An overview of the multidatabase system MRDSM

W. Litwin

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 374

AN OVERVIEW OF THE MULTIDATABASE SYSTEM MRDSM

Witold LITWIN

Mars 1985

AN OVERVIEW OF THE MULTIDATABASE SYSTEM

MRDSM

Witold LITWIN

INRIA, B.P 105, 78153 Le-Chesnay, France

ABSTRACT.

MRDSM is a prototype relational multidatabase system developed within SESAME project at INRIA. The system provides functionalities for management of collections of relational databases. Such systems will be highly needed since many databases are now available. The system is an extension of the well known MRDS database system (Multics Relational Data Store). We overview informally the main MRDSM functionalities and the corresponding implementation techniques. Most of these functionalities are yet unknown to other systems.

PRESENTATION DU SYSTEME MULTIDATABASE

MRDSM

Witold LITWIN

INRIA, B.P 105, 78153 Le-Chesnay, France

RESUME.

MRDSM est un prototype de système relationnel multibases, développé dans le projet SESAME à l'INRIA. Ce système offre des fonctionnalités permettant de gérer des collections de bases de données relationnelles. De tels systèmes seront nécessaires, car de nombreuses bases de données sont désormais disponibles. Le système est une extension du système de gestion de base de données MRDS (Multics Relational Data Store), bien connu. Nous présentons informellement les principales fonctionnalités de MRDSM et les techniques d'implementation correspondantes. La plupart de ces fonctionnalités sont encore inconnues des autres systèmes.

1. INTRODUCTION

The idea in the database approach to data management is that data to be managed together should constitute a database /ANS75/. Formal aspects and some verbalism taken apart, a database is a set of jointly managed files. Joint management basically consists of two functionalities : (i) one may define multifile (interfile) dependencies, (ii) one may simultaneously manipulate several files. These functionalities lacked to file management systems. Files in those systems were, in this sense, independent. The need for joint management appeared when files became widely used. Many users (applications) began then to have data of interest in several files. One had then to face problems such as combination of data in different files into user defined structures, mutual consistency of data in different files, redundancies, privacy,... As long as the facilities for joint management are then not provided, the user has to solve the corresponding problems by himself. This may be annoying, since the problems are complex.

Depending on the data model, database files are called tables, relations, record types,.... Any database is defined in *intension* by its conceptual schema, shortly schema /ANS75/. The schema is also a partial *extension* of the database name. Data in the database, also called content, are an *extension* of the schema. Content and schema constitute the conceptual level. This (logical) level hides the internal levels that constitute the database implementation. The whole database, is a model of some real universe. Basically, the universe is shared in some way by a community of interrelated users (applications). Typically, it is an enterprise /ANS75/.

A concept central to the database approach is the one of a privileged user called (database) administrator. The administrator defines the schema and, in this sense, the whole database. During this prework, usually called data integration process, the administrator is expected to remove inconsistencies, redundancies, data type differencies,... that could appear among independent files (or even existed in files used by the applications prior to the database creation). Also, he is expected to define privacy requirements and optimal internal files. If application data existed in independent files, they are recreated in database files. One time the database set up, the administrator may further tune it, through changes to logical as well as internal levels.

All these capabilities mean that the administrator **overpowers** and, in this sense,

controls all users. This transfer of power is nevertheless assumed beneficial. At first to the **community as a whole**. Next, hopefully, to **particular** users. The community interests are basically **identified** with those of its **managers** (the ones of the enterprise).

The system that manages the database is called database (management) system (DBS). A DBS may manage **independently** several databases. Also, many databases are now independently accessible through computer networks. In particular, videotex systems, like Prestel, Telelel or Telidon etc. provide databases about almost everything (cinemas, restaurants, banking, trains, airlines, law,...) at disposal of virtually everybody. Not very surprisingly for someone aware of the database concept origin, many users start to have data of interest in **several** databases /MOU81/, /LIT82/. For reasons largely similar to those encountered earlier by file users, this situation calls for some possibilities of joint management of **databases**.

In order to deal with such situations one may first envisage to reapply the database approach. This means that one envisages to integrate all involved data into a single database. However, in contrast to the situation when data were in independent files, it seems frequently impossible to envisage a redefinition of data in new, more efficient, internal files. One has to rather use as **implementation** the **existing databases**. The conceptual schema is then typically called global schema. The single database defined by this schema, through integration process performed by some global administrator, is a kind of distributed database. It is physically distributed when data are at different sites. Else, it is only logically distributed. A non-distributed (classical) database is then called a centralized database. Users do not see any difference between a distributed database and a centralized one with the same schema. For instance, there is no difference at the conceptual level between a distributed relational database and a centralized one. Thus one works with a distributed relational database as one would work with a classical relational database.

However, one may easily observe that this approach may not suffice. This, precisely because, as years ago the files, the databases become widely spreaded. Therefore, no matter how many databases constitutes a distributed one, a user will frequently find a **distinct**, centralized or distributed, **database** and data of interest in **both**. Next, the task of definition of global schema may obviously be very complex when many databases are involved. Furthermore, one may simply not wish **any** global administrator, even if a distributed database could technically be created. More

reasons are discussed later on.

More general approach may then be to admit that, at least sometimes, one needs to manage jointly **several** visibly distinct databases, i. e. without global schema. These databases may be centralized as well as distributed. Since the idea in the database approach is that data to be managed together should constitute **one** (centralized or distributed) database, the envisaged approach is an **extension** of the database approach. Since, by the same token, database systems are intended (and effectively designed) **only** for joint management of data in a database, one needs a **new** type of systems. New functionalities to be provided by such systems should basically correspond to (i) and (ii), since many of the underlying problems are obviously quite similar. However, since one deals now not only with files, but with whole databases, the concept of **file** should be basically extended to the one of a **database**.

This approach to data management was called multidatabase approach. Its overall goal is to be a **more general** methodology for database design, **better adapted to user needs** than the database approach alone. It was developed in /LIT79-83/ (see also /WIE83/). New functionalities were called multidatabase functionalities. The corresponding systems were called multidatabase systems. The term itself was introduced in /LIT82/. In earlier papers we spoke about systems for logically distributed data without global schema. The term is now quite spreaded /BRE84/, /DAY83/, /DAY84/, /LEF84/,....

The corresponding research started within SIRIUS project on distributed databases /LIT82/. It now continues in SESAME project at INRIA, specifically devoted to multidatabase systems. The project goal are the corresponding design techniques. This includes first determination of **fundamental** multidatabase functionalities. For this purpose one particularly analyzes limitations of **basic concepts** of the database approach and develops **more general ones**. Special care is exercised on **best concordance** of new proposals with general user needs. Next, one searches for efficient implementation techniques. The outcomes are validated through design of a prototype relational multidatabase system MRDSM. This system is a generalization of the well known relational database system MRDS (Multics Relational Data Store) /MUL82/. Most of MRDSM functionalities are yet unique to this system.

Below, Section 2 enters more into the details of the principles of the multidatabase approach. Then, Section 3 informally overviews the multidatabase functionalities

provided by MRDSM. Section 4 discusses some points relative to the implementation. Section 5 concludes the overview.

2. THE MULTIDATABASE APPROACH.

2.1 The principle.

The principle of the multidatabase approach is that data to be managed and, in particular, manipulated together, should constitute **one or more** centralized or distributed database. One may then need to know **several** conceptual schemas. Then, one may need to define dependencies between **some databases**. Finally, one may need to formulate joint manipulations of data in **different** databases. For instance, when one wishes to retrieve restaurants and cinemas at the same street from two, centralized or distributed, relational databases RESTAURANTS and CINEMAS /LIT82/. Or, when one wishes to update both databases in one query when the name of a street changed.

Joint manipulations of different databases were called multidatabase queries. More examples of multidatabase queries will be shown later. Deeper discussion may be found in /LIT84b/ and /WON84a/.

2.2 Application domain.

The application domain of the multidatabase approach includes of course the one of the database approach. In addition, it includes all cases of data to be managed through distinct schemas and, in particular, manipulated together. This case should be typical :

- of large sets of databases on sites or networks. A global schema may in this case be envisaged for a reasonably small subset. However, especially for the network case, it is unlikely that an administrator will even only **understand** all schemas in a reasonable time. No one is typically able to understand even all data of only one enterprise.

- when administrators do not wish any global administrator, because of the corresponding constraints.

- when administrators model differently the same universe and wish to keep their models. Data type and/or value incompatibilities may then impede any useful global schema. Especially, since there is not yet general solution to the **update problem** and

it is even strongly doubtful that any such solution exists.

- in particular, when an administrator prefers some data type, values or physical data structures in a database, **in spite of** inconsistencies, redundancies or performance inefficiencies at the community (interdatabase) level. This corresponds to a **fundamental** inversion of priorities with respect to the database approach. In particular, there is no more reasons for the global integration process, since its goal is exactly opposite.

- when frequency of multidatabase queries is too small for the effort of the global schema definition.

- when frequency of changes of database schemas covered by the global one and/or of the number of such databases is too rapid (these changes must lead to the ones of the global schema and/or to the ones of the mapping between this schema and the database schemas).

On the other hand, it may be **advantageous** to create more than one database, provided existence of multidatabase queries. This, even when a centralized or distributed database may be envisaged. This case may occur :

- when one has to manage data of quite different nature. It is a natural tendency to do not integrate such data, but rather to separate them. Provided however the possibility of multidatabase manipulations from time to time.

- when one wishes to perform sophisticated operations. Such operations have tendency to be too slow on large databases.

- when one wishes to know the source of manipulated data that in occurrence is the **database name**. The subjective importance attached to data values may depend on that. For instance, many persons have a preferred restaurant guide.

- especially, when one wishes to keep personal data distinct from any other. For instance, when one wishes to separate private data about restaurants from those contained in a public restaurant guide.

- when one wishes to totally control some data, especially personal ones. By analogy to /LIN81/, one may speak about total database autonomy.

Summing up, the multidatabase approach on the one hand includes the database approach. On the other hand, it applies to situations where the database approach fails. Finally, it applies to situations when both approaches may work, but, provided

the existence of multidatabase queries, one prefers more than one database.

2.3 Background motivations.

These motivations appear particularly well from the analogy at the cover of /ULL83/. Although probably partly involuntarily, this analogy is highly representative, of "vices and virtues" of the database approach. The database is a roast chicken. The administrator is naturally the cook. Other users have no access to the chicken itself. They (only) perceive different odours (views).

In this analogy, the intention in the multidatabase approach is to allow a user to freely choose and mix odours of different chickens. Especially, to allow him to know who cooked the chicken, since, as someone said, all cooks are equal, but some are more equal than others. For a cook, it the right to cook typically differently. In particular, it is a recognition of importance of freedom to cook one's own (personal) chicken. Especially, since one has then the opportunity to taste not only odours, but the chicken itself. Note that one usually prefers the latter opportunity.

More seriously, one background motivation for the multidatabase approach is the belief that wider scale integration is frequently either constraining to users /HAM79/, or **useless** or even **impossible**. Main reason for that is that the idea of integration seems to be mainly founded on the assumption of existence of a **unique** (global, total) **truth** (reference), **same for all users**. This unique truth should be found by the administrator and then should be kept pure (consistent) through his power and the DBS. Different particular users' truths may then only be (partial) **derivations** from the unique one. View mechanisms are intended for this purpose.

While this assumption looks reasonable for quite small universes, it seems too restrictive for larger ones in general, and for the multidatabase environment in particular. It seems that one should rather consider then the existence of **multiple** truths, **relative** (subjective) to **different users** and/or to particular database administrators (different observers of some universe). These truths may be **incompatible**, (appreciations of a restaurant for instance). As in the relativity theory, there may then be **no unique truth**. If one tries then to **enforce** some unique truth, it may be considered as false or useless by **any** user (if a restaurant is bad for one user and good for another, the appreciation "average" may be considered as false or useless by both). In particular, there may be no way to derive user truths, leaving users unsatisfied.

These reasons, **and not technical ones**, seem already most responsible for the

current failure of the concept of a very large database (one per enterprise). It does not seem so most appropriate to use similar ideas for the management of even larger collections. In contrast, **unnecessity** of unique truth and priority to **multiple** truths, as defined by distinct schemas of even the same universe, seems a **natural advantage** of the multidatabase approach.

Some background motivations are also relative to ultimate consequences of the idea of wide scale integration. They may appear sad and, might be, dangerous. Uniform landscape that one may imagine as consequent to some very large global schema, may strangely resemble an orwellian one. The global administrator may look like Big Brother. Especially, since he should enforce the **community** needs over **user** needs, the community being represented by its **managers**. It does not seem the hazard that probably the first book on the subject is /KAL84/. Note also, that an enhancement of control over users, was already frequently presented as particularly attractive aspect of the database concept, when one marketed this concept to entreprises. Fortunately, wide integration stayed until now a dream.

2.4 Main concepts.

2.4.1 Multidatabase.

A multidatabase is a set of databases or of multidatabases where :

(1) - one may perform multidatabase manipulations,

(2) - eventually, one may define multidatabase (interdatabase) dependencies.

A particular, although optional, kind of multidatabase dependency is the multidatabase name. For instance, a multidatabase that includes databases about restaurant guides may be named R-GUIDES /LIT84b/. Databases on a DBS, in particular on any well known relational DBS, **are not** a multidatabase. No DBS provides indeed neither (1) nor (2).

The functionalities (1) and (2) are straightforward generalizations of (i) and (ii), characteristic of the concept of a database. However, a multidatabase is defined by a **collection** of schemas, instead of **one** conceptual schema. By the same token, a multidatabase has basically **no administrator**. If multidatabase dependencies are (explicitely) defined, they mainly result from ponctual (local) agreements among database administrators or users. In particular, it may be that **no one** has the knowledge of the whole set of definitions of data and dependencies.

Note that users perceive a multidatabase differently of a database, since they see

several schemas. Note also that, with respect to the concept of a database, the one of a multidatabase gives more importance to data **manipulation** than to data definition. This trend characterized in fact also the database approach. However, only since the relational model was proposed.

First, one may consider a relational database as a set of (flat) **files** respecting even only (ii). Next, the whole model may be seen as postulating quite **elementary** data structures, versus **powerful** (assertional) data manipulation languages. Earlier models postulated in contrast **a lot of** schema structuring, in particular with respect to interfile dependencies definition, and quite **elementary** (navigational) manipulation languages. (ii) was then not possible with these models, since a navigational language manipulates only one record and so one file at the time. It is clear that the enormous success of the relational model results mainly from the idea of (ii). It is then natural, while conceiving new general concepts, to take at once to account the postulates of this model.

The relational model is, for us, representative of the dynamic (data) integration principle. In contrast, earlier models represent the static integration principle. The static integration largely needs **human** (administrator) prework and the corresponding power transfert. The dynamic integration relies in contrast mostly on sophistication of **data manipulation** capabilities. The way we conceived the concept of a multidatabase expresses in particular the conjecture that, in the multidatabase environment, **dynamic integration will prevail over the static one.**

2.4.2 Multidatabase system.

A multidatabase system (MDBS) is a system providing to users the following functionalities /LIT82/, /LIT83/:

(a) - all those of a DBS,

(b) - a language for multidatabase manipulations,

(c) - eventually, a language for multidatabase dependencies definition.

A general architecture for an MDBS is described in /LIT83/. This architecture takes to the account also other particular functionalities that one may require from an MDBS. Especially, those relative to multidatabase views and/or data model translations at internal levels, when databases are heterogeneous with respect to data models. A view, may in particular present some databases as one database. Like through a distributed DBS, one may then manipulate the corresponding data

using (mono)database queries only. One calls such views superviews /MOT81/.

The main functionalities of an MDBS have to be supported by many secondary ones. One has to deal thus with aspects relative to privacy, concurrency, reliability, performance optimization, etc.

Note that it is not mandatory for an MDBS to deal with heterogeneous data models, as some people could think from literature. In contrast, the functionality (b) is a **fundamental**. It means in particular that multidatabase queries should be at **user** disposal. It also means that a system for the work only through a global schema **is not** an MDBS. It is only a logically and, may be, also physically, distributed DBS.

In this classification, the system, like for instance, INFOBASE /LYN83/, is an MDBS for semantic data model databases. MESSIDOR system /MOU81/ is a multidatabase system for heterogeneous bibliographic databases. The system MULTIBASE /CHA83/, /LAN82/ is an MDBS iff convenient multidatabase extensions to Daplex database manipulation language are available to users (papers we could see do not speak about such extensions, but they may exist). The SDD-1 system /BER80/ or DELTA system /LIT82/ are (distributed) DBS. Same thing for R* /WIL82/, althogh this system is closer to the ideas in MDBS through the some aspects of the concept of "site autonomy" /LIN81/ and the possibility of formulation of multisite qeries. However, R* site is not a database, at least in the sense discussed here. Might be, only not yet, since the meaning of the term "site" in R* seems to evolve towards more logical sense.

Indeed, from the classical (physical) meaning of network node in /LIN81/, this concept evolved into a more logical interpretation as "database instance" /SEL84/. Nevertheless, even this meaning does not correspond (yet ?) to a database in the sense discussed here. It designates rather a participating DBS and/or a relational data store in the sense of MRDS /MUL82/. Especially, since notions like permanent identification of data through their "birth" site name or like transparency of data migrations, etc., may concern **implementation** levels only. If data migrated at the **conceptual** level, for instance from their "birth" **database** RESTAURANTS, into another **database**, let it be MY-FAV-REST, then: (i) such migration **would not** be transparent by its meaning itself ; (ii) logical and physical schemas of MY-FAV-REST **would not** contain "birth" database name RESTAURANTS. The whole notion of "birth" database would even seem debatable, since imagine only that a database whose data could migrate anywhere should disappear or should be

renamed for some reasons...

2.4.3 Multidatabase system types.

One may foresee several types of MDBS, depending on involved databases and/or systems. With respect to system aspects, an MDBS may in particular be:

- monosystem. This means that all databases are managed by the same DBS. This is for instance presently the case of MRDSM.

- system homogeneous. Databases are then managed by different copies of the same DBS. Typically, at different sites.

- system heterogeneous. Databases are managed by systems that differ to some extent. The differences may concern data definition and/or manipulation languages within the same data model. Or, - the data models themselves and/or secondary functionalities. This is the environment assumed for MULTIBASE and ADDS systems /BRE84/, as well as in /POP84/.

With respect to data model aspects, it seems that MDBSs for relational databases will be particularly important for obvious reasons. We called them relational multidatabase systems. A relational MDBS may in particular manage databases that are internally (locally) managed according to another model.

2.4.4 Multidatabase manipulation language.

2.4.4.1 The concept.

The data manipulation language of an MDBS was called multidatabase manipulation language (MML). While a data(base) manipulation language (DML) may be navigational, an MML **must be** assertional. The reason is that as multidatabase manipulations involve, by definition, several databases, they must involve, at least in *intension*, more than one records. Therefore, in particular, the multidabase approach has **no sense** for navigational language based data models.

2.4.4.2 Database naming.

An MML should on the one hand provide a database manipulation facilities. This simply means that any MML should contain a DML. On the other hand, it should at least provide a way to identify databases to be manipulated together. This means in particular that, in contrast to any DML, an MML should allow to refer to **database names.**

2.4.4.3 Types of manipulations.

An MML should obviously allow at least all types of manipulations analogous to those already characteristic of a DML. It thus, should allow multidatabase retrievals, updates, insertions and deletions. These operations will be discussed more in details later on.

2.4.4.4 Types of information flow.

In the multidatabase environment, one has to consider two basic types of information flow /COD71/:

(a) - between databases and workspaces (outside world). This is the only type of information flow considered by the database approach.

(b) - between databases.

Note that (b) type flow renders the traditional clear-cut distinction between the concepts of retrieval and the one of modification less sharp. The reason is that retrievals involve modifications of some (target) databases. This type of multidatabase manipulations was called interdatabase manipulations.

2.4.4.5 Inheritence/conversion.

The emergence of interdatabase manipulations restricts the universality of the inheritance rule /COD71/. We recall that through this rule one assumes in any modern DML that a workspace or, more generally, the result of a query, **inherits** the retrieved data (sub)schemas. This assumption was possible because workspaces are, by definition, created by the queries. Database schema may in contrast preexist the incoming data and one may wish it to remain **the same** afterwards. One may therefore need to move data between **different** schemas. Inheritance rule does not apply thus to some cases of (b) type flows, as data schemas need to be **converted** to the target schema.

2.4.4.6 Simplicity.

It should be **simple** for users to formulate their queries. One interpretation is that an informal query (user wish) should then lead to shortest understandable to the user (formal) query. In particular, a multidatabase wish should lead most frequently to only **one** multidatabase query.

This interpretation of simplicity seems useful and general enough. First, it is already implicitly one of the goals of the relational data model /COD82/. Next,

relational queries are typically simpler than navigational ones, according to both: our sense and general perception. Furthermore, the universal relation interface queries may be even simpler, again by both criteria. Note that "simple" in our sense does not mean "natural language expression". Note also, that shorter expressions are simpler only if they are **understandable**. A good language should thus allow longer expressions for novices and shorter ones for experienced users.

2.4.4.7 Typical environment.

Database languages were designed for data assumed **defined by an administrator**. One goal assigned to this **human** intervention was to smooth differences and redundancies between data structures and/or value types of different users (the static integration work). A multidatabase has basically no administrator. A universe may then be described from database to database through different naming conventions and/or value types and/or data structures etc. This, even if all databases are data model and DBS homogeneous.

A good MML should therefore provide simplicity **also** in this environment. Since such environment was not foreseen for a DML, **general** enhancements to expressive power of present manipulation languages are needed. These enhancements should particularly concern the power of **dynamic integration**. For relational data, it is easy to see that a good MML should be more than complete.

3. MRDSM SYSTEM.

3.1 The goal.

MRDSM system is a prototype relational multidatabase system. It provides at INRIA site the multidatabase management of databases of the well known Multics Relational Data Store (MRDS) database system /MUL82/. The goal of the prototype is to experiment functionalities resulting from the multidatabase approach. In particular, to test the implementation techniques. Finally, to show that to extend a relational database system into a monosystem, MDBS may be a rather easy task. One wishes to convince in this manner that, on the one hand, it should be **worthy** to modify in this sense existing systems. On the other hand, that, **new** relational systems should be systematically **multidatabase** ones, as the cost of basic multidatabase features should be only a small fraction of the overall ones.

3.2 Overall architecture.

MRDSM is an extension to MRDS done without changes to this system. For MRDS,

MRDSM is a user among others. For MRDSM, MRDS is the exclusive server of databases and of relational operations. The way in which multidatabase operations are implemented will be described later on.

The system consists of various components providing functionalities discussed below. The corresponding system architecture and mutual interactions are described in details in /WON84/.

3.3 Main functionalities.

3.3.1 Data definition.

3.3.1.1 Multidatabases.

All MRDSM databases constitute implicitly a multidatabase named, by default, INRIA. However, the elements of this multidatabase are not directly databases, but project multidatabases, constituted each of all databases within one Multics project (basic user community). Then, each project multidatabase contains all multidatabases of project members (Multics users). A user multidatabase may be structured further, as it may contain databases only.

The project and user multidatabases bear respectively the corresponding project name and user name. The user multidatabase is also called user universe. Within its universe, a user needs only relative (multi)database names. For other databases, one invokes also the corresponding user names and, if needed, the project names. Relative names are completed to the absolute names by the system.

This way of structuring and naming multidatabases was chosen mainly because of naming principles of MRDS and of Multics in general. Examples of multidatabase definition are in /WON84/ and /WON84a/.

3.3.1.2 Databases.

Databases are defined using the standard MRDS facilities. This includes in particular the access rights.

3.3.1.3 Multidatabase dependencies.

Presently, one may declare two types of such dependencies :

- manipulation dependencies that interrelate manipulations of different databases,

- privacy dependencies that are constraints on matching of data from different databases.

Description of these dependencies is postponed to Sections 3.3.3 and 3.3.4, since it requires concepts of Section 3.3.2.

3.3.2 Data manipulation.

3.3.2.1 MDSL language.

The multidatabase manipulation language of MRDSM was called MDSL (the one of MRDS is called DSL). MDSL is SQL-like type language. It includes DSL and offers various multidatabase manipulation functionalities. Full description of these functionalities may be found in /LIT84b/. Some of them, although developed for multidatabase usage, turn out to be interesting for a DML as well.

MDSL queries (we recall that in our terminology a query may be any manipulation and so in particular an update) may be classified according to various criteria. These criteria are not always disjoint. Basically one may use the following types of queries :

- all DSL queries,
- multidatabase retrievals, modifications, insertions and deletions,
- interdatabase queries,
- elementary multidatabase queries,
- multiple multidatabase queries,
- incomplete queries,
- queries with dynamic attributes,
- queries with new standard (library) functions.

MDSL provides also some auxiliary commands. For instance for query preprocessing, for multidatabase schema displaying etc., /WON84a/. Finally, one may call any Multics command. Thus one may call text editors, execute or compile programs etc.

Detailed description of DSL possibilities may be found in /MUL82/. Below, we will only overview multidatabase possibilities of MDSL.

3.3.2.2 Multidatabase retrieval, modification, insertion and deletion.

The corresponding queries are formulated using MDSL (and DSL) commands : **retrieve, modify, insert** and **delete**. The difference with respect to the corresponding

DSL queries is that the clauses -where and/or -select refer to relations and attributes in different databases. If two relations share then a name, they may be prefixed first by the corresponding database names. Next, if necessary, by the multidatabase names, until the designations are unique.

3.3.2.3 Interdatabase queries.

In such queries both the source and the target are databases. The command type may be retrieve, insert or delete. The effects are as follows :

- **retrieve** command replaces the existing target relation(s) with the selected tuples and the corresponding schema(s). It thus applies the inheritance rule.

- **insert** command adds the selected tuples to the existing target relation(s). If needed, the corresponding attribute order and/or types are **converted**. The source database(s) remain(s) unchanged.

- **delete** command acts like insert, except that the source is changed as typically for a delete.

The multidatabase delete corresponds to the needs of users who, although wishing to delete data from a database, still wish to archive these data elsewhere.

3.3.2.4 Elementary multidatabase queries

An elementary multidatabase query produces a relation in a workspace. Basically, it is a usual relational DSL query except that :

-select and or -where clauses refer to more than one database.

- some interrelational joins may not be specified.

For instance, assume that *R (name, street, tel,...)* is the relation about restaurants in RESTAURANTS database and *C (name, street, tel,...)* is the relation about cinemas in CINEMAS. Then, the query "retrieve names of restaurants and cinemas at the same street" is an elementary multidatabase query. See /LIT84b/ or /WON84a/ for the corresponding MDSL expression.

In what follows, a usual relational query or an elementary multidatabase query will be jointly called elementary queries.

3.3.2.5 Multiple multidatabase queries.

3.3.2.5.1 The concept.

A multiple query is a query to be repeated for some databases or for some sets of

databases. The set of all concerned databases is called the scope of the query. Each step (one repetition) gives rise to one elementary subquery. Any subquery is evaluated with respect to the corresponding database(s). Eventualy, its predicate is adapted. In particular when it refers to attributes that **are not** in the database. A subquery that may finally extract some content, is qualified as pertinent. The result of a multiple query is typically the one of all pertinent queries. Multiple queries create or modify thus basically **sets of relations**.

For example, the query asking for a modification of some *tel* in both *R* and *M* is a multiple query. This query replaces two relational queries. In general, one multiple query may avoid even very many elementary queries. Multiple queries are thus **simpler** for multidatabase users, in the sense of the Section 2.4.4.7. From general point of view, this concept is one of the enhancements to the power of dynamic integration, we spoke about in that section.

In fact, it seems that multiple queries will be among facilities **most desired** by multidatabase users. This functionality characterizes presently only MRDSM and INFOBASE systems. One also calls such queries diffusion queries or broadcast queries.

In MDSL, multiple queries are formulated through the application of several new concepts. In particular, one may apply the following ones, described more in /LIT84b/:

- multiple identifiers,
- semantic variables with explicit or implicit domains,
- options on the target list.

3.3.2.5.2 Multiple identifiers.

We called designator a name that one uses in a query in order to designate some data. In known DMLs, designators are unique identifiers that means that they univocally identify an object (an attribute, a relation, an entity or record types, etc.). In MDSL, this is also a **necessary** property of any elementary query (and **sufficient** when all joins are specified). However, one may also use multiple identifiers. Such identifiers designate some objects sharing the same name in the query scope. For example, the designator *tel* would be the multiple identifier of both *R.tel* and of *C.tel* objects, if the scope involves both databases.

A query may invoke several multiple identifiers. Any multiple identifier means then

that the query concerns all objects it designates. The corresponding multiple query is equivalent to all pertinent subqueries resulting from all possible choices of the corresponding unique identifiers. For instance, a query "**modify** ... *tel*..." would be equivalent to two relational queries : "**modify** ... *R.tel*..." and "**modify** ... *C.tel*...". A multiple query with multiple identifiers **simplifies** thus the formulation of multidatabase manipulations of objects bearing the same name in different databases.

Note that this notion reveals useful also for the database approach. As long as the same attributes bear the same names in different relations, usage of multiple identifiers may provide the **referential integrity**.

3.3.2.5.3 Semantic variables.

A semantic variable is a designator whose values are object names called semantic variable domain. The domain may be :

- explicit which means that it is explicitly enumerated in the query,

- implicit which means that it is deductible from the semantic variable name.

A query may invoke several semantic variables, together with multiple identifiers. Any semantic variable means that the query concerns all names in its domain. These names may in particular be multiple identifiers. The corresponding multiple query is equivalent to all pertinent subqueries resulting from all possible choices of the corresponding unique identifiers.

Semantic variables allow basically to replace with one query several similar elementary queries concerning objects named differently. In particular, they may express dynamically implicit or explicit **abstractions or generalizations**. The exact manner in which semantic variables are defined in MDSL is described in /LIT84b/. As a simple example, assume that one wishes to retrieve all cinemas and restaurants in discussed databases. One may say in MDSL "retrieve *X*", with *X* being a semantic variable ranging over values *R* and *C*. *X* stays here for an implicit generalization, let us call it *Interesting-Place*.

3.3.2.5.4 Options on the target list.

The target list is the list in the -select clause. In a relational DML, it means implicitly that all the corresponding objects are wanted. This assumption appears not always justified in the multidatabase case. One reason is that the user will sometimes not know in detail all the corresponding schemas. Options on the target

list allow then to precise :

- whether a given object is mandatory or optional. For instance, a multiple query may precise that one is interested by restaurants from a database only if they are characterized by *tel* attribute.

- attribute priorities. For instance, one may say to select *street* for restaurants in a database only if there is no attribute *tel*.

- etc. /LIT84b/.

Options may be used jointly with multiple identifiers and semantic variables.

3.3.2.6 Incompletely specified queries.

An MDSL query is incompletely specified or, in short, is incomplete, when some interrelational equijoins are not defined in the -where clause. Basically, one may avoid equijoins on relation keys. Undefined joins are called implicit joins /LIT84/. They are deduced by the system from the database schema. This deduction is called completion process. The result is called a complete query. A complete query may be an elementary query or a subquery of a multiple query.

This facility further simplifies MDSL query formulation in two ways :

- query formulation may be shorter,

- some multiple queries must be incomplete.

The latter property results from the fact that one may express a manipulation in a form more independent of the manner in which some databases are structured into relations. The completion process provides then the corresponding details for each database. If one had to formulate only complete queries, then it could be no way to express all different details in the same query.

The goal of the concept of incomplete query is clearly to some extent similar to the one of the concept of **universal relation** /KEN83/, /MAI83/, /ULL83a/. However, MDSL does not assume that the schema consists of **only one** relation. Corresponding consequences are described in /LIT84/. One major consequence is that one may perform not only retrievals, but also **modifications**.

3.3.2.7 Dynamic attributes.

Dynamic attributes are transforms of actual attributes that are defined dynamically for the purpose of a query. A transform may consist of a formula, of a

dynamically defined table, of a call to a dictionary or of a program in any Multics programming language. Once defined, a dynamic attribute is manipulated as any actual attribute. Detailed description of this functionality may be found in /VIG84/.

Dynamic attributes are another functionnality enhancing the power of dynamic integration (section 2.4.4.7). They respond mainly to user needs that cannot be determined in advance and/or may change from query to query. In such cases, view concept appears **inadequate**, because of its **static** nature. In particular, dynamic attributes respond to the following needs :

- instant transformation of attribute value type according to user needs. For example, an actual currency may be transformed into an arbitrary one. The exchange rate may furthermore be totally subjective and different from query to query. For instance, it may first reflect user opinion about $ value in three months, then in a year. Note the fundamental inadequacy of view concept to the whole currency exchange problem (daily variation of rates, number of possible conversions,...)

- dynamic transformation of some attributes into one attribute. For instance, if needed, one may instantly define the attribute month-salary from actual attributes : day-salary and number-of-working-days-per-month.

- instant and subjective homogenizations of different attribute types. In particular, of those with fuzzy relationship. For instance, a user of databases, let them be MICHELIN and GAULT-M, representing the famous restaurant guides Michelin and Gault-Millau, may ask for restaurants that are "good" in **his** opinion. For that, he may say through the appropriate dynamic attribute, that "good" means, **in this query**, "***" in the restaurant quality rating of MICHELIN and at least 16/20 in the GAULT-M scale.

Dynamic attributes may be defined for retrievals and/or for updates. Presently, when a dynamic attribute is used for both purposes, the user has to basically supply both transforms. In the future, update transforms should be deduced typically automatically. This, through usage of an expert system.

3.3.2.8 New standard functions.

Case study show that, in the multidatabase environment, a number of new standard (library) functions are useful. The main reason are the basic properties of multidatabases, discussed in the section 2.4.4.7. Currently MDSL provides the

following functions :

- NAME function. It provides either the name of a given object, or of its container (ex. relation for an attribute) and so on. In particular, it allows to respond to queries for object **names**, like for instance, "Guides recommending restaurant Maxim's". If the restaurant is recommended by MICHELIN and GAULT-M, the response will be the **names** of these databases (note that actually Maxim's is not recommended by Michelin).

- EXTEND function. This function dynamically extends a relation with an arbitrary value as new attribute. In conjunction with NAME, it allows in particular to convert an object name into its attribute. For instance CINEMA may become an attribute of C, indicating then **explicitly** that any tuple of C models a cinema.

- NORM functions. These functions merge into one tuple all tuples retrieved by a multiple query and corresponding to the same real object. For instance, all tuples corresponding to a restaurant recommended by more than one guide (note that a restaurant is not always recommended by all guides and that the recommendations may differ, so it is not a simple redundancy). The recognition of correspondence to the same real object results from the equality of values of some attributes called normalization key. Note that to find such keys may be a non-trivial problem.

- UPTO function. This function limits multiple information relative to the same object. For instance, one may ask for at most two recommendations of a restaurant, or for at most one indication of its telephone number etc. In particular, one may express preference to MICHELIN and GAULT-M recommendations, if any.

More about MDSL standard functions may be found in /LIT84b/ and /ABD84/.

3.3.3 Manipulation dependencies.

Two manipulations are dependent if one triggers another. In MRDSM, one speaks then respectively about source and complement. This dependency may be multidatabase, as a query to a database may trigger a query to other databases. Also, the source and/or the complement may be multidatabase queries.

The execution of a complement may precede or may follow the end of the source manipulation. In particular, it may be deferred. One source may have several complements. A complement may be a source for other complements etc. For practical reasons and in order to avoid (infinite) cycles, one should nevertheless limit the number of complements that a source may in definitive trigger. Current

limit is about 200 complements /REG83/.

3.3.4 Multidatabase privacy.

Case studies show that multidatabase privacy constraints should mainly prohibit to collect too much data about the same real object from various databases. For instance, the French law would typically prohibit queries to too many databases if one looked for data of a citizen. In general, the quantity of information permitted depends on user name and/or data concerned. Typical constraints may require /ACH84/:

- that the scope of a multiple query does not exceed a given size,
- that the scope does not include given databases,
- that some values do not appear in the result,
- etc.

MRDSM allows to define arbitrary privacy procedures. The procedures should be written in one of Multics programming languages (i.e there is no MRDSM language dedicated for this purpose). Each procedure is declared for a given user and/or command type and/or data. It is then called when a query corresponds to declared arguments. Same arguments may trigger several procedures.

Presently MRDSM does not keep history of queries. A procedure that should prohibit users from trying to accumulate data through several small scope queries, has thus to constitute the corresponding history data by itself.

4. IMPLEMENTATION ASPECTS.

Details about implementation of discussed functionalities are in references. Below, we describe only a few guidelines to query processing.

4.1 Elementary complete queries.

The main assumption for the system design was that MRDS is the exclusive server of relational store. The objective was of course to avoid the cost of reprogramming relational operations. However, MRDS is able to execute only monodatabase queries. One way to use then MRDS for multidatabase operations with multidatabase joins is to **dynamically** render the corresponding data a **database**. The following solution applying this principle was therefore developed for elementary complete queries :

- MRDSM looks whether the query is a multidatabase query. If not and it is a monodatabase query, then it calls MRDS. If it is an auxiliary or Multics command, then control is passed to appropriate subsystems.

- Else, MRDSM decomposes the multidatabase query into a set of MRDS queries. Each of these queries selects data designed in the original query that correspond all to the same database. The selection expressions are optimized /COR84/. The optimization consists mainly from performing restrictions first. In the case of updates, the final selection expressions involve key attributes, even if the the original -select clause did not mention them.

- The selected data end up in a dynamically created database, called working database /GUE81/. The schema of this database involves attributes designated in the query and, eventually, key attributes.

- MRDSM produces a query to the working database. This query involves operations that were not performed through previous selections and restrictions. In particular, the multidatabase joint expressions within the original -where clause.

- For retrievals, this query produces the final result. For updates, update queries to source databases are furthermore produced. The corresponding tuples are identified through matching of working database keys values onto the ones in source databases.

- The working database is destroyed, unless MRDSM needs it for further purpose.

4.2 Incomplete queries.

The completion process, i. e. the research for implicit joins, consists in an examination of connections through domains /LIT84/. For this purpose, both the query and schema are considered as defining a graph which nodes are relations. Joins that should complete the query correspond then to minimal trees over given sets of nodes. Basically, these nodes are names of relations designated in the query.. Each tree leads then to one subquery. The resulting complete query is the union of subqueries that are always union compatible. Or, it is a multiple query that is a set of such complete queries.

4.3 Normalization functions.

NORM functions are processed through extensive use of various types external joins. Since, external joins are unknown to MRDS, algorithms for efficient

processing of such operations have been investigated /ABD84/.

5 CONCLUSION.

We have presented an informal overview of the prototype relational multidatabase system MRDSM. We focused on the general philosophy of the system and on its main functionalities. Most of these functionalities are yet unknown to other systems. Nevertheless, databases becoming widely spreaded, such functionalities should be soon required by many users.

Most of these functionalities constitute new possibilities at the level of data manipulation language. They enhance manipulation simplicity and possibilities of dynamic integration. The corresponding needs result from growing availability of many independently administrated databases. In particular, the public videotex systems, like Prestel, Telelel or Telidon etc. will soon provide hundreds of databases at virtually everybody disposal. This situation is new for the database approach that guided the features of the current languages. The proposed concepts are intended to be a response to main user needs in this new environment. Various kinds of multiple queries, dynamic attributes, and interdatabase queries should reveal particularly needed.

Future work will concern basically the same directions. More effort will nevertheless be devoted to more ponctual consequences of the already identified general notions and to the corresponding implementation techniques. In particular, more attention will progressively be devoted to knowledge processing techniques, since they appear best tool for many envisaged developments. For example, an expert system should be able to avoid to some extent the need for both retrieval and update declarations of dynamic attributes. It should indeed be frequently able to deduce one of the declarations from the other one.

REFERENCES.

/ACH84/ Achour, M. Contrôle de confidentialité dans un système de gestion multibases (MRDSM). Th., Univ. of Tunis. INRIA, (June 1984), 216.

/ANS75/ ANSI-SPARC interim report on data base management system. Doc. 7514TS01, Washington DC, (Feb. 1975).

/ABD83/ Abdellatif, A. Traitement de requêtes multiples par un système de gestion multibase (MRDSM). Th., Univ. of Tunis. INRIA, (June 83), 80.

/ABD84/ Abdellatif, A. Jointures externes et fonctions standards dans le système multibases MRDSM. Rapp. DEA, Univ. Paris 6, (June 1984), 67.

/BER80/ Berstein, P., A. et al. Query processing in a System for Distributed Databases (SDD-1). ACM-TODS, 6, 4, (June 1980).

/BRE84/ Breitbart, M. ADDS : another multidatabase management system. 3-rd International Seminar on DDSS, Parme, (March 1984), North Holland, 7-24.

/CHAN83/ Chan, A. On the development of distributed database management technology. EUTECO, North-Holland, 641-656.

/COD71/ Codd, E., F. A database sublanguage founded on the relational calculus. ACM SIGFIDET, (Nov. 1971), 35-68.

/COD79/ Codd, E., F. Extending the database relational model to capture more meaning. TODS, 4, 4, (June 1979).

/COD82/ Codd, E., F. Relational Database: A Practical Foundation for Productivity. CACM, 25, 2 (Feb 1982).

/COR84/ Cortes, R. MRDSM: optimisation of processing of multidatabase queries. Position papers of 3-rd International Seminar on DDSS, Parme, (March 1984), University of Parma, 40-42.

/DAN82/ Daniels, D. et al. An introduction to Distributed Query Compilation in R*. DISTRIBUTED DATA BASES, North-Holland, 1982, 291-309.

/DAY83/ Dayal, U. Processing Queries over Generalization Hierarchies in a Multidatabase System. VLDB 83, Florence, (Oct. 1983), 342-353.

/DAY84/ Dayal,U., Gouda, M., G. Using Semiouternjoins to Process Queries in Multidatabase Systems. ACM-PODS, Waterloo (Canada), (Apr. 1984), 153-162.

/GUE81/ Guemara, S. MRDSM : un systeme de gestion de multibases relationnelles. Res. rep. INRIA-SIRIUS MOD-I-046, 69.

/HAM79/ Hammer, M., McLeod, D. On database management system architecture. Lab. for Comp. Sc; MIT, MIT/LCS/TM-141, (Oct. 1980), 75-85.

/KAB83/ Kabbaj, K. Conception d'un système de gestion multibases. Th. 3-rd Cycle, Paris VI, (Mai 1983), 258.

/KAL84/ Kalinichenko, L., A. Methods and tools for integration of heterogeneous databases (in russian). Nauka, 1983, 424.

/KEN83/ Kent, W. The Universal Relation Revisited. TODS 8, 4, (Dec. 83), 644-648.

/KOR84/ Korth, H., F., Kuper, J., M., Feingenbaum, J., Gelder, van A., Ullman, J., D. System/U: A Database System Based on the Universal Relation Assumption. ACM-TODS, 9, 4, (Sep. 1984).

/LAN82/ Landers, T., Rosenberg, R. L. An overview of MULTIBASE. 2-nd Symp. on Distributed Data bases. Berlin, (Sep. 1982). Proc. North-Holland, 1982, 153-184.

/LEF84/ Lefons, E. Silvestri, A. Multidatabase systems. 3-rd International Seminar on DDSS, Parme, (March 1984), North Holland, 25-42.

/LIN81/ Lindsay, B., G. Object Naming and Catalog Management for a Distributed Database Manager. 2nd Int. Conf. on Distr. Comp. Syst., (Apr. 1981), 31-40.

/LIT79/ Litwin, W. Distributed data bases : a way of thinking about. 1-st International Seminar on Distributed Data Sharing Systems. Aix-en-Provence (May 1979), 451-470.

/LIT80/ Litwin, W. A model for a distributed data base. 2-nd ACM Comp. Exp. University of Laffayette. Luisiane, (Feb. 1980), 54-71.

/LIT81/ Litwin, W. Logical model of a distributed data base. 2-nd International Sem. on Distributed Data Sharing Systems. Amsterdam (June 1981), North-Holland, 173, 207.

/LIT82/ Litwin W. et al. SIRIUS Systems for Distributed Data Management. DISTRIBUTED DATA BASES. North-Holland, 1982, 311-366.

/LIT83/ Litwin, W., Kabbaj, K. Multidatabase Management Systems. ICS 83, Nurnberg, (March 83). B. G. Teubner, 482-505.

/LIT83a/ Litwin, W. MALPHA : a multidatabase manipulation language. EUTECO, North-Holland, 428-463.

/LIT83b/ Litwin, W. The future of distributed databases (position paper). EUTECO, North-Holland, 464-469.

/LIT83c/ Litwin, W., Kabbaj, K. Les systemes multibases. Réseaux, No 6, 83, 4-8.

/LIT84/ Litwin, W. Jointures implicites dans le système MRDSM. Rapp. de rech. SESAME. INRIA, (March 1983), 12.

/LIT84a/ Litwin, W. MALPHA : a relational multidatabase manipulation language. IEEE-COMPDEC, Los Angeles, (Mai 1984), 234-242.

/LIT84b/ Litwin, W. Concepts for multidatabase manipulation languages. JCIT-4, Jerusalem, (June 1984), 433,442.

/LYN83/ Lyngbaek, P., McLeod, D. An Approach to Object Sharing in Distributed Database Systems. VLDB 83, Florence, (Oct. 1983), 364-376. Extended version to app. in ACM-TOIS.

/MAI83/ Maier, D., Ullman, J., D. Maximal Objects and the Semantics of Universal Relation Databases. TODS, 8, 1, (March 1983), 1-15.

/MOT81/ Motro, A., Buneman, P. Constructing Superviews. ACM-SIGMOD 1981, 56-64.

/MOU81/ Moulinoux, C., Faure, J. C., Litwin, W. MESSIDOR system. ACM-SIGSMALL Symposium on small systems. Florida. (Oct 1981) 130-135.

/MUL82/ Multics Relational Data Store (MRDS) Reference Manual. CII-Honeywell Bull. Ref. 68 A2 AW53 REV4, (Jan. 1982).

/POP84/ Popescu, R. Gligor, V., D. Transaction management issues in integrated heterogeneous database management systems. 3-rd International Seminar on DDSS, Parme, (March 1984), North Holland, 43-56.

/REG83/ Regaieg, N. Traitement de compléments de manipulation dans un système multibase (MRDSM). Th., Univ. of Tunis. INRIA, (June 83). 67.

/ROT78/ Rothnie, J. B. Distributed database management. VLDB-78, Berlin, (Sep. 1978).

/SEL84/ Selinger, P. et al. The impact of site autonomy on R*. Databases-Role and Structure. Cambridge Univ. Press, 1984, 151-176.

/SHI84/ Shili, B. Dépendances interbases et jointures implicites dans un système de gestion multibases (MRDSM). Th. Univ. of Tunis. INRIA, (June 1984), 216.

/STA84/ Staniszkis, W., Turco, G. Network database management system architecture. 3-rd International Seminar on DDSS, Parme, (March 1984), North Holland, 57-76.

/ULL83/ Ullman, J., D. Principles of Database Systems. 2-nd ed. Computer Science Press.

/ULL83a/ Ullman, J., D. On Kent's "Consequences of Assuming a Universal Relation". TODS 8, 4, (Dec, 83), 638-643.

/VIG84/ Vigier, Ph. Traitement des attributs dynamiques par un système de gestion multibase (MRDSM). Rapp DEA, Paris 6, (June 1984), 62.

/WIE83/ Wiederhold, G. Database design. McGraw-hill Book Company, 1983.

/WIL82/ Williams, R. et al. R* : An Overview of the Architecture. 2-nd International Conference on Databases. Academic Press, 1982.

/WON84/ Wong, K. Conception et Réalisation d'un Système de Gestion Multibases Relationnel MRDSM. Th. 3-rd Cycle. UPS, Toulouse (June 1984), 229.

/WON84a/ Wong, K. Bazex, P. MRDSM : a relational multidatabase management system. 3-rd International Seminar on DDSS, Parme, (March 1984), North Holland, 77-87.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique