatabase itself is always of a uniform environment. An object hierarchy defined in EXPRESS for a local engineering database would, for example, be represented as a set of subject tuples and a set of (data) item tuples, along with a few other sets of metadata tuples for inheritance/integrity, equivalence, rules, and so on for incorporation into the Metadatabase. As such, each and every local model is represented and integrated (but not duplicated, nor removed) into the Metadatabase as ordinary metadata tuples populating the structure. This design is based entirely on the logic of the representation for organizing metadata, and hence the metastructure in Figure 2 self-describes the modeling concepts prescribed. When a subject's attributes are recorded generically and globally in the ITEM metaentity separate from the SUBJECT, with their associations recorded as such, the global representation of local models is extensible without losing structural integrity; yet structural limitations are precisely a common problem with traditional cannonical data representations. Therefore, *metadata independence* [Hsu et al. 1992] at the model integration level is achieved, since any change in enterprise models (which are simply metadata instances) would involve only ordinary metadata transactions similar to the usual relational processing and does not require any change to the structure itself, nor reloading/recompilation. The term metadata independence is phrased in the same philosophy as the well-known data independence concept.

The metastructure itself, as well as the global representation of local models as metadata instances, is developed using the Two-Stage Entity Relationship (TSER) method, which is reported elsewhere [Hsu 1985; Hsu et al. 1987; 1993]. TSER provides at the first, functional stage-two basic representation constructs: SUBJECT (similar to objects in object-oriented paradigm) and CONTEXT (rule-based description of process and other contextual knowledge). At the second, structural stage, ENTITY (characterized with singular keys) and RELATIONSHIP (characterized with composite keys) plus two more types of associations signifying special integrity rules (functional and mandatory) are defined. These constructs are in turn defined in terms of DATA ITEM and RULE, where the latter is further decomposed into CONDITION and ACTION, and then into FACT which relates to DATA ITEM. Associating constructs such as expressions are also defined along the way. All these constructs, combined with hardware and software classes of metadata, give rise to the GIRD model. This approach contrasts in a fundamental way with other repository systems in terms of its self-descriptiveness [Hsu et al. 1991] and metadata independence. Other than the ongoing ISO efforts on new IRDS, virtually all previous systems such as the NIST IRDS, IBM Repository, and Digital CDD plus do not support knowledge, nor heterogeneous environments. They all tend to focus on managing software rather than integrating enterprise information, as manifested in their inability to support a global query such as the proposed

MGQS approach. (The above description satisfies the purpose of this article; the full details are provided by Hsu et al. [1991].)

This design provides a few basic properties important to the execution model:

(1) The GIRD model can be implemented in the same manner as a regular schema using a relational DBMS, where the TSER constructs of the model provide design specifications.
(2) Local information models (represented in the Metadatabase as tuples) can be added, deleted, or modified without causing the Metadatabase to restructure or recompile.
(3) All metarelations in the GIRD model (e.g., APPLICATIONS, SUBJECT, DEFINE, CONTEXT, RULE, ITEM, EQUIVALENT, and BELONGTO) are normalized. Thus, they can be managed and processed as a (relational) database implementing the model.
(4) Contextual knowledge is represented in terms of relations as well. A particular class is the equivalence between data items. For instance, the fact that 5/31/94 in the American date format is equivalent to the European 31/5/94 is established through ITEM and EQUIVALENT, with the attendant conversion rules and routines represented through RULE. These rules and routines include any mapping models as needed.
(5) The required metadata as defined in Section 2 are all included in the GIRD model. Thus, Figure 2 shows the high-level semantics of the Metadatabase.
(6) In addition to effecting a full-fledged metadata management facility, the Metadatabase also simplifies the mapping requirements. Since all local databases map directly with the Metadatabase rather than among themselves, the complexity is $O(N)$ as opposed to $O(N^2)$.
(7) The usual concern of semantics loss during local-global mapping is also kept to a minimum, since the mapping is based on metadata constructs as opposed to relying on data structures. While the latter tends to be arbitrary and rigid, the former can be accomplished through rigorous (reverse) modeling at the time when the local databases are represented into the Metadatabase.
(8) The modeling and creation of the Metadatabase follow the tradition of data and knowledge systems analysis and design. Full discussions of a particular methodology and its CASE implementation can be found in Hsu et al. [1992; 1993].
(9) The metadata independence nature of the architecture supports scalability in terms of adding new systems, while its implementation using a regular DBMS assures scaling up in software engineering. When a new system is added, only the system's models (schema) need to be understood and reverse-engineered (or registered) into the Metadatabase using ordinary (meta-) tables transactions. The new metadata are then automatically integrated with those of existing systems through

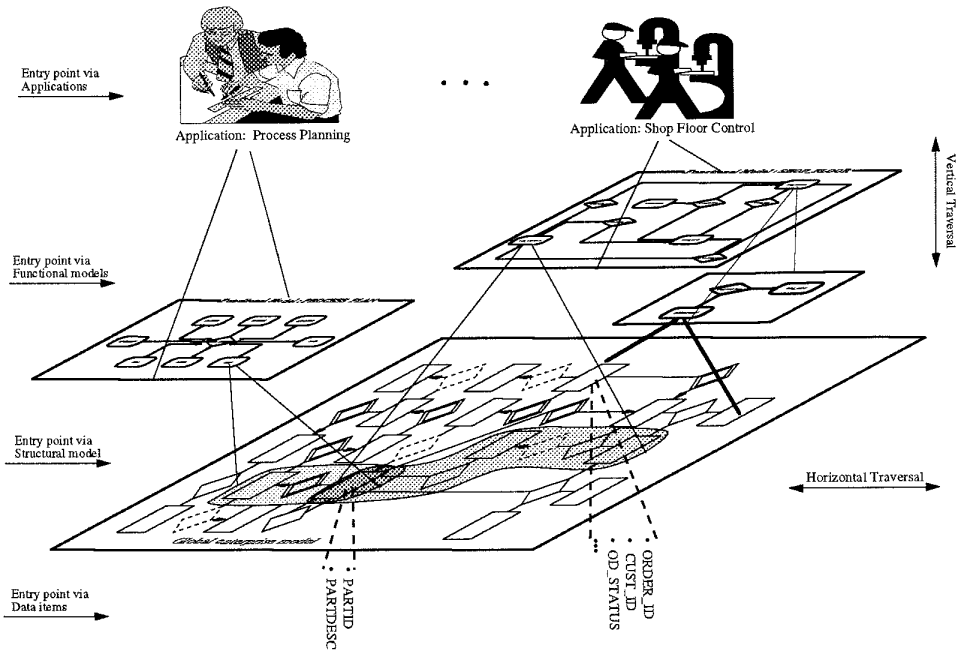


Fig. 3. Model traversal methods.

the GIRD semantics (see Section 6). This property is also discussed by Hsu et al. [1992; 1993].

3.2 Query Formulation

3.2.1 *Model Traversal.* Two basic methods are employed for model traversal: vertical and horizontal (Figure 3). The vertical method specifies the traversal depth cutting across four layers: application systems, functional models, structural models, and data items, each of which can be entered independently before moving up or down therefrom. The horizontal method traverses the entity/relationship (ER) surface of the global model in a network manner. These two methods give rise to a fully connected three-dimensional traversal mode allowing users to formulate global queries from the global information constructs directly using GUI.

The user could page through the global model (i.e., the entire Figure 3) according to the logical associations between these modeling constructs. Depending on the user's experience with the information models, the traversal could be very pinpointing (specifying the exact constructs containing the data items needed) or very general (browsing through a list of data items with little precise specification or through only the applications) or anything in between. The different entry points shown in Figure 3 indicate how a user might choose to perceive and approach the global model. It is interesting to point out that the instances shown in Figure 3 would be stored as instances of the metastructure in Figure 2, and the system would

generate the equivalent GUI of Figure 3 by deriving these instances directly from the Metadatabase. The traversal mode does not assume nor impose any a priori syntax other than the semantics of the constructs used. Thus, the semantics of a query formulated this way is specified completely through the selection of the constructs, their metadata instances, and particular data values.

Selecting E/R and Data Items. The result of a simple selection in global query formulation is equivalent to the projection operation of relational algebra, which reduces the number of columns in a table. For instance, selecting data items a and b from ER1 can be expressed as:

ER1 projected_to a, b: $\{\langle r.a, r.b\rangle | ER1(r)\}$
where r is a row (i.e., tuple) of an ER.

If the result of the selection involves data items contained in multiple ERs, the interpretation would be different depending on whether or not the automatic equijoin determination is used. The automatic determination will lead to first performing equijoins among the ERs and then the projection on the results of the joins. For example, consider a case where data items a and b are selected from ER1, and c and d are selected from ER2. Assuming ER1 and ER2 are directly connected in the global model (i.e., they share a common item (x) as part of their primary key or foreign key), the operation can be expressed as

1. ER1 join ER2 on x: $ER = \{r{+}{+}t - \langle t.x\rangle | ER1(r) \wedge ER2(t) \wedge (r.x = t.x)\}$
   r++t denotes a row made by concatenating row t onto the end of row r
2. ER projected_to a, b, c, d: $\{\langle u.a, u.b, u.c, u.d\rangle | ER(u)\}$.

If the selected ERs are not directly connected to each other, then a connected solution path needs to be determined (see Section 3.2.1) in order to determine the necessary equijoins.

When the equijoin determination is not used, it implies that a Cartesian product (*) between ER1 and ER2 is performed among the ERs to precede the projection operation. The same example would be expressed as

1. $ER1 * ER2$: $ER = \{r{+}{+}t | ER1(r) * ER2(t)\}$
2. ER projected_to a, b, c, d: $\{\langle u.a, u.b, u.c, u.d\rangle | ER(u)\}$.

Selecting Derived Data Items. Besides selecting the persistent (stored) data items, the user can also include derived (run-time) data items in a global query. A derived data item is not stored in any local system, but computed by using the persistent data item(s). Continuing the above example, suppose data item x = f(a,c). The derived item (x) can be selected from either ER1 (ER1 projected_to x) or ER2 (ER2 projected_to x). The result of the selection causes both a and c to be included in the global query:

$\{\langle t.a\rangle | ER1(t)\}$, $\{\langle r.c\rangle | ER2(r)\}$.

The derived data items are defined via business rules in the Metadatabase. The computation of derived data items using a rule processor is discussed in Section 3.5.

Specifying Selection Conditions. Each selection condition (f(a)) imposes a restriction on a data item a. For example, specifying the selection conditions "f(a) $\wedge$ (g(b) $\vee$ h(c))" for an entity/relationship ER1 containing a, b, c would mean

$$\{r|ER1(r) \wedge f(r.a) \wedge (g(r.b) \vee h(r.c)) \}.$$

If, on the other hand, c is contained in ER2, then there are two different actions depending on whether equijoin determination is used. If the equijoin determination is used, it means to first perform an equijoin between ER1 and ER2 on the common item x

$$ER = \{r{+}{+}t{-}\langle t.x\rangle|ER1(r) \wedge ER2(t) \wedge (r.x = t.x) \}$$

and then a selection on ER

$$\{u|ER(u) \wedge f(u.a) \wedge (g(u.b) \vee h(u.c)) \}.$$

If the equijoin determination is not used, it means to perform a Cartesian product between ER1 and ER2

$$ER = \{r{+}{+}t|ER1(r) * ER2(t) \}$$

before the selection operation.

Formulating a Nested Query. A successfully formulated global query can be kept as a run-time view by giving a unique name, which can then be recalled to formulate a nested global query. MGQS supports two different methods of relating run-time views: UNION (+) and NOT EXISTS (−). The user starts with selecting a method and two views to combine. Then the user selects items from the set of common items of these views presented by the system. The result of the UNION of views V1, V2 based on data items a, b (V1 + V2 on a, b,) could be expressed as

$$\{r{+}{+}t{-}\ \langle \text{duplicate columns}\rangle|V1(r) \vee_{a,b} V2(t) \}.$$

The result of V1 NOT EXISTS (i.e., NOT EXISTS works as the relational algebraic operator DIFFERENCE) in V2 based on data items a, b (V1 − V2 on a, b,) could be expressed as

$$\{r|V1(r) \wedge_{a,b} {\sim}V2(t) \}.$$

The above discussions have also shown that the traversal mode includes analogs of the five primitive algebraic operations (i.e., selection, project, product, union, and difference) that define relational completeness, the common yardstick used for the expressive power of traditional query languages [Date 1995]. We might stress that MGQS supports nonrelational queries such as a nested query and an object-oriented query just like it does with the relational, although the Metadatabase itself is primarily relational (and hence is sufficed with relational completeness). The invoking of

local systems that have nested query capabilities or object-oriented languages (e.g., C++) would be done via the local query code generators.

3.2.2 *Query Validation.* MGQS detects and prevents inconsistencies in the selection of join conditions by examining the knowledge in the Metadatabase. Specifically, assistance is provided for the following three potential problems:

(1) *Domain incompatible conditions*: the system will use the properties of data domains to guide a user's entry of values for a selection condition. It will only allow values that conform with the type(s) (real, integer, character, etc.) of the data item(s) selected.
(2) *Semantically inconsistent conditions*: the formal semantic constraints, such as a functional dependency model, are used to detect and prevent semantic inconsistencies.
(3) *Conditions conflicting business rules*: business rules might implicitly nullify or render invalid certain selection conditions that the user intends. The system will perform this consistency checking via the rule-processing method provided in Section 3.4.

3.3 Automatic Completion of Global Query Formulation

The MGQS method determines all of the required metadata that users do not provide to complete the query after model traversal. A key task here is the automatic determination of the join conditions implied in a user's (incomplete) formulation of a query by using the global data model from the Metadatabase. The data items and ERs that are not originally selected for the global query, but that are needed for the joins, will be added to the global query for completion.

The steps of metadata derivation for completing a global query include the following:

3.3.1 *Determining a Minimal Set of Data Objects.* Equivalent(a) is a function that requests the Metadatabase to retrieve all equivalent data items of the specified data item, a. DataObject(A, subject, application) is a function that requests the Metadatabase to retrieve an exclusive set of data objects within the boundary of the specified subject and/or application, such that every resulting data object contains at least one data item $a \in A$.

The Minimum-Set-of-ERs Algorithm

Step 1. Set $O_{min} = D^u$;
Step 2. For (each $a_k^u \in A^u$)
  O_k = DataObject(Equivalent(a_k^u), subject, application);
Step 3. For ((each $o \in O_{min}$) and ($A^u \neq \emptyset$))
  For ((each $a_k^u \in A^u$) and ($A^u \neq \emptyset$))
    if $o \in O_k$
      $A^u = A^u - \{a_k^u\}$;
Step 4. if $A^u \neq \emptyset$ after Step 3 then Step 4.1 and Step 4.2

```
Step 4.1 For ((a_k^u ∈ A^u) and (‖O_k‖ = 1))
           {
           A^u = A^u − {a_k^u};
           Omin = Omin ∪ O_k;
           For each a_1^u ∈ A^u and A^u ≠ Ø
             If (o_k ∈ O_1)
               A_u = A_u − {a_1^u};
Step 4.2 get_minimal_sets(O_min, A^u, results) /* results = { } initially */
         {
         current_best = ‖A^u‖ + ‖O_min‖
         select a_k^u from A^u;
         for ((each o_k ∈ O_k) and (O_k ≠ Ø))
           {
           If A^u − {a_k^u} = Ø))
             If ‖O_min ∪ {o_k}‖⟨current_best
               {
               results = {O_min ∪ {o_k}};
               current_best = ‖O_min ∪ {o_k}‖;
               }
             else If ‖O_min ∪ {o_k}‖ = current_best
               results = results ∪ {O_min ∪ {o_k}};
           else
             get_minimal_sets(O_min ∪ {o_k}, A^u − {a_k^u}, results);
           }
         }
```

3.3.2 *Determining the Shortest Solution Path.* In the context of global query formulation, the global data model is considered as a network G, which is a graph where all nodes (the set N) are connected with links (the set L). The set of selected entities and relationships representing the semantics of a global query defines the set of nodes (N^o) to be connected by the solution path sought. The shortest solution path (SG) that connects all selected entities and relationships denotes the complete formulation of the global query. In the Metadatabase, all connected entities and relationships share some common prime attributes and hence always represent some meaning. In other words, the semantics of a global query is specified implicitly or explicitly by the user's selecting data objects (i.e., SUBJECTs, entities, relationships, and data items), while the shortest (connected) solution path signifies a meaningful query as well as guaranteeing a minimum number of equijoins to be performed.

An algorithm is presented for searching for the SG. Although there exists generic algorithms and even specialized entity-relationship algorithms for the SG problem, a new idea is employed to develop a, hopefully, more efficient SG algorithm to deal with large networks.

The algorithm begins by creating messages: one message for each of the nodes in N^o. A message contains a unique identifier of the message (ID), the cost (cost), and an indicator (from) signifying the preceding node of the

message. When the name of the node (i.e., entity or relationship) in the global data model is unique, this name can be used as the ID; otherwise, the system would generate unique identifiers. The cost of a message represents the number of links that a message has been passed through, from the originating node (i.e., one in N^o) to the current node. Last, "from" contains the ID of the previous node that sent the message.

The algorithm centers around messages. The idea is to send a message from the originating node to all its adjacent nodes, which, in turn, pass each message to new nodes one unit distance farther from their originating node. When two messages with the same IDs reach a node, only the one with the smallest cost is kept. This cycle continues until a node has collected all messages that were originally created; at this point a solution path has been found. The total cost of the path is then calculated by adding all costs of all collected messages. If the total cost of a new solution is less than the cost of the previous solution, a better solution is found. The new total cost is then recorded, and the node is marked as the current root R of the solution. This solution-searching process is continued until the shortest path is found. Two criteria are used to terminate the algorithm: (1) current_cost = nb_nodes − 1 (current total cost equals to the number of nodes in N^o minus one), which means all nodes are directly connected and (2) nb_cycle = current_cost − nb_nodes + 2, where nb_cycle is the number of cycles for message sending. The formal algorithm is given below, which is followed by a proof for the termination conditions.

LEMMA 3.3.2.1. *A lower bound for the best solution (shortest path) that can be found for connecting the nodes in* N^o *has total cost* $C_R = \|N^o\| - 1$, *where* $\|N^o\|$ *is number of elements in* N^o.

PROOF. The best solution occurs only when the root node $R \in N^o$, and all other nodes k ($k \neq R$, $k \in N^o$), are directly connected to R.

Thus, $1_R = 0$, (1 is the distance from a node to the root)

$$1_k = 1, \ \forall k \in N^o, \text{ and } k \neq R$$

$$C_R = \quad 1_n = \quad 1_k + 1_R = \|N^o\| - 1. \quad \square$$

LEMMA 3.3.2.2. *The longest possible distance* (L^m) *between a furthest node* n_1 ($n_1 \in N^o$) *to the root node* R *in a solution with the total cost of* C_R *is* $L^m = C_R - \|N^o\| + 2$.

PROOF. Let n_1 be the furthest node from the root R, and $K = N^o - \{n_1\}$ (Figure 4).

$$C_R = \quad 1_n = \quad 1_k + 1_{n_1}, \quad \text{where} \quad k \in K, \text{ and } k \neq n_1$$

$$\Rightarrow 1_{n_1} = C_R - \quad 1_k$$

From Lemma 3.3.2.1, the smallest possible cost to connect k nodes is $1_k = \|K\| - 1$, where $K = N^o - 1$. Thus $L^m = \text{MAX}(1_{n_1}) = C_R - \|N^o\| + 2$. $\square$

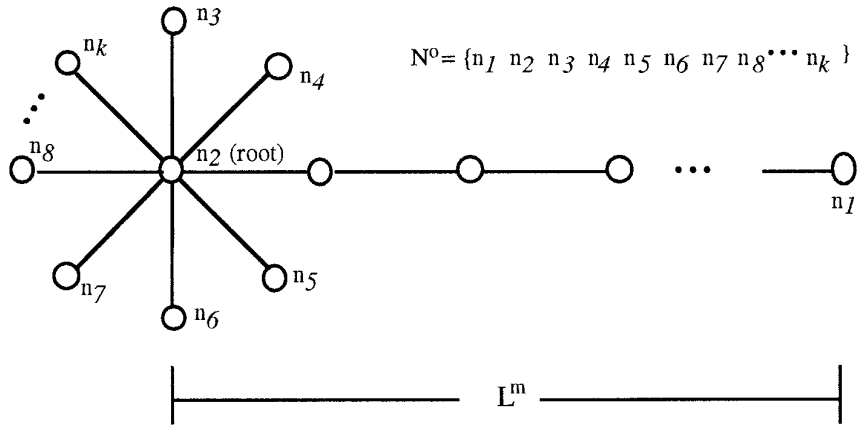


Fig. 4. Determining the number of cycles needed.

THEOREM 3.3.2.3. *The algorithm will reach the shortest path* (*all shortest paths if more than one exist*) *at the end of* i *message-sending cycles, where* $i = C_R - \|N^o\| + 2$.

PROOF. From Lemma 3.3.2.2, the largest distance possible between a node (n_1) to the root (R) is $1_{n_1} = C_R - \|N^o\| + 2$. Each cycle of message sending will pass the message (m_{n_1}) one unit distance closer to the root. Thus exactly i ($i = 1_{n_1}$) cycles are needed for the furthest message m_n to reach R. Also, if a better solution with root R′ exists ($C_{R'} < C_R$), the number of cycles required to reach the solution is $i' = C_{R'} - \|N^o\| + 2 \Rightarrow i' < i$. □

The Shortest-Path Algorithm

```
{
current_cost = a large number;
root = NULL;
create message for each node in N°;
initialize current_cycle;
next_cycle = NULL;
initialize visited_nodes;
nb_cycle = 1;
for ((current_cost > nb_nodes − 1) and
  (nb_cycle ≤ current_cost − nb_nodes + 2))
  {
  for (each message m_i in current_cycle)
    {
    list_of_nodes = get_related_entrel(m_i − >from);
    for (each node n_j in list_of_nodes)
      {
      if (n_j ∈ visited_nodes)
        {
```

```
            add n_j to visited_nodes;
            m_i ->cost = m_i ->cost + 1;
            m_i ->from = n_j ->key;
            add m_i to next_cycle;
            }
          if (m_i not in n_j)
            {
            add message m_i to n_j;
            if (new_best_solution( ))
              {
              root = n_j;
              current_cost = calculated_cost( );
              }
            }
          }
        }
      }
    current_cycle = next_cycle;
    next_cycle = NULL;
    nb_cycle = nb_cycle + 1;
    }
```

Semantic Ambiguity on Solution Paths. Different solution paths represent different semantics. When the shortest-path determination algorithm returns two or more shortest solution paths, it indicates that further interaction with the user is needed to clarify the ambiguity. A straightforward method is to display all the alternative paths (graphically) with suggestions, interpretations, and possibly dialogues. Then, the user will select the correct path from the alternatives. Toward further automation, the MGQS will match the ERs involved in a path with the ERs of the SUBJECTs and user-defined views stored in the Metadatabase. If a match is found, the matching path is likely to be the correct one. Therefore, a recommendation could be made to the user. The idea is to offer more assistance by providing, or acquiring, more information about each path and the matching with SUBJECTs and views. However, the user will have to make the final decision for the correct solution path.

3.3.3 *Equijoin Conditions Determination.* To complete the query, join conditions needed to implement the solution path found must also be determined. The method starts from the root node. It backtracks the message-passing route (i.e., following the value of "from") for each message collected in the root to its originating node (i.e., when from = NULL). Along the route, join conditions are determined for each pair of adjacent nodes. Two routes may share a common section. Therefore, a list of processed adjacent pairs is maintained, and the results from the previous join

determination process between the same adjacent nodes are used to improve performance.

The specific join condition between two nodes (ER1 and ER2) is determined based on the type of connection between them, which is model structure specific. A join condition is added to the GQ when matching items (i.e., two equivalent items) are found from the two nodes. The particular methods for the TSER model used in the MGQS prototype (see Section 4) can be found in Cheung [1991].

3.4 Query Processing

3.4.1 *Query Optimization and Decomposition.* Two heuristics are used for global query optimization: (1) minimizing the number of equijoins between ERs and (2) minimizing the number of local files being accessed. During the query formulation, data items may be selected from SUBJECTs, applications, or the enterprise as a whole instead of explicitly from ERs. Determining a minimal set of ERs that contains all of these data items and provides the shortest solution path will minimize equijoins. Determining a minimal set of files is done in the following way:

Step 1. Determine the set of files (F_k) containing the data item (a_k):
For each $a_k \in A$, the set of data items selected by the user and determined for the join conditions by the system:
F_k = GET_FILE(a_k);
GET_FILE() returns all files containing a_k from the Metadatabase.

Step 2. Determine the minimal set of files that contain all $a_k \in A$:
Same as step 4 in the minimal set of ERs algorithm, but replace ERs with files, O_{min} with F_{min}, O_k with F_k, a_k^u with a_k, and A^u with A.

Step 3. Determine the set of data items A^f that will be retrieved from each file, ($f \in F_{min}$).

Decompose Global Query into Local Queries. The purpose of the decomposition procedure is to decompose the global query into subqueries, such that each subquery concerns only one local system. The decomposition procedure has the following steps:

Step 1. Decompose the global query (GQ) into multiple global queries (GQ_i). Let $A^F = U_f \{A^f\}$, $f \in F_{min}$.
Let GQ = {F_{min}, A^F, $C(A^c)$}, transform the logical expression $C(A^c)$ into disjunctive normal form using the laws of Boolean algebra:
$C(A^c) = C_1(A^c) \vee C_2(A^c) \vee \ldots \vee C_i(A^c)$
where $C_i(A^c)$ is an elementary conjunct of the form $X_1 \wedge X_2 \wedge \ldots \wedge X_i$. The result of this step is GQ_i = {Fmin, A^F, $C_i(A^c)$} (i.e., $GQ = U_i\ GQ_i$).

Step 2. Decompose each GQ_i into local queries.
Step 2.1. Group file f ∈ Fmin by application (local system)

```
For each f ∈ Fmin
  {
  s = GET_APPL(f); /*returns the application that file f belongs to */
  F_s = F_s ∪ {f};
  }
```

Step 2.2. Separate conditions $C_i(A^c)$ by applications.

Let $C_{si}(A^c)$ be the condition expression that concerns only application system s.

$C_i(A^c) = (\wedge_{si}\ C_{si}) \wedge J_i(A^c)$

where $J_i(A^c)$ are join conditions that involve item from two different applications.

The result of this step is a set of local queries (i.e., $LQ_{si} = \{F_s, A^F, C_{si}(A^c),\}$) and the join conditions $J_i(A^c)$.

3.4.2 *Generating Local Queries.* A local query using the local data manipulation language (or local retrieval language for a file system) is generated for each $LQ_{si} = \{F_s, A^F, C_{si}(A^c),\}$. A code generator is designed specifically for a particular DML (e.g., C++ for EXPRESS, SQL, Rdo/Rdb, and dBASE languages) based on the structure of LQ and the syntax of the DML.

3.4.3 *Result Integration*

Step 1. Assemble local results, R_{si}, into the result of global query GQ_i.

$R_i = \{r1{+}{+}r2{+}{+}\ldots|R_{1i}(r1) \wedge R_{2i}(r2) \wedge \ldots \wedge J_i(A^c)\}$

r1 and r2 are the rows in local results $R_{1i}(r1)$ and $R_{2i}(r2)$ respectively, and r1++r2 denotes a row made by concatenating row r2 onto the end of row r1.

Data conversion is needed during these join operations, if the two items involved in a join condition are equivalent data (Section 3.4).

Step 2. final result, $R_{GQ} = \{r|R_1(r) \vee R_2(r) \vee \ldots\}$

Derived data items are computed according to the business rules, at the end of this step. The persistent items used for the derivation will be replaced by the derived data item (Section 3.5).

If the global query GQ is formulated by relating two run-time views using the operators UNION or NOT EXISTS, the result R_{GQ} would be:

(1) UNION views V1, V2 based on data items d: $R_{GQ} = \{u|V1(u) \vee_d V2(u)\}$
(2) V1 NOT EXISTS in V2 based on data items d: $R_{GQ} = \{u|V1(u) \wedge_d \sim V2(u)\}$.

3.5 Knowledge Processing Using a Rule-Based Approach

3.5.1 *Convert Equivalent Data.* Function Join_condition_satisfied( ) returns TRUE if the two rows in the join operation satisfy all conditions in $J_{abi}(A^c)$, and returns FALSE otherwise.

```
Join_conditions_satisfied(rowA, rowB, J_abi(A^c))
  {
  For (each condition (itemA operator itemB) in J_abi(A^c))
    {
    valueA = extract(rowA, itemA); /* returns value of itemA in rowA */
    valueB = extract(rowB, itemB);
    C_rules = find_convert_rules(itemA, itemB);
    If (C_rules ≠ NULL)
      valueA = convert_item(itemA, itemB, valueA, C_rules);
    If (compare(valueA, valueB, operator) == FALSE)
```

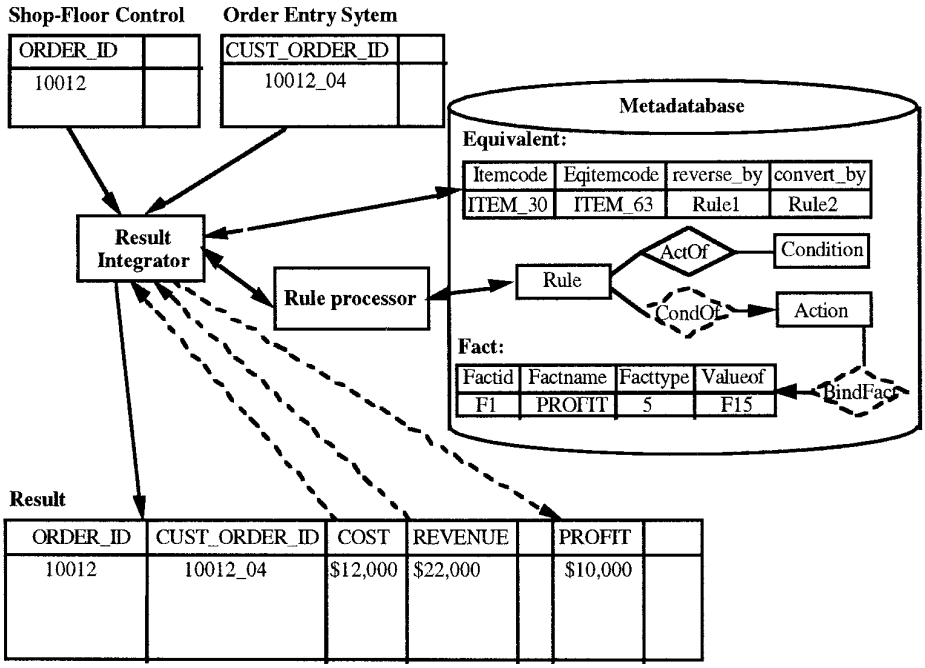


Fig. 5. Data conversion and derivation through a rule-based approach.

```
        return(FALSE);
    }
  return(TRUE);
  }
```

The function find_convert_rules(itemA, itemB) searches the equivalent table of the Metadatabase for the conversion rule(s) (Figure 5) that could convert the value of itemA (valueA) into the format of itemB. As an example, in Figure 6 find_convert_rules(itemA, itemB) returns two rules (i.e., rule2 and rule4), which will first convert the value of itemA to itemC format (using rule2) then to itemB format (using rule4). The function returns "NULL" if no conversion is needed for comparing itemA and itemB.

The function convert_item( ) triggers the rule processor that will search and fire the actions of the conversion rules [Bouziane 1991].

3.5.2 *Compute Derived Data Item.* Function get_involved_items(d_d, D_p) returns the list of persistent items D_p and the action ID (Actid) of the action that binds a value to d_d. For example, as shown in Figure 4, the search of d_p starts from identifying a fact (Factid) that binds a value to PROFIT (i.e., factname = "PROFIT" and Facttype = 5) in the Fact table of the Metadatabase. Using the Factid through the Action table we find all the rules (R) that may cause the binding. Therefore, all the involved persistent items D_p can be identified through the Actions and Conditions that are related to R (see Bouziane [1991, chapt. 9]). D_p is first used for

Itemcode	Eqitemcode	reverse_by	convert_by
itemA	itemC	Rule1	Rule2
itemB	itemC	Rule4	Rule3

Fig. 6. Equivalent table.

global query formulation and retrieval; then the values of D_p (V_{dp}) are used to compute d_d.

Let Result be the result from assembling the local results, and let V_{dd} be the computed result of the derived item.

```
Compute_derived_item( )
  {
  For each row (r) in Result
    {
    ActionID = get_involved_items(d_d, D_p);
    V_dp = extract(r, D_p);
    populate_fact_table(D_p, V_dp);
    V_dd = BChainer(ActionID);
    insert (row, V_dd);
    }
  }
```

The procedure populate_fact_table( ) builds the fact table using the persistent items code and their values V_{dp}. Function BChainer(ActionID) triggers the rule processor, which uses backward chaining to try to reach the goal (ActionID) according to the fact table. The computed value of the derived item is returned by the function.

3.5.3 *Check Business Rules.* Consider the following example. Rule12 and Rule14 are stored in the metadatabase:

```
Rule12:
  If WO_SEQ.WS_ID = '0001'
  Then WORK_ORDER is a Milling job
Rule14:
  If WORK_ORDER is a Milling job And
  WO_SEQ.START_DATE ≠ WO_SEQ.END_DATE
  Then PRINT("Warning: END_DATE should be the same as START_DATE for
  a milling job")
```

If the users select a condition such as

```
WO_SEQ.WS_ID = '0001' AND
WO_SEQ.START_DATE ≠ WO_SEQ.END_DATE
```

then MGQS will detect the inconsistency and issue a warning to the users. This method calls for a forward-chaining inferencing strategy for the rule processor. The consistency checking can be triggered whenever a new

condition is specified or by request (e.g., when a user requests the processing of a global query).

Step 1. Populate the fact table.

Step 2. Activate the forward chainer.

The function FChainer( ) triggers the forward chainer, which will fire rule(s) according to the fact table.

Note: details of all of the functions described in this section can be found in Bouziane [1991] and Cheung [1991].

4. THE IMPLEMENTATION

4.1 The MGQS Architecture and Prototype Method

The general approach discussed above is implemented with particular designs. The Metadatabase and its attendant concurrent architecture [Babin 1993; Hsu and Rattner 1990; Hsu and Skevington 1987; Hsu et al. 1992; 1994] provide the technological basis for the particular MGQS methods.

An MGQS architecture that complies with the basic requirements described in the earlier sections and the idea of model assistance is developed, as shown in Figure 7. It can be conceptually divided into two major components: (1) metadata manager and (2) global query manager. The metadata manager consists of the Metadatabase management system shell, rule processor, metarelation manager, and routine manager [Bouziane 1991]. It acquires pertinent knowledge from the Metadatabase, and thereby it provides model assistance for the global query operation through the user interface. The global query manager, on the other hand, includes the necessary modules, such as a query formulator, processor, translator, and result integrator, that would fulfill the basic requirements of global query operations.

4.2 Metadatabase Query Language

The Metadatabase Query Language (MQL) allows programs to query a collection of autonomous local information system(s) in a nonprocedural way. Its ability, supported by the metadatabase, to accommodate the logical and physical heterogeneities of the local systems distinguishes MQL from other distributed database query languages. The semantics of an MQL query is provided in precisely the same way as the traversal mode discussed in Section 3.2.1. Instead of using the GUI of the MGQS, a syntactical structure is employed in its place.

The major functionalities which MQL provides are the following:

—retrieve metadata against the Metadatabase,

—support queries requiring joins of data from different local systems operating under different schemata,

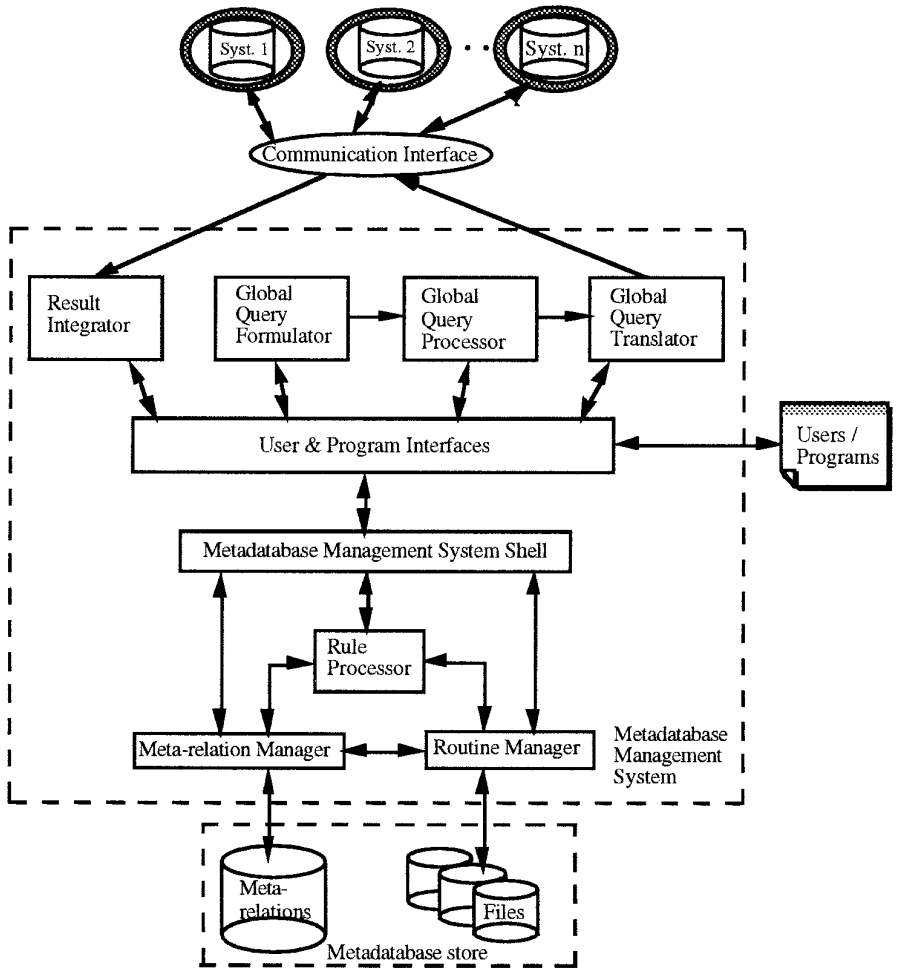


Fig. 7. The architecture of the model-assisted GQS.

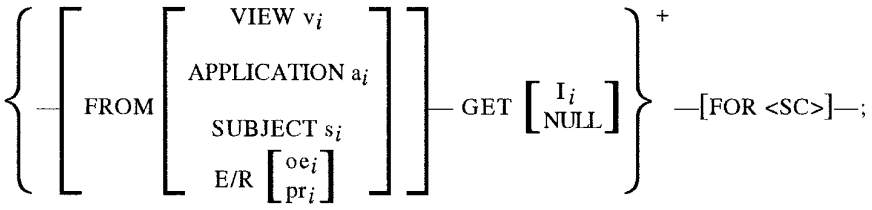
—support queries involving data with multiple definitions (i.e., differing names, formats, units, and/or value coding) within a single system or across different local systems,

—use user-familiar names in queries to qualify data elements in different databases,

—minimize technical details expected of users for the global query formulation while supporting the above functionalities (e.g., the physical locations, local names, and implicit join conditions).

The Syntax. The conceptual structure of MQL is based on the Two-Stage Entity-Relationship representation method. In Figure 8, the GET command is the only command that must be stated in a global query. It is used to specify the list of items for retrieval. I_i denotes the list of item names that are delimited by a blank character "space," and NULL is a

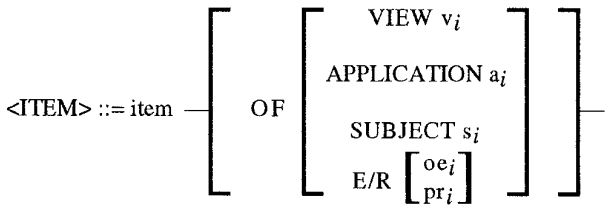
<GQ> ::=



<C> ::= <condition> [<conjoin> <condition>]*

<conjoin> ::= AND | OR

<condition> ::= <ITEM> <bound> [value |<ITEM>]



<bound> ::= = | < | > | <= | >= | ≠

Fig. 8. The syntax of MQL.

symbol to indicate that no data item will be retrieved. In the syntax expression, v_i denotes a run-time view name; a_i denotes an application name; s_i denotes a subject name; oe_i denotes an entity name; and pr_i denotes a plural-relationship name.

The FROM clause is used to indicate where the items are to be retrieved from. There are four ways of using the FROM command, depending on the level of detail a user specifies in the global query: FROM VIEW, FROM APPLICATION, FROM SUBJECT, and FROM E/R. If the items are selected from a user-defined run-time view, the view name must be provided by using the FROM VIEW clause. Otherwise, FROM is optional in this syntax, especially when the user does not know where specific data items can be retrieved from. During the global query formulation, the FROM and GET commands can be used repeatedly until all of the items to be retrieved are specified (see Figure 9 for an example).

Last, the FOR clause is used for specifying the selection condition(s) and join condition(s) (⟨C⟩). The conditions are conjoined by the logical operators AND or OR. Each condition states the binding of an item with a constant value or another item. The logical location of an item involved in a condition can be further specified with the OF clause. The OF clause works in the same manner as the FROM clause.

Global query: "Find part ID and quantity completed for Jane Doe's order which has a desired date of 5/10/91"

```
FROM APPLICATION SHOP_FLOOR GET PARTID NUM_COMPLETED
FROM SUBJECT CUST_ORDER GET DATE_DESIRED
FROM OE/PR CUSTOMER GET CUST_NAME
FOR     CUST_NAME = "Jane Doe" AND
        DATE_DESIRED = "5/10/91";
```

Fig. 9. An MQL example.

Defining Run-Time View. A run-time view is a user-defined information view based on the existing information model. After its creation, the view is kept as a virtual relational table during the global query operation. To define a run-time view, the user uses the command

DEFINE VIEW v_i ⟨GQ⟩;
where ⟨GQ⟩ is a global query in MQL.

v_i is the name given to the new defined view. In essence, view v_i is the result of a global query.

A global query can be formulated based on a run-time view (using the FROM VIEW and OF VIEW clauses) or by relating run-time views using the UNION and EXIST functions (described below). With the MQL syntax, a view can be built recursively on another run-time view. The capability of view definition allows complex (nested) global queries to be formulated using the MQL.

Standard Functions. Standard functions are provided in MQL, including

AVG( )—average of the values (numeric values only) in the column,
COUNT( )—number of values in the column,
MAX( )—largest value in the column,
MIN( )—smallest value in the column, and
SUM( )—sum of the numeric values in the column.

Three functions DISTINCT, GROUP BY, and ORDER BY are provided for arranging the resulting tables. DISTINCT eliminates tuples with duplicate values in the column. GROUP BY arranges tuples into groups such that within any one group all tuples have the same values for the column. ORDER BY sorts tuples (ascending or descending) according to the values of a column.

Last, UNION and NOT EXISTS are used in MQL to formulate a nested global query by relating two run-time views. The syntax of UNION and NOT EXISTS is as follows:

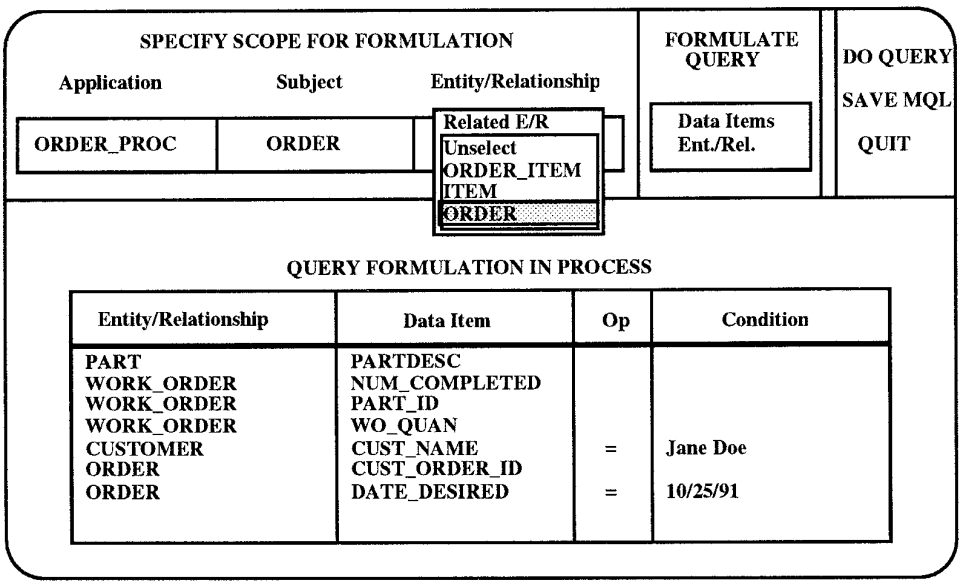


Fig. 10. MGQS user interface for global query formulation.

$\langle \text{view}_i \rangle$ UNION $\langle \text{view}_j \rangle$ on I_i
$\langle \text{view}_i \rangle$ NOT EXISTS $\langle \text{view}_j \rangle$ on I_i

I_i is a list of data item(s) contained in both view $_i$ and view$_j$. UNION works as the union operator of set theory; the result of a UNION operation is a table with all unique data items from both views. Rows with duplicate values on I_i are eliminated from the result of the UNION. NOT EXISTS works as the difference operator of set theory. The result of a NOT EXISTS operation is a table containing rows from view $_i$, such that there is no value of its I_i that matches any values of I_i in view$_j$.

4.3 Implementation of the Design

A prototype model-assisted global query system based on the architecture described in the previous section is built for the purpose of concept verification [Babin 1993; Bouziane 1991; Cheung 1991; Rattner 1990]. This prototype is written in the C language and has two versions running respectively on a Micro VAX workstation and an RS6000 workstation. It provides functions of global query formulation, global query optimization, local query translation, result integration, and on-line intelligence using a rule-based approach.

The prototype MGQS user interface is designed according to the model traversal method described in Figure 3. Figure 10 illustrates the implementation in a window environments. The upper left window entitled "SPECIFY SCOPE FOR FORMULATION" is used for model traversal (browsing), where horizontal movement realizes horizontal navigation, and a pull-down menu enables vertical immersion for each subtitle (i.e., Application, Subject, and

Entity/Relationship). Pertinent objects (application, subject, and entity/relationship) are displayed in the entry fields.

The center window, "FORMULATE QUERY," is for selecting data objects (two types: data item and entity/relationship) into the global query. This way, a user can explicitly include an entity/relationship in a global query for semantic purposes even though no data item will be retrieved from it. The progress of the formulation is displayed in the window on the lower half of the screen. Selection conditions for the query can be specified via the "QUERY FORMULATION IN PROCESS" window.

The "Do Query" button in the upper right side of the screen will execute the formulated global query. The "Save MQL" button will translate the formulated global query into the syntax of the Metadatabase query language (see Section 4.2) and save it into a file. Last, the "QUIT" button will exit from the MGQS system.

The formulated global queries will be decomposed into optimized subqueries through the global query processor. The query translator then generates local DML code for each subquery before it is sent to the local system for processing. Finally, as local results arrive, they are interpreted and assembled by the result integrator.

5. AN EMPIRICAL CASE STUDY

5.1 The Heterogeneous Environment

The computer-integrated manufacturing (CIM) facility at Rensselaer is used as a test bed for this research. Four functional systems are employed—namely, a process-planning system developed as a dBASE III+ application on an IBM PC/AT, a shop floor control system running under the PC/Oracle DBMS, an order entry system implemented as a Rdb/VAX application, and a product database designed in the EXPRESS language and implemented on an object-oriented ROSE platform running on an IBM RS6000. The metadatabase is the fifth system residing on an IBM RS6000. This setup provides a heterogeneous, distributed environment for MGQS prototyping and verification.

The Enterprise Model. A rigorous modeling and reverse-engineering (for paradigm translation) process was engaged to create the enterprise information models and populate the Metadatabase. The process, while beyond the scope of this article, is reported by Hsu et al. [1992; 1993]. This process yielded a global data model and other information models created expressly for local systems, and resulted in the enterprise metadata stored into the Metadatabase. Thus, all local systems were not disturbed at all by this process; only a separated Metadatabase was added to the environment. The global data model for this case is shown in Figure 11. When stored into the Metadatabase, this model is logically corresponded to the structural layer in Figure 3 while physically populated as instances into EnRel, Integrity, and Item metaentities in Figure 2.

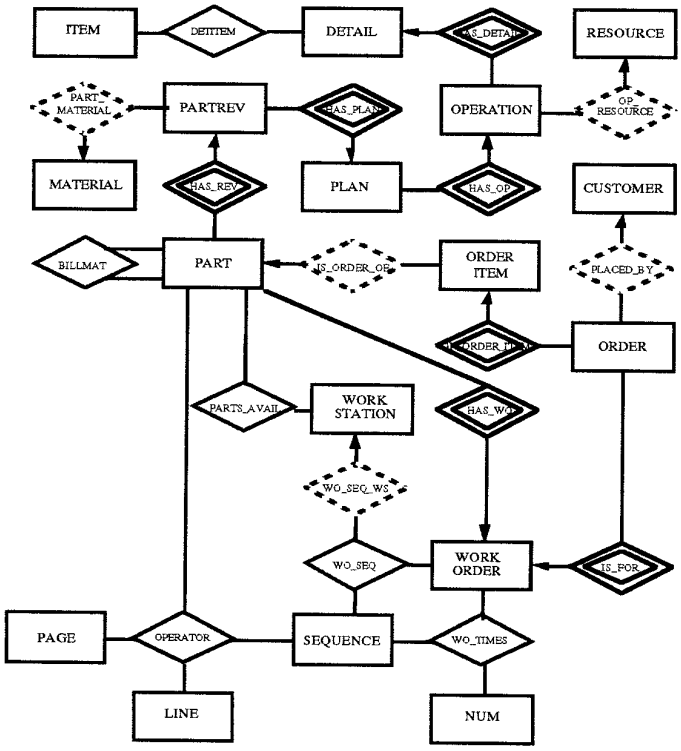


Fig. 11. The global model for CIM at Rensselaer.

It is worth noting that MGQS not only does away with imposing data standardization or naming convention on any of the local systems, but also preserves high local autonomy by dealing changes among the local systems dynamically via updating data equivalence, the conversion rules, and other metadata (as pertinent). These changes to the Metadatabase require only ordinary database transactions, since the GIRD model features metadata independence [Hsu et al. 1991].

5.2 A Global Query Scenario

Consider the following global query example:

> "Find the customer order ID, part ID, part description, and quantity completed for Jane Doe's order which has a desired date of 10/25/90."

This request involves data from three application systems, i.e., order entry, shop floor control, and process planning. (Note that the user posing this query may not know that the query traverses multiple systems.) The user engages the system through the model-assisted dialog menus and windows to formulate the query while paging through the models. The model traversal starts with selecting an application as an entry point which is very general. As shown in Figure 12, the shaded lines show the particular

Query: " Find the customer order ID, part ID, part description, quantity, and quantity completed for Jane Doe's customer order which has a desired date of 10/25/90."

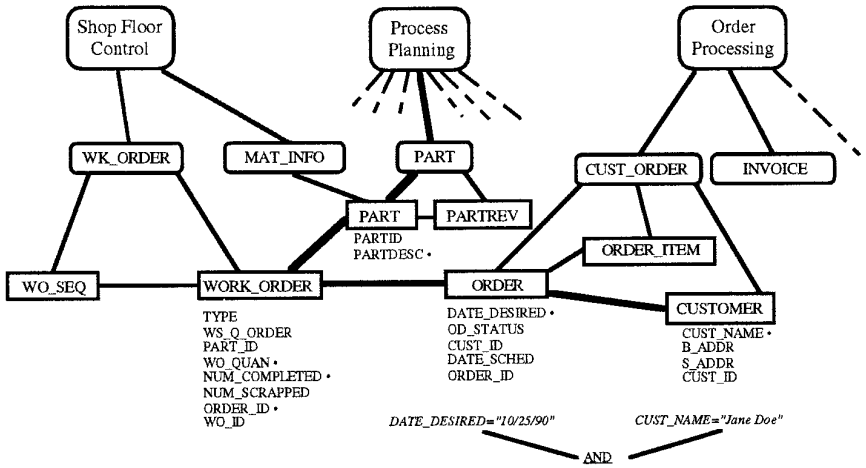


Fig. 12. Query example: Model traversal tree.

path followed by the user to formulate this query. Note the relationship between the data items (listed at the bottom of the model) and the systems in which they are stored (shown at the top). The user does not need to know the information model; it is presented here as an illustration of the global data model network stored in the Metadatabase. This network is used by MGQS itself to guide the query formulation process. The items tagged with asterisks have been requested in the query. Also observe that the rounded boxes in the figure represent SUBJECTs and the Squared boxes the corresponding Entities and Relationships of the enterprise model. During the process the user marks the data fields needed along with any conditional statements. (In this query example, DATE_DESIRED = "10/25/90" and CUST_NAME = "Jane Doe") The formulation is completed when all the items are selected and when the selection conditions are specified.

Once the formulation is completed, MGQS will first determine the solution path for the global query from the enterprise model. Equijoin conditions are then inserted to the formulated global query according to the path. The join conditions for this example are:

PART.PARTID = WORK_ORDER.PART_ID
WORK_ORDER.ORDER_ID = ORDER.CUST_ORDER_ID
ORDER.CUST_ID = CUSTOMER.CUST_ID

Note the synonyms PARTID vs. PART_ID and ORDER_ID vs. CUST_ORDER_ID are identified by the prototype system. Also, PARTID in PART, PART_ID in WORK_ORDER, CUST_ORDER_ID and CUST_ID in ORDER, and CUST_ID in CUSTOMER are not selected during the formulation; therefore, MGQS will have to add these items into the global query. Then,

PART_I	ORDER_I	WO_QUAN	NUM_COM	CUST_ID	CUST_ORDER	DATE_DESIR	CUST_NAME	PART_DESC
PZ1	12345	0	1	JD001	12345_15	10/25/90	Jane Doe	PUZZLE#1
PZ51	12346	0	3	JD001	12346_16	10/25/90	Jane Doe	PUZZLE#51

Fig. 13. Final result of the global query.

MGQS decomposes the global query into a set of optimized, locally bound subqueries using the implementation models in the Metadatabase. Each subquery requires data retrieval from one and only one local system. MGQS will disseminate these requests using the native query languages of the local systems involved. These local queries are sent out across the network to the respective local systems to be serviced. A rule-based shell system for each and every local systems was included in the prototype to provide data update and other active database functionalities [Babin 1993; Hsu et al. 1992]. This aspect, however, is beyond MGQS and is not needed in the discussion here. It suffices the purpose to simply consider that there is a shell at each node serving as the interface between the metadatabase and these local systems.

Each local application system's shell receives a request to process the local query. Upon reception, the subquery is executed by calling the local DBMS with the file containing the generated local query. The results are then sent back to the MGQS where they are assembled logically and presented to the user (Figure 13).

As shown in Figure 13, the values of the ORDER_ID and CUST_ORDER_ID are coded differently. Data conversion is required in order to complete the join operation correctly. MGQS calls upon the rule processor to fire the proper conversion rule containing contextual knowledge, which manipulates raw data or triggers the desired procedures for the conversion before the join.

A Test for the Metadatabase Query Language. The same global query example is used for testing the Metadatabase query language (MQL). Two formulations for the example using MQL are shown in Figures 14 and 15. Both formulations provide the same result as in Figure 13. Note the formulation in Figure 15 is less precise. The user does not pinpoint the entities and relationships that are involved in the global query. Instead, only the involved applications are specified. The MGQS fills in the necessary information by consulting the contents of the Metadatabase. Minimally, a user will only need to specify the data item(s) and the necessary selection condition(s) for a global query. The rest of the query processing is the same as described before.

5.3 Discussion

It is worth noting that the OES was originally implemented as a VAX/VMS file system. New OES using the Rdb, a relational DBMS on a Micro VAX, was designed and built after the MGQS prototype had been functionalized. The adaptation to the new system for MGQS was done rather quickly. First, we removed the old OES models from the contents of the Metadatabase. Then the TSER models of the OES, obtained from the top-down

```
FROM OE/PR PART GET PARTID PARTDESC
FROM OE/PR WORK_ORDER GET ORDER_ID WO_QUAN NUM_COMPLETED
FROM OE/PR ORDER GET DATE_DESIRED
FROM OE/PR CUSTOMER GET CUST_NAME
FOR DATE_DESIRED = "10/25/90" AND
    CUST_NAME = "Jane Doe";
```

Fig. 14. MQL formulation.

```
FROM APPLICATION PROCESS_PLAN GET PARTID PARTDESC
FROM APPLICATION SHOP_FLOOR GET WO_QUAN NUM_COMPLETED
FROM APPLICATION ORDER_ENTRY GET DATE_DESIRED
CUST_NAME
FOR DATE_DESIRED = "10/25/90" AND
    CUST_NAME = "Jane Doe";
```

Fig. 15. MQL formulation.

modeling effort, were incorporated automatically into the Metadatabase (i.e., consolidating the new model with the existing enterprise model and populating the Metadatabase with the new models). Finally a customized code generator for Rdo, a DML for Rdb, is programmed. The total effort for the changes essentially was completed when the code generator was completed. This situation illustrates the idea of changes under metadata independence as mentioned before.

In the whole, the CIM facility at Rensselaer provides an environment with a reasonable complexity to test the prototype MGQS and the model assistance concept. All major objectives (i.e., information sharing, local system autonomy, and model assistance) of the MGQS are achieved and proven to be feasible with the prototype system. Most of the envisioned functionalities (with the exception of ambiguity checking and derived item querying) are implemented. In addition, the data conversion capability has demonstrated a use of the contextual knowledge and rule processor for providing on-line intelligence. This same method can be applied to the function of ambiguity checking and derived data querying.

There are, however, the needs to investigate the performance issues especially when run-time data conversion and other rule operations are involved. The empirical study did not provide sufficient observations to draw scientific conclusions from. Nonetheless, the preliminary results do not appear to suggest that these operations are a bottleneck. One reason might be the fact that data conversion in MGQS is performed at local nodes in a truly distributed manner by local systems using concurrently the common global representations as the target. Most other rules are processed in a similar concurrent design using the Metadatabase's distributed

Site	Database	Data Objects	Attributes	Format/Domain
A	PJ	PROJECT	JNO	INTEGER
			J_LOC	CHAR(40)
		SUPPLY	SNO	INTEGER
			PNO	CODE2
			JNO	INTEGER
B	S	SUPPLIER	SID	INTEGER
			S_LOC	CHAR(40)
C	P	PART	PNO	CODE1
			PNAME	CHAR(20)
			LENGHT	REAL

Fig. 16. An example of three systems A, B, and C.

shells for local systems (see Babin [1993] and Hsu et al. [1992; 1994] for details, which are beyond the scope of this article).

6. ANALYSIS

The MGQS approach is justified in this section through a comparison with previous results with respect to the objectives (Section 2.1) of information sharing in heterogeneous distributed environments. A comment on the generalizability of the MGQS results is also provided.

6.1 Metadata Modeling versus Schema Integration

A key element in conventional distributed databases is schema integration. All databases are logically structured and controlled under a single integrated schema in a homogeneous environment [Cardenas 1987; Rem 1991; Thomas et al. 1990]. The user would be able to share information across all databases as if there were only one classical centralized database. Therefore, local transparency and conflict resolution are achieved through enforcing an integrated schema under a single data model. The primary problem with this approach, however, is lack of local system autonomy and adaptability.

To illustrate, consider the example in Figure 16. In this example SNO and SID are two synonyms of the same logical attribute (one in data object SUPPLY of database PJ at site A and the other in data object SUPPLIER of database S at side B). Further, PNO in PART and the PNO in SUPPLY are structured differently with different domains CODE1 and CODE2, respectively. The integrated schema could be developed as shown in Figure 17 (assuming a relational system). In the integrated schema, the two conflicting definitions are reconciled by changing attribute name SID to SNO in SUPPLIER and the value domains of PNO in both SUPPLY and PART to type integer. The local systems will recompile and reload so as to implement the changes according to the integrated schema.

```
CREATE TABLE PROJECT (
      JNO  INTEGER,
      J_LOC  CHAR(40));
CREATE TABLE SUPPLY (
      SNO  INTEGER,
      PNO  INTEGER,
      JNO  INTEGER);
```

```
CREATE TABLE PART(
      PNO  INTEGER,
PNAME  CHAR(20),
      LENGTH  REAL);
CREATE TABLE SUPPLIER (
      SNO  INTEGER,
      S_LOC  CHAR(4O));
```

Fig. 17. Integrated schema for the example using DDBMS approach.

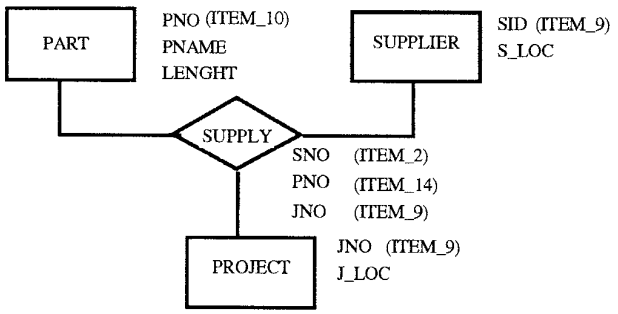
Figure 18 shows the global model that MGQS would employ for this example. Note that each data item in the global model is assigned with a unique item code. For instance, the two PNOs in PART and SUPPLY are logically the same attributes with the same name, but reside in different local systems, and hence they have different item codes. However, the attribute JNO in the database PJ is only a single data item, since, albeit shared by both SUPPLY and PROJECT, it is the same attribute with the same name in the same system; therefore they have the same item code. The fact that PNO in PART and SUPPLY are coded differently is represented as a tuple in **Equivalent** along with the conversion rules Rule1 and Rule2. Synonyms SID and SNO are also reflected in **Equivalent**. Therefore, no change is required of the local systems using this method, while any names could be used by users in any systems to address the attributes and obtain globally consolidated results. The same process is utilized to effect, equivalent with respect to types, semantic formats (e.g., dates) and user-dependent presentation. Note that the process does not require an integrated global schema. Concerning adaptability, such as modifying, adding, or deleting (local) models, any such changes to the enterprise models can be simply handled as metadata transactions against the Metadatabase without affecting its schema (the GIRD model). This, as mentioned in Section 3.1, is the significance of metadata independence. Most previous results, some of which avail the similar local system autonomy as discussed in the above example, do not satisfy metadata independence.

6.2 MQL versus MSQL

The MSQL [Litwin et al. 1989; 1990] facility is an extension of SQL for manipulating data managed by the multidatabase. The major results include (1) additional commands for multiple databases and (2) enhanced naming convention for data objects. Applying MSQL to the same global query example in Figure 12 would yield the following result:

```
Create Multidatabase CIM (SHOPFLOOR PROCESS_PLAN ORDER)

Use CIM
Select  SHOPFLOOR.WORK_ORDER.PART_ID,
        SHOPFLOOR.WORK_ORDER.ORDER_ID,
        SHOPFLOOR.WORK_ORDER.QUAN,
        SHOPFLOOR.WORK_ORDER.COMPLETED,
```



Equivalent

Itemcode	Eqitemcode	reverse_by	convert_by
ITEM_10	ITEM_14	Rule1	Rule2
ITEM_2	ITEM_9	NULL	NULL

RULE

Rname	rtype	description
Rule1	M	convert item_14 to item_10
Rule2	M	convert item_10 to item_14

Fig. 18. Global model for the example using the MGQS.

```
        PROCESS_PLAN.PART.PARTDES,
        ORDER.ORDER.DATE_DESIRED, ORDER.CUSTOMER.CUST_NAME
From    SHOPFLOOR.WORK_ORDER, PRO-
        CESS_PLAN..ORDER, PROCESS_PLAN.PART, ORDER.CUSTOMER
Where   PROCESS_PLAN.PART.PARTID = SHOPFLOOR.WORK_
        ORDER.PART_ID
And     SHOPFLOOR.WORK_ORDER.ORDER_ID = ORDER.ORDER.
        ORDER_ID
And     ORDER.CUSTOMER..CUST_ID = ORDER.ORDER.CUST_ID
And     ORDER.ORDER.DATE_DESIRED = "10/25/93";
```

The above example is evidently more complicated than using MQL (see Figures 14 and 15). The reason is simple: MSQL does not utilize a metadata facility for global query formulation. Users have to provide all the technical details, including precise location of data items, data equivalence, and join conditions. Furthermore, two equivalent data items, PARTID in PART and PART_ID in WORK_ORDER, in a join condition must have the same format in order for MSQL to integrate the results.

6.3 MGQS versus Multibase

Multibase [Dayal and Hwang 1984; Smith et al. 1981] is one of the pioneering research efforts and arguably the most well known practical system in the field of heterogeneous, distributed DBMS. A single query language (DAPLEX) based on the functional data model is employed for information retrieval. Preserving local system autonomy and resolving data

incompatibilities are the major design objectives for the Multibase. Thus, it is a natural reference point for MGQS to be compared with.

Multibase has three levels of schema: (1) a global schema at the top level, (2) an integration schema and one local schema for each local system at the middle level, and (3) one local host schema per local database at the bottom level. The global schema, local schemata, and the integration schema are all defined according to and in terms of the functional data model. Each of the local host schemata is translated into a local schema, and the integration schema is used to describe the so-called integration database containing information needed for integrating results from local databases (i.e., information about mapping between conflicting data definitions). The local schemata and integration schema are then mapped into global schema.

To provide a single query language for information retrieval from heterogeneous systems, Multibase has to synchronize multiple data definitions at the global schema level. Also, multiple data definitions for the logically identical data are not allowed. This is the classical case of integrated schema as discussed in Section 6.1. In contrast, MGQS supports a single global query environment by consulting the Metadatabase instead of imposing restrictions via the global model. Multiple data definitions are allowed, and they are stored in the metadatabase along with the conversion methods.

6.4 MGQS versus Other HD_DBMSs

Other heterogeneous, distributed DBMSs such as Federated databases [Sheth and Larson 1990] have proposed similar three-level schema structures as Multibase to achieve local system autonomy. The basic difference from the latter is, in order to promote an open architecture, these approaches do not enforce a single global model. Instead, several logically related subsystems are grouped together with an integrated model (referred to as external schema) developed for them. Multiple external schemata are coexisting in the global system. This approach provides more flexibility for integration design and reduces the scope of schema integration by cutting down the number of subsystems involved in an integrated model. However, since each subsystem may participate in multiple external schemata that are custom developed, evolving the overall structure, such as incorporating a new subsystem to the environment, is still fundamentally more complicated.

Moreover, there are no metadata management facilities used. Global query formulation relies, therefore, on an extended query language and the user's technical knowledge of the subsystems and their external schemata. A minimal extension of the language is to allow relation names to be qualified with database names in case of naming conflicts. To address an attribute, the user has to know precisely which relation of what database the attribute belongs to. For instance, PNO of PART is referred to as P.PART.PNO. Thus local system transparency is not completely supported in these approaches.

6.5 MGQS versus Other Metadata-Supported Systems

Increasingly, more global query systems have come to the same conclusion of utilizing metadata to resolve some well-known problems in query processing. The driving issues include semantics, optimization path selection, and result integration, which correspond respectively to steps 1.3, 2 (particularly 2.3), and 6 of the MGQS definitional algorithm in Section 2.2.1. Some metadata management facilities may also be used in the forms of a knowledge base or a repository. Their main difference from the MGQS approach lies in the scope of metadata and the metadata independence of the management facilities. A prime example is the Composite Information Systems (CIS) [Siegel and Madnick 1991; Wang and Madnick 1989; 1990] of MIT. It is one of the first works to emphasize the significance of metadata and use it to resolve some semantic conflicts in query translation and result integration. Since there are no formal knowledge methods included in its metadata facility used, the scope and functionality would be affected. In addition, CIS assumes all local systems and the global schema are relational.

The Carnot approach of MCC (Microelectronics and Computer Technology Corporation) [Collet et al. 1991] differs from the CIS approach in its global schema construction and the use of metadata representation model—namely, the Cyc knowledge base. This approach makes the global schema much easier to construct and maintain. However, Collet et al. state, there are two major problems that the Carnot approach has not yet resolved—i.e., (1) how to integrate the results returned from subsystem queries and (2) how to develop a graphical entity-relationship representation of the global schema and an intelligent interface for specifying queries. Both of these are direct results of the Cyc knowledge base design. The MGQS approach facilitates both, due in part to the scope and the structure of the Metadatabase (see Figures 2 and 3, and Sections 3.5 and 6.1).

In contrast to CIS and Carnot, the recently reported KIM (Knowledge-base Information Manager) Query System [Ferrara 1994] provides an iconic ER-based user interface for global query formulation, but does not seem to address such issues as system evolution, optimization path, and result integration. Again, although KIM features a repository similar to Carnot, its design does not support a full vision of enterprise metadata.

6.6 Conceptual Evaluation of the MGQS Approach

The model-assisted global query approach provides several important functionalities for sharing information in heterogeneous distributed environments. The defining and enabling characteristic is the Metadatabase implementing the global model in a metadata-independent design. Based on the discussions in the above sections, these properties are categorized into five areas, as discussed below.

Maintain Local System Autonomy Using the Metadatabase. As mentioned above, MGQS employs a Metadatabase for identifying and resolving the heterogeneities among these systems. Instead of imposing an inte-

grated schema over all the local systems, the Metadatabase approach allows the local systems to keep their environments insulated and to control their own data definitions and everything. The functional model, structural model, implementation model, and the associations among these models of each local system are independently represented, consolidated, and stored in the Metadatabase. Thus, their manipulation and management are as easy as any ordinary data in a database.

Allow Direct and Visual Query Formulation from Enterprise Metadata. A global query is formulated by selecting the data constructs while browsing the global enterprise information models. Regardless of the entry point chosen, the user will eventually lead to the needed data constructs by paging through the global model without noticing the boundary of local systems. This is a high level of direct manipulation (of information objects) that researchers have advocated for a cognitive user interface. The constructs used (see Figure 2) are clearly compatible with a graphical representation and hence support visual methods for query formulation.

Provide On-Line Intelligence and Assistance for Challenging Tasks such as Semantics, Path Selection, and Result Integration. MGQS provides on-line knowledge to facilitate both global query formulation and processing. During query formulation, the pertinent information contents and semantics of the heterogeneous systems are either provided interactively to the user, or MGQS utilizes them to automate certain tasks for the user (see Sections 2 and 3). The on-line knowledge provided to the global query processing is transparent to the user. They include the automatic decomposition and optimization of global query and the recognition and conversion of equivalent data resources across local systems. Combined, the approaches provide a broad range of metadata support covering the entirety of the definition discussed in Section 2.2. They are sufficient for certain difficult tasks, including semantics, path selection for query optimization, and result integration using data equivalence knowledge (see above comparisons). Since the Metadatabase contains both enterprise knowledge resources and data models, its extent of metadata is unique and is capable of providing significant on-line intelligence and assistance.

Include Rule-Based Knowledge Processing for Extensibility. MGQS acquires contextual knowledge from the knowledge model of the Metadatabase to provide on-line intelligence for the global query operation. Two representative examples of contextual knowledge are business rules, which describe the intended use of data, and operating rules, which facilitate decision processes across systems. Global queries involving derived items is another example on the use of contextual knowledge. A rule base model and an inference engine are part of the Metadatabase system. New rules concerning MGQS can be relatively easily incorporated into its architecture and execution model.

Achieve Open-System Architecture through Metadata Independence. The MGQS approach provides an open-system architecture that is flexible for

adding, deleting, or modifying a local system in the integrated enterprise. The property is referred as metadata independence in Section 3.1. For instance, to remove the shop floor control system from the CIM enterprise in Section 5, no change is required of the MGQS. The metadata pertaining to the shop floor system will be removed from the Metadatabase through ordinary transactions (i.e., deleting tuples from the metarelations). Accepting a new subsystem to the integrated enterprise requires primarily a modeling effort, which is no more than that required by conventional approaches. However, the addition of this new model to the existing global model is merely a matter of performing, again, ordinary Metadatabase addition transactions. A documentation of this modeling process is provided, for example, by Hsu et al. [1992; 1994].

6.7 The Generalizability of the MGQS Approach

How much of the MGQS approach can be utilized by other systems than the prototype at Rensselaer, which makes heavy use of a particular Metadatabase and a particular modeling method TSER? The answer turns out to be correlated in steps with the development from Sections 2 to 4. The conceptual model in Section 2 is most general and generic. Any system may be able to develop its own metadata facility (be it a knowledge base, a repository, or a metadatabase) utilizing the metadata requirements specified in the model. At the next level, the particular MGQS execution model is based on the particular Metadatabase. Although its design is rooted in TSER, the GIRD model is simply a specification for creation of the metadatabase. Therefore, the model may be implemented in any system supporting the usual relational-compatible facilities. The third level, the most specific, involves TSER modeling and the Metadatabase management system. The full repetition of the results reported in Sections 5 and 6 will require all elements in Sections 2, 3, and 4. However, these elements are available in the open literature and can be adopted by the general readership.

7. CONCLUSIONS AND CONTRIBUTIONS

The model-assisted global query system accomplishes information sharing in heterogeneous, distributed environments. The goals of the approach and its conceptual model were discussed in Sections 1 and 2, justified in Sections 5 and 6, while the execution methods were established in Sections 3 and 4.

In particular, MGQS contributes a direct method to end-user query formulation through on-line assistance using metadata (Section 2). It allows the user to articulate directly in terms of information models with which they are familiar. The pertinent information contents and data semantics of the heterogeneous multiple systems are provided interactively to the user, thereby further alleviating the technical complexities and semantic ambiguity during formulation. In a similar manner, some technical support is also afforded, including diagnosis and feedback of query

formulation according to the business rules and other contextual knowledge in the system. This direct method contributes in its own right to the conceptual foundations of graphical and interactive user interface technology.

New methods that utilize the on-line knowledge (or metadata) for major global query processing tasks are also developed. They encompass the areas of global query optimization and decomposition, query translation, and result integration. The knowledge needed for these tasks (e.g., implied data items, shortest solution path, and join conditions; local system access paths; query construction across local database schemata; and data equivalency and conversion) is automatically derived from the Metadatabase. Without such on-line intelligence and assistance, the required knowledge would have to either be supplied by users at run/compile-time or be predetermined at design time through schema integration and other standardization approaches.

This article has resolved some interoperability issues of heterogeneous multiple information systems through offering an alternative to the approaches that rely on schema integration. Schema integration is a major source of technical complexity of heterogeneous distributed DBMS at both design time (efforts and restrictions) and run-time (mappings and architectural overheads). Additional knowledge, such as conflicting or alternating data definitions and their resolutions, is also supported and stored in the Metadatabase for MGQS. Conversion is done through a rule processor firing conversion rules. As a result, the only global modeling effort required of this approach is the development of the enterprise information model itself. It is still challenging, especially concerning knowledge acquisition; but it nevertheless avoids the excessive complexity of integration at the schemata level that would ensue on the enterprise model in the case of conventional approaches.

This research has also identified and characterized a model for the notion of "on-line intelligence and assistance" in end-user query interface and query processing in terms of the enterprise metadata. It is also shown through MQL, its query language, that programming query languages can also benefit from the on-line intelligence and assistance. As we have shown in Section 4, the syntax and the technical details of MQL are an improvement over existing global query languages. The concept and methods were tested with the prototype system using the CIM facility at Rensselaer.

Further research is currently underway to enhance MQL and MGQS methods, especially by incorporating additional rule-oriented capabilities into the system. MGQS is also being employed in to facilitate global data management and event-based information flow management in Metadatabase research. New progress might come from applying the basic methods to other information modeling paradigms (than TSER). Finally, information visualization [Hsu and Yee 1993] will be a natural extension to MGQS, where an agent may be developed to personalize the Metadatabase, and a visual/virtual environment can replace the GUI-based user interface used presently.

REFERENCES

AFSARMANESH, H. AND MCLEOD, D. 1989. The 3DIS: An extensible object-oriented information management environment. *ACM Trans. Inf. Syst. 7,* 4 (Oct.), 339–377.

ANGELACCIO, M., CATARCI, T., AND SANTUCCI, G. 1990. QBD*: A graphical query language with recursion. *IEEE Trans. Softw. Eng. 16,* 10, 1150–1163.

AZMOODEH, M. 1990. BRMQ: A database interface facility based on graph traversals and extended relationships on groups of entities. *Comput. J. 33,* 1, 31–39.

BABIN, B. 1993. Adaptiveness in information systems integration. Ph.D. dissertation, Decision Sciences and Engineering Systems Dept., Rensselaer Polytechnic Inst., Troy, N.Y.

BENJAMIN, A. J. AND LEW, K. M. 1986. A visual tool for managing relational databases. In *Proceedings of the IEEE International Conference on Data Engineering.* IEEE, New York, 661–668.

BERNSTEIN, P., GOODMAN, N., WONG, E., REEVE, C. L., AND ROTHNIE, J. B., JR. 1981. Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Syst. 6,* 4 (Dec.), 602–625.

BATINI, C., LENZERINI, M., AND NAVATHE, S. B. 1986. A comparative analysis of methodologies for database scheme integration. *ACM Comput. Surv. 18,* 4.

BLASER, A. 1988. *Natural Language at the Computer: Scientific Symposium on Syntax and Semantics for Text Processing and Man-Machine-Communication,* A. Blaser, Ed. Springer-Verlag, New York.

BOUZIANE, M. 1991. Metadata modeling and management. Ph.D. dissertation, Computer Science Dept., Rensselaer Polytechnic Inst., Troy, N.Y.

BUNT, H. C. 1988. Natural language communication with computers: Some problems, perspectives, and new directions. In *Human-Computer Interaction: Psychonomic Aspects,* G. C. Veer and G. Mulder, Eds. Springer-Verlag, Berlin, 406–442.

CARDENAS, A. F. 1987. Heterogeneous distributed database management: The HD-DBMS. *Proc. IEEE 75,* 7.

CHEUNG, W. 1991. The model-assisted global query system. Ph.D. dissertation, Computer Science Dept., Rensselaer Polytechnic Inst., Troy, N.Y.

CHUNG, C. W. 1990. DATAPLEX: An access to heterogeneous distributed databases. *Commun. ACM 33,* 1 (Jan.), 70–80.

COLLET, C., HUHNS, M. N., AND SHEN, W. M. 1991. Resource integration using a large knowledge base in Carnot. *IEEE Comput. 24,* 12 (Dec.), 55–62.

DATE, C. J. 1995. *An Introduction to Database Systems.* 6th ed. Addison-Wesley, Reading, Mass.

DAYAL, U. AND HWANG, H. 1984. View definition and generalization for database integration in MULTIBASE: A system for heterogeneous distributed databases. *IEEE Trans. Softw. Eng. SE-10,* 6, 628–644.

FERRARA, F. M. 1994. The KIM query system: An iconic interface for the unified access to distributed multimedia databases. *ACM SIGCHI Bull. 26,* 3 (July), 30–39.

FARDINER, M. M. AND CHRISTIE, B. 1987. *Applying Cognitive Psychology to User-Interface Design.* John Wiley and Sons, New York.

GOULD, J. D. AND LEWIS, C. 1985. Designing for usability: Key principles and what designer's think. *Commun. ACM 28,* 3 (Mar.), 300–311.

GREENBERG, S. AND WITTEN, I. H. 1985. Adaptive personalized interfaces—A question of viability. *Behav. Inf. Tech. 4,* 1, 31–45.

GYSSENS, M., PAREDAENS, J., AND GUCHT, D. V. 1990. A graph-oriented object model for database end-user interfaces. In *Proceedings of ACM SIGMOD.* ACM, New York, 24–33.

HARTSON, H. R. AND HIX, D. 1989. Human-computer interface development: Concepts and systems for its management. *ACM Comput. Surv. 21,* 1 (Mar.), 5–92.

HEROT, C. 1980. Spatial management of data. *ACM Trans. Database Syst. 5,* 4, 493–513.

HIRSCHMAN, L. 1989. Natural language interfaces for large-scale information processing. In *Integration of Information Systems: Bridging Heterogeneous Databases,* A. Gupta, Ed. IEEE Press, New York, 308–314.

HSU, C. 1985. Structured databases system analysis and design through entity-relationship approach. In *Proceedings of the 4th International Conference on the Entity Relationship Approach.* IEEE Computer Society Press, Los Alamitos, Calif., 56–63.

HSU, C. AND RATTNER, L. 1990. Information modeling for computerized manufacturing. *IEEE Trans. Syst. Man Cybernet. 20,* 4, 758–776.

HSU, C. AND SKEVINGTON, C. 1987. Integration of data and knowledge model in manufacturing enterprises: A conceptual framework. *Manuf. Syst. 6,* 4, 277–285.

HSU, C. AND YEE, L. 1993. Model-based visualization for enterprise information management. In *Proceedings of the 4th Annual Conference on AI, Simulation and Planning in High Autonomous Systems.* IEEE Computer Society Press, Los Alamitos, Calif., 324–327.

HSU, C., BABIN, G., BOUZIANE, M., CHEUNG, W., RATTNER, L., AND YEE, L. 1992. Metadatabase modeling for enterprise information integration. *J. Syst. Integration 2,* 1, 5–37.

HSU, C., BABIN, G., BOUZIANE, M., CHEUNG, W., RATTNER, L., RUBENSTEIN, A., AND YEE, L. 1994. The metadatabase approach to integrating and managing manufacturing information systems. *J. Intell. Manuf. 5,* 333–349.

HSU, C., BOUZIANE, M., RATTNER, L., AND YEE, L. 1991. Information resources management in heterogeneous distributed environments: A metadatabase approach. *IEEE Trans. Softw. Eng. SE-17,* 6 (June), 604–625.

HSU, C., PERRY, A., BOUZIANE, M., AND CHEUNG, W. 1987. TSER: A data modeling system using the two-stage entity-relationship approach. In *Proceedings of the 6th International Conference on the Entity Relationship Approach.* IEEE Computer Society Press, Los Alamitos, Calif., 461–478.

HSU, C., TAO, Y., BOUZIANE, M., AND BABIN, G. 1993. Paradigm translations in integrating manufacturing information using a meta-model: The TSER approach. *J. Inf. Syst. Eng. 1,* 3, 325–352.

HUTCHINS, E. L., JOLLAN, J. D., AND NORMAN, D. A. 1986. Direct manipulation interfaces. In *User Centered System Design,* D. A. Norman and S. W. Draper, Eds. Lawrence Erlbaum, Hillsdale, N.J., 118–123.

KRISHNAMURTHY, R., LITWIN, W., AND KENT, W. 1991. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data.* ACM, New York, 40–49.

LITWIN, W., ABDELLATIF, A., ZEROUAL, A., AND NICOLAS, B. 1989. MSQL: A multidatabase language. *Inf. Sci. 49,* 59–101.

LITWIN, W., MARK, L., AND ROUSSOPOULOS, N. 1990. Interoperability of multiple autonomous databases. *ACM Comput. Surv. 22,* 3, 267–293.

LODDING, K. N. 1983. Iconic interfacing. *IEEE Comput. Graph. Appl. 3,* 2, 11–20.

MOTRO, A. 1990. A tolerant and cooperative user interface to databases. *IEEE Trans. Knowl. Data Eng. 2,* 2 (June), 231–246.

NILAN, M. S. 1992. Cognitive space—Using virtual reality for large information resource management problems. *J. Commun. 42* (Autumn), 115–135.

NORCIO, A. F. AND STANLEY, J. 1989. Adaptive human-computer interfaces: A literature survey and perspective. *IEEE Trans. Syst. Man Cybernet. 19,* 2 (Mar./Apr.), 399–408.

RATTNER, L. 1990. Information requirements for integrated manufacturing planning and control: A theoretical mode. Ph.D. dissertation, Dept. of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Inst., Troy, N.Y.

REDDY, M. P., PRASAD, B. E., AND REDDY, P. G. 1989. Query processing in heterogeneous distributed database management systems. In *Integration of Information Systems: Bridging Heterogeneous Databases,* A. Gupta, Ed. IEEE Press, New York, 264–277.

REM, S. 1991. Guest editor's introduction: Heterogeneous distributed database systems. *IEEE Comput. 24,* 12 (Dec.), 7–11.

RICH, E. 1984. Natural-language interface. *IEEE Comput. 17,* 9 (Sept.), 39–47.

ROBERTSON, G. C., CARD, S. K., AND MACKINLAY, J. D. 1993. Information visualization using 3D interactive animation. *Commun. ACM 36,* 4 (Apr.), 56–71.

SHETH, A. P. AND LARSON, J. A. 1990. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Comput. Surv. 22,* 3, 183–236.

SHIPMAN, D. W. 1981. The functional language DAPLEX. *ACM Trans. Database Syst. 6,* 1, 140–173.

SHYY, T. M. AND SU, S. Y. W. 1991. A high-level knowledge base programming language for advanced database applications. In *Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data.* ACM, New York, 338–346.

SIEGEL, M. AND MADNICK, S. 1991. A metadata approach to resolving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases.* VLDB Endowment Press, Saratoga, Calif., 133–146.

SMITH, J. M., BERNSTEIN, P. A., DAYAL, U., GOODMAN, N., LANDERS, T., LIN, K. W. T., AND WONG, E. 1981. Multibase integrating heterogeneous distributed database systems. In *Proceedings of the AFIPS NCC.* Vol. 50. AFIPS, Montvale, N.J., 487–499.

STONEBRAKER, M. 1988. Introduction for user interfaces. In *Readings of Database Systems,* M. Stonebraker, Ed. Morgan Kaufmann, San Mateo, Calif., 337–339.

STONEBRAKER, M. AND KALASH, J. 1982. TIMBER: A sophisticated relation browser. In *Proceedings of the 8th International Conference on Very Large Data Bases.* VLDB Endowment Press, Saratoga, Calif., 1–10.

THOMAS, G., THOMPSON, G. R., CHUNG, C.-W., BARKMEYER, E., CARTER, F., TEMPLETON, M., FOX, S., AND HARTMAN, B. 1990. Heterogeneous distributed database systems for production use. *ACM Comput. Surv. 22,* 3, 237–266.

WANG, R. AND MADNICK, S. 1989. Facilitating connectivity in composite information systems. *Data Base* (Fall), 38–46.

WANG, R. AND MADNICK, S. 1990. A polygen model for heterogeneous database systems: The source tagging perspective. In *Proceedings of the 16th International Conference on Very Large Data Bases.* VLDB Endowment Press, Saratoga, Calif., 519–538.

ZLOOF, M. M. 1977. Query-By-Example: A data base language. *IBM Syst. J. 4,* 324–343.

Received October 1993; revised July 1994, February 1995, and August 1995; accepted August 1995