

IST. EL. INF.  
BIBLIOTECA  
Posiz. Archivio

||  
*Consiglio Nazionale delle Ricerche*

||  
**ISTITUTO DI ELABORAZIONE  
DELLA INFORMAZIONE**

**PISA**  
||

GLOBAL REQUESTS IN A DISTRIBUTED DATA BASE  
SYSTEM FOR OFFICE AUTOMATION PURPOSE

F. Rabitti

Nota Interna B80-17

Luglio 1980

"Global Request in a Distributed Data Base System for Office  
Automation Purpose"

F. Rabitti\*

Istituto Elaborazione della Informazione - C.N.R. - Pisa

\* Il Dottor Fausto Rabitti ha svolto la sua attività di borsista C.N.R. presso l'Istituto di Elaborazione della Informazione negli anni 1978-79, attività che ha proseguito nel 1980 presso il Computer Systems Research Group dell'Università di Toronto, grazie ad una borsa C.N.R. per l'estero.

## **1 Discussion of Global Requests problem in OFS.**

- Introduction and discussion of the Global Requests problem in the OFS Distributed Data Base System.

### **1.1 Global Query requirements in the general case.**

The Global Query (GQ) problem in a Distributed Data Base (DDB) consists in allowing an end-user (either an on-line user, on a high level conversational interface, or an application program) to perform a query on any relation of the DDB, involving any possible file storing such relations, without being aware about the locations of these files in the Distributed System.

A further requirement about the GQ is that the answer must be consistent with the global data model of the DDB (that is, each data element, logically involved in the GQ, must be correctly considered, respecting the logical relations of the data model) and with the actual situation of the DDB (that is, the answer must refer to the most up-to-date situation of the DDB).

It is impossible, in principle, to obtain an answer to a GQ which refers to the situation of the DDB at the moment of the reception of the answer, because of the concurrent activities, such update operations, in progression on the Distributed System (DS), unless completely blocking all the other activities in the DS excepting those involved in the GQ.

It might be reasonably required that the answer to a GQ refers to the situation of the DDB at an intermediate moment between the occurrence of the request and of the answer. Of course, shorter this interval is, more up-to-date the answer to the GQ can be assumed.

In the general case there can be multiple copies of the relations, or of sets of tuples constituting each relation, and they can be located in several nodes of the DS. In this case, the most convenient copy (the least expensive one, in terms of network load), among the ones which are currently up-to-date, ought to be accessed; this decision, in a distributed environment, is not an easy matter.

The problem of allowing general GQs in a DDB is very difficult to be solved in an effective way, that is, without overloading the DS and the

Communication Subsystem, in detriment of performance, or without blocking the other concurrent activities in the DDB.

Obviously, the property of this consideration depends from the amount of processing power that is available, in the DS, for the execution of general GQs. As it has been shown by actual implementations, like the Distributed Ingres (Ref.1) and the SDD-1 (Ref.2), at the state of art of today the processing of general GQs can be considered acceptable, even if it is not good yet, in a DS of considerable processing power; but it cannot be considered acceptable in a DS of limited processing power, based for instance on cheap Micro and Mini systems as nodes. In the latter case, there is no choice between overloading the DS with an unbearable amount of work or blocking the other concurrent activities allowed in the DDB.

In the general case, performing a join between two relations (see the following Example-1) implies that, if a copy of both relations is not completely kept in one location of the DS, a heavy communication activity (either in size or in number of messages) is required across the network.

The situation becomes far more complex when a relation can be spread among several locations (nodes) of the DS; that means the tuples composing the relation can be located in several nodes, and the same tuple occurrence can be stored in more than one node, if duplication is allowed.

In the second case, a larger amount of communications activities can result necessary for a GQ: they must occur among a group of locations which altogether keep at least one occurrence of each tuple, for each relation involved.

In this latter situation, also a GQ logically involving only a relation (see the following Example-2) may require, for example in case of a nested query, that more than one node in the DS (again, a group of locations which altogether keep at least a copy of each tuple of the relation involved), must cooperate, in a likely complex way, in order to build up the answer.

It must be remarked that the policy of keeping a complete copy of a relation in many nodes of the DS, which could be considered as the easiest solution to the previous GQ problems, besides the aspect of storage space wasting, presents a severe trade-off in securing multiple copies consistency, in consequence of concurrent update operations.

### *Examples*

Two examples are shown here, to explain the previous two kinds of problems.

They make use of the SLQ (Sequel-Like-Query) language of MRS (Ref.3), a relational data base system.

#### *Example-1*

Given: Rel' (Attr.1',Attr.2', ... ,Attr.N')  
Rel'' (Attr.1'',Attr.2'', ... ,Attr.N'')

Join query:

*select* Attr.k',Attr.l'' *from* Rel',Rel''  
*where* Attr.i' =Attr.j'' *and* clause{Attr'} *and* clause{Attr''}

NOTE: clause{Attr'} shows whichever compound condition, as allowed in MRS, on a subset of attributes of Rel'.

#### *Example-2*

Given: Rel (Attr.1,Attr.2, ... ,Attr.N)

Nested query:

*select* Attr.i *from* Rel *where* Attr.k  
*in\_**select* Attr.k *from* Rel *where* clause{Attr}

## **1.2 Office Form System environment.**

The Global Query problem is to be studied, here, in the Distributed Data Base environment on which the Office Form System (Ref.4) is based.

The Office Form System (OFS) is a distributed system for office automation purpose, which deals mainly with data items in "form" format; besides the OFS is intended to work in a distributed environment based on processing units of limited capability, like Micro and Mini systems.

The representation and handling of these forms, as well as the maintenance and use of internal control structures, are strongly based on an underlying Relational Data Base, MRS (Ref.3) compatible (that is, this Relational DB can be accessed through the MRS, too).

In OFS environment there is a one-to-one correspondence between form type and relation, and between form occurrence and tuple. Fields of a form are merely attributes of the corresponding relation.

Nevertheless this Relational DB, distributed over multiple OFS stations (which constitute a Distributed System) is not really an integrated Distributed Data Base System.

In fact, each OFS station owns its Relational DB (or better, its set of Relational DBs), all Relational DBs in such total DS are compatible (they have the same relation definitions), but each OFS station can access only the DB (or DBs) located inside itself. Moreover, the interactions between the DBs in different stations are mainly for specific OFS purposes, as explicitly sending and receiving forms among the stations, and not for more general purposes characterizing a Distributed Data Base System, as Global Query and Global Update operations on relations spread over the DS.

This previous situation corresponds to the original aims of OFS project. But the OFS has been further studied and adapted for the development of more advanced office automation procedures in a distributed environment, above all in the direction of the automation of several procedural activities, as in the case of "sketch" mechanisms (Ref.5), and their progressive integration in a functional Office Information System (Ref.6).

Developping this kind of researches, it has become evident the usefulness of providing the single OFS station (whether under control of an end-user, with possible help of several automatic procedures, or completely automatized) with Global Data Base services on data contained in forms within the overall Distributed System.

Allowing Global Queries, and eventually Global Updates, in the OFS Distributed System means that it becomes also a real Distributed Data Base System.

### 1.3 DDB special features in OFS environment.

- Special features and restrictions of the Distributed Data Base in OFS environment.

Some special features, which characterize the OFS distributed environment, will characterize also the definition of this particular Distributed Data Base System.

Let us examine the most relevant aspects of OFS, mainly as regards to the definition of the underlying DB environment.

In the OFS environment there is a total identification between the concepts of:

form type	$\Leftrightarrow$	relation
form occurrence of a type	$\Leftrightarrow$	tuple of a relation
field of a form type	$\Leftrightarrow$	attribute of a relation

These two kinds of terminology can be alternatively used according whether the same objects are seen in the view of the Office Form System or in the view of the underlying Relational DB. The only real difference concerns the OFS user interface.

In effect, although the OFS stores and manages forms as tuples in relations, included the uses of indices, the end-user of OFS sees a form like a paper form displayed on his screen. For all the OFS operations at the user interface (filling, finding, modifying, etc.) a form is presented in this way.

That is accomplished using a special file, which contains the specification of the format of the blank form (this file is not a relation file).

Through this special file, and the relation file of the form (which contains the data, namely the fields values, of the form occurrences) any particular form occurrence can be displayed to the user in a paper-form format.

Nevertheless this feature concerns only the OFS user interface, so, for the purpose of accessing data associated to forms (this is in effect the purpose of a Query) it can be completely neglected.

The following point can be remarked.

In the present situation, if a user want to access the data of a form-type (data in corresponding relation), exploiting the full capability of a DBMS Query Language, he must access these data through the Sequel-Like-Query language (SLQ) of MRS; this is a traditional approach to the DB access.

It seems that the OFS, for a more integrated structure, should provide a single form oriented interface to the user. In this environment it would be

attractive a form oriented DB access language where queries are specified by filling forms in a manner similar to Query By Example (Ref.6 and Ref.7).

The previous observation can be extended to the Global Query case (that is, Global Queries can be included in a form oriented DB access language).

Any way this choice of user interface does not affect the DDB System which has to perform these GQs.

The most distinguishing feature of this type of DB is the unique occurrence of tuples in the DS.

There is only a physical copy of a form instance, at any time, in the DS. The particular location of a form occurrence has a semantic meaning for OFS: this form actually belong to a OFS station (or better, to a DB of this station), and so can be involved in the particular activities of this station.

In this way, such DB System deeply differs from other Distributed DB Systems (Ref.1 and Ref.2), which allow duplication of tuples, and in which the choice of location for a tuple copy is determined not only by a semantic factor but also by performance, security and fail recovery considerations.

This unique existence is reached with the introduction of a particular key in each form occurrence. This key is unique, for a certain form type, in the overall system. So a form type and a key uniquely determine a form occurrence in the DDB.

A centralized central mechanism provides that the keys are not replicated.

On the ground of a key, it can be determined if a form occurrence is a form or a copy, and the order ("meter") of the copy.

In this way a "relation", in such a global DB, can be considered partitioned in a number of mutually exclusive subsets, which altogether form the relation. Each subset is in a logical OFS station, and can be called "station relation". Moreover, such a subset of a relation can be partitioned in further subsets, since many DBs may belong to a OFS station. These subsets can be called "local relations".

In this way the DDB, in this OFS environment, can be so defined:

- a *relation* is composed by all tuples (form occurrences) spread on the DS;
- a *station\_relation* is composed by all tuples, of a relation, owned by a station;
- a *local\_relation* is composed by all tuples, of a relation, contained in a single DB, which is located in a station.

So we have that:



- a *Data\_Base (DB)* is a set of all local relations, some of them possibly empty, corresponding to all the relations definitions (form types) in the DDB;
- a station owns several DBs, they can be called the *Station\_Data\_Base (SDB)*;
- the union of all DBs in the DS (in all the stations) constitutes the *Distributed\_Data\_Base (DDB)*; it is to be remembered that the intersection of the various DBs is empty.

Another important observation is that all the relations defined for the D B, at global level, are also explicitly defined for all the DBs, at local level. This homogeneity among the various DBs greatly helps the distributed algorithms for the global operations.

Another feature of OFS is relevant at this point. The main communication activity between OFS stations is the possibility, for a station, of sending a form to another station ("ship" request) and the possibility, for a station, of receiving forms sent to it by other OFS stations ("mail" request).

For this purpose, the concept of "tray" is introduced: a tray is intended to hold the forms sent to a particular station from other stations, until the target station decides to accept these forms.

The control and management of trays is centralized, but conceptually each tray belongs to its destination station.

The node where the centralized activities of this DS reside (storage and management of trays as well as global keys control mechanism, and other functions, like tracing of movements of forms in the DS) is called "host", whereas the other nodes, where the OFS stations reside, are called "satellites".

The choice of centralizing the locations of trays is justified by the requirement of saving forms which are sent to temporarily not connected or not active OFS stations, and of providing overall control for the DS. Nevertheless this choice does not affect the logic of the OFS system or the communication network topology.

Logically, a tray is a single DB, as previously defined (that is, set of all local relations), even if the implementation may be different: for example, in the actual implementation (the non global one) the forms of all trays are contained in a single DB.

We will refer to the logical representation of trays.

The operations OFS stations can do on the trays are limited to sending and receiving forms as mail. Even if a tray is a DB which logically belongs to an OFS station, this station cannot do the same DB operations (real query, update, etc.) it can do on the DBs located in the station (SDB). So

the SDB is not intended to include the DB-tray of the station.

This choice is mainly due to implementation restrictions and to the intention of differentiating the DB-tray from the other station DBs: in fact the OFS station does not have the effective control of forms present in the tray, since forms can be inserted asynchronously by other stations.

It must also be noted that actually an OFS station, even if it owns several DBs, can operate on only one DS at time.

The logical global structure of the DDB of OFS is shown in Fig.1.

#### 1.4 GR special features in OFS environment.

- Particularities and special features of the Global Requests problem in the OFS Distributed Data Base, compared with the general Global Query problem.

As it results in the previous discussion (Par. 1.3), each form is uniquely identified by its key, and if a form is located in a particular OFS station, either in its station relation or in the local relation in its tray, it means that specific semantic operations (checking, updating, processing in general, etc.), related to this station, are expected to be performed on this form.

This two aspects of unique existence of a tuple in a relation, over the global DDB, and of functional semantic related to the location of a tuple in the DDB, determine the particularity of the Global Query problem in this DDB, in comparison with the general case discussed for Global Query in a DDBS (Par. 1.1).

In this case, a simplification factor, compared with the general case discussion (Par. 1.1), is that, since no duplication is allowed in a relation, there is no problem about the choice of the correct, up-to-date copy (problem of consistency of multiple copies, concurrently updated) or the choice of the most convenient copy (problem of efficiency, in terms of minimizing network subsystem load) to access for processing a Global Query. A great simplicity is discovered also in the case of allowing Global Update, since only a copy of each tuple is to be update, safeguarding the consistency constraint.

But there are also complexity factors (not just related to the logical complexity, but also to the resulting overhead).

First of all, no complete relation can be kept in a single location, since the tuples are spread over multiple locations (OFS stations).

In this situation, as pointed out in Par. 1.1, a large amount of communication activities, through the network, can be necessary for a GQ, if we do not put any restriction.

Another important point, related to complexity, is that, being the tuples located in a particular "local relation" for the particular function of the OFS station that own it and not for particular properties of their attribute values, there is no general knowledge in the DDB System about the distribution of tuples in the DS according to some qualifications of their contents (attribute values), so it is impossible to direct a Global Query to a subset of satellites (namely, station relations) according to some qualifications in the Global Query request.

To explain this difference, it can be considered, for example, a DDB

where, in the same relation, all tuples, with the value of the attribute  $A_i$  in the range  $R$ , are located at the same station  $S$ . In this case, a GQ on this distributed relation, with the qualification: "*where  $A_i=r$  and etc.*", with  $r$  belonging to  $R$ , could be limited to the DB on station  $S$ .

This lack of general knowledge in the DDB System makes necessary that all satellites are to be involved in any Global Query, and all their DBs are to be searched.

But, due to functional semantic related to the OFS station where a tuple is located, it must be allowed that the range of a Global Query (that is, the OFS satellites to which this GQ request is broadcasted) can be limited by a specific request, included in the GQ statement, from the OFS station which issues this Global Query. The meaning of this explicitly requested limitation is fully understood by the user, or the application OFS program, issuing this GQ.

There are two important limitations in comparison to a general GQ: these limitations permit, in this DDB system, to avoid the great amount of complexity involved in processing some GQs, as discussed in Par. 1.1.

The first limitation is that no GQ is allowed where there is requested a join between two attributes of two different relations (problem of Example-1, Par. 1.1), that is where there is requested a comparison between two fields of two different form types (join limitation).

The second limitation is that, even if a Global Query, inside a single relation, logically should involve together more than one local relation in more than one satellite (problem of Example-2, Par. 1.1) (that means, in this DDB system, that all local relations, in every satellite, should be involved), a GQ is processed as it was separately directed to each local relation, and intended to be limited to each local relation, without any kind of interaction among different satellites and different local DBs (cross-over limitation).

In such a way, with join limitation we avoid the difficult solution of the problem shown in Example-2, Par. 1.1, with cross-over limitation we avoid the as much difficult solution of the problem shown in Example-2, Par. 1.1.

Nevertheless these two limitations can be accepted in this particular DDB System, which must be the support of the OFS, that is an office automation system intended to manipulate forms; but they could not probably be accepted in some other particular DDB.

In effect it results that occurrences of GQ qualified on more than one form type, not allowed by the join limitation, or occurrences of GQ qualified over the results of an inner GQ on the same form type, not allowed by the cross-over limitation, are very seldom necessary. They are often operations with no, or only poor, meaning, in relation to the OFS form system; so they, when not avoided, could be performed with a succession of GQs or with other operations pertinent to the OFS environment

(for example, the use of join-form types for the most required join query operations).

With these limitations the processing of a GQ, in such a DDB, is split into the coordinated processing of a corresponding Query in each interested local DB, or better, in the contained local relation. These resulting "local queries" have to be coordinated in a distributed algorithm, but they do not require any interaction of data base activities with other parallel local queries, arisen out of the same GQ. That is, the local query does not depend from the result of other local queries (as explicitly forbidden by join and cross-over limitations), but they are to be synchronized, in time and range of application, by a global distributed algorithm, implementing the GQ processing, for the correctness and consistency of the result.

In this way, the acceptability of these limitations make possible to study and design an effective solution (namely, that distributed algorithm) of the GQ problem for this DDB system, intended to be implemented in a Distributed System of limited processing power, based on Mini and Micro systems connected in a network.

This solution, unlike other solutions to the GQ problem in DDB, already implemented (see the discussion in Par. 1.1), is expected to have acceptable performance characteristics, also in a distributed environment with little powerful Processing Units.

A final remark can be made about the problem of Global Update. In this particular DDB system, the usually most difficult problem with which an update operation has to deal in a distributed environment, that is the consistency of various (partial) copies of the same relation, is here avoided since there is no tuple duplication. Furthermore, acting within the limitations posed for Global Query, the processing of Global Update is technically straightforward and quite similar. There could be exploited the same distributed algorithm, used for the Global Query, with minor modification in the handling of data accessed (designed attribute values, namely form fields, are changed instead of simply accessed for collecting data).

But this problem is probably to be discussed in the view of the OFS system: a Global Update may result in a change of some fields in all forms of a certain type, which satisfy the requested qualification; this result clashes with the OFS assumption that a form is owned by the OFS station where it resides, and only this station (user on-line or an application program) can make changes in it, until this form is shipped to another station. To comply with this OFS functional assumption, the use of Global Update could be restricted to some specially acknowledged OFS station, available only to someone like a "super-user", with managing purpose.

For the previous reasons, we have decided to implement also Global Update activities, but in such a restricted way.

### **1.5 Requirements for the algorithm implementing GR.**

- Correctness and behaviour requirements for the global algorithm implementing Global Requests in the OFS Distributed Data Base System.

A logical correctness requirement and a global behaviour requirement can be stated for the algorithm implementing the processing of both Global Query and Global Update (we assume the case that the Global Update service is implemented in this DDB system, but it is allowed to a few selected OFS stations).

Global Query and Global Update are called together Global Requests (GR).

### 1.5.1 Correctness requirement

Each form occurrence (tuple), of the specified form type (relation), must be *logically\_searched* in processing a GR, and this operation must be done no more than once; that is, each form must be logically counted exactly once, in each global operation.

That means no tuple, in the relation involved, must be missed, in the whole DS, but no tuple must be considered, for example, in two or more different locations (local relations) in the DDB.

In fact it is to be observed that, in a distributed environment, other concurrent operations keep going on during the period of time a GR is operating, so a form occurrence could be transferred from a local location to another, to a different satellite.

The GR algorithm must correctly handle these situations.

We have used the expression "logical searching" of tuples, about the processing of GR, because it is not required to physically access all tuples in local relations (local DBs), since, by the use of indices, it is sometimes possible to access, in a local DB, the subset of the local relation specified by the GR qualification.

It must be remarked here that a GR does not move forms within the DDB, since, in conformity to OFS specification, only the OFS station can do it; a GR can only collect data from forms, if a GQ, or change data in forms, if a QU.

The policy adopted to deal with the correctness requirement is to develop an intelligent algorithm which can prevent the multiple searching of the same form (besides assuring it is really searched), rather than to use the unique existence of the key associated with each form to discover multiple fetched items (second possible policy).

There are several reasons for that choice.

First of all in many GQ there is not the key among the form fields requested; so, with the adopted policy, we do not need to add the key to each item collected. With the other policy the key is necessary to detect, in the node of DS where all local answers are gathered, the presence of items coming from the same form, accessed in several local relations.

For the same reason, with the adopted policy, it is possible to process locally (in each satellite) data collected by a GQ, for example counting these items or performing mathematical operations on them, without carrying, with the local result, the keys of forms involved.

In this way it is sometimes possible to greatly reduce the amount of data flow, associated to GQ processing, in the network. Moreover it is greatly reduced the workload necessary to build a GR answer from the several local answer.

With the adopted policy, in the node of the DS, where all local answers to the same GR are gathered (it will be, for the particular function distribution of this DS, the central node designed as "host" - Par. 1.3) to build the global answer, it is only necessary to collect together the various item sets (GR answers from the satellites involved) received.

In this case of the second policy, to build the global answer it is necessary to merge the various item sets, basing on the associated key, to detect and erase the duplicated items. This task can be very costly, in terms of workload in the gathering node of DS (that is, the host node), if a great number of item sets is to be merged.

Nevertheless it can be observed that a synchronization at functional level, among the parallel operation on the satellites for a GR, is necessary even if the second policy is adopted to avoid missing some forms in transit in the DS, since the use of key in each item collected only permits the detection of duplicated items, not the detection of missing items.

Thus, the conclusive reason for adopting the first policy is that, working at the design of a reasonable solution, it has been found that it was necessary a relatively little improvement to the complexity of the algorithm for GR processing to allow it, besides avoiding to miss any form, also detecting duplicated items, without the extensive use of keys.



### 1.5.2 Behaviour requirement

This requirement is a usual assumption for algorithms which have to work in a distributed environment.

It requires that the actions of a distributed algorithm do not block the overall behaviour of the DS, that is the other parallel activities in the various nodes of the DS.

In the particular case of OFS system, this requirement means that the activities resulting from the execution of a GR must not cause the blocking of the OFS stations, like form creation (getting a global key from the host node) and form moving ("shipping" and "mailing", from a satellite to another one, through the DB-trays in the host).

Obviously, the temporary blocking of satellites distributed activities is allowed (not local activities of OFS station), while each satellite is performing its portion of a GR.

To deal with this behaviour requirement, two different policies have been studied; they led to the design of two different algorithms: algorithm Alpha and algorithm Beta.

Both algorithms may involve some temporary blocking of distributed activities of OFS satellites, even if these blocking situations are of quite different types.

Algorithm Alpha can involve some temporary partial splitting of the DB-trays and different accessing ways to them (that may concern the local relation in the DB-trays involved in the GR) in the host node.

Besides, algorithm Beta presents the advantage of a larger distribution of functions among concurrent cooperating processes in the important host node, but with the trade-off of possible larger overall blocking periods if the DS is not enough powerful or well balanced (namely, if there are nodes quite slower than others or if the host node is too small).

**NOTE:**

The algorithm Alpha is simpler to implement because of the existing OFS distributed features, above all for the actual centralized implementation of the DB-trays, so it is expected to be implemented as first; but the algorithm Beta is cleaner from the logical view-point, even if its implementation requires some change in the physical structure of DB-tray in the host node, so it is expected to be implemented as second.

However it seems more appropriate to state a difference of the efficiency of these two algorithms in terms of the physical configuration of the DS on which the OFS DDB system is expected to run: the algorithm

Beta could be more useful and effective on a DS with a more powerful host node and more homogeneous satellite nodes.

Finally an observation can be made about the general policy of the GR algorithm (both types Alpha and Beta): in the whole DDB system the GRs are performed serially, in such a way that only one GR can be in execution at any time.

This main assumption is due to the need of not overloading the DS with many concurrent GR operations, each of them could involve a great load of processing and communications activities in the global DS, that could excessively slow down or even stop the other concurrent activities, both distributed and local, of the satellites (OFS stations).

The policy of global serialization of GR greatly helps in the determination of the consistency point of each GR (Par. 1.1), that is the point corresponding to the situation, in the evolution of the DDB, which the answer of a GR refers to.

In fact, since the consistency point is bounded by the initial time when the GR begins the execution in the whole DS (accordingly to a GR centralized serialization mechanism, in the host node), and the final time when the GR execution ends and the global answer is collected, narrower is this interval, better is the determination of the consistency point.

The adopted general policy, if from one side can have a worse GR throughput average time, from the other side has certainly the narrowest time interval for GR execution and so the best consistency point determination, since several GR executions, being serialized, cannot overlap and slow down each other.

## **2 Syntax and semantic of Global Requests in OFS.**

- Presentation of syntax and semantic of Global Requests allowed in the OFS Distributed Data Base System.

### **2.1 Syntax of Global Request.**

The syntax of the Global Requests (Global Query and Global Update) is presented in Bakus Normal Form notation, as requested by the compiler-compiler YACC (Ref.8), used to build the parser and the syntactical analyzer for such GRs in this system. YACC converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm.

SYNTAX

```
global_request: statement ';'
;

statement: global_query
| global_update
;

global_query: globalq query_item =>
=> from what max_transfer where_clause
;

global_update: globalup what giving =>
=> set set_clause_list where_clause
;

query_item: form_oriented_query
| unique_count_query
| math_query
;

form_oriented_query: station image return_count
| station field_list return_count
;

station:
| station
;

return_count:
| retcount
;

field_list: field '(' field_number_list ')'
;

field_number_list: field_number
| field_number ',' field_number_list
;

field_number: {integer}
;
```

```
unique_count_query: unique field_list
    | count
    | count unique field_list
    ;
```

```
math_query: station key select_extreme =>
    => numeric_field_name
    | count sum product square_sum =>
    => numeric_field_name
    ;
```

```
key:
    | key
    ;
```

```
select_extreme: max
    | min
    ;
```

```
sum:
    | sum
    ;
```

```
product:
    | product
    ;
```

```
square_sum:
    | squaresum
    ;
```

```
numeric_field_name: field field_number
    ;
```

```
field_name: field field_number
    ;
```

```
what: form_type
    | form_type range '(' station_list ')'
    ;
```

```
form_type: {identifier}
    ;
```

```
station_list: station_id  
             | station_id ',' station_list  
             ;
```

```
station_id: {identifier}  
           ;
```

```
max_transfer:  
             | mazonline {integer}  
             ;
```

```
giving:  
        | retcount  
        | retcount station key max_transfer  
        ;
```

```
set_clause_list: set_clause  
                | set_clause ',' set_clause_list  
                ;
```

```
set_clause: field_name '=' constant  
           ;
```

```
constant: string_constant  
         | null_constant  
         ;
```

```
string_constant: {quoted string}  
               ;
```

```
null_constant: null  
              ;
```

"where\_clause" Qualification - Case A.

```
where_clause:  
            | where ofs_predicate  
            ;
```

```
ofs_predicate: field_name ofs_comp_op string_constant  
             ;
```

```
ofs_comp_op: '='
| '>'
| '<'
| '>' '='
| '<' '='
| match
;
```

"where\_clause" Qualification - Case B.

```
where_clause:
| where boolean
;
```

```
boolean: '(' boolean ')'
| boolean or boolean
| boolean and boolean
| predicate
;
```

```
predicate: field_name comp_op field_name
| field_name comp_op string_constant
| numeric_field_name num_comp_op =>
=> numeric_constant
| field_name null_comp_op null_constant
| field_name in_op nested_query
;
```

```
numeric_constant: numeric {quoted string}
;
```

```
comp_op: num_comp_op
| match
| '~' match
| smatch
| '~' smatch
;
```

```
num_comp_op: '='
| '>'
| '<'
;
```

```
| '>' ='  
| '<' ='  
| '~' ='  
|
```

```
null_comp_op: '='  
| '~' ='  
|
```

```
in_op: in  
| '~' in  
|
```

```
nested_query: select field_name where_clause  
|
```



## 2.2 Semantic of Global Request.

### *Data types in form fields*

Before discussing how to make a Global Request, it is suitable to discuss the data types of form fields, that is, the data types of attribute values.

Every field, in every form type, is formally a string of ASCII characters; but each field can be interpreted in two different ways:

- 1) as a "string constant", that is, the content is not interpreted when processed (for example, the comparison is done character by character);
- 2) as a "numeric constant", that is, the content is interpreted as a rational number when processed (for example, the comparison is done on the rational number obtained translating the ASCII string, moreover this number can be used also for mathematical operations, like sum, product, etc.).

It must be noted that a string constant can be a rational number, in ASCII representation.

A form field is called numeric field, when it is expected to contain a numeric constant, instead it is called string field (or simply field), when it is expected to contain a string constant.

It is clear that the same form field can be called numeric field when we consider the content as a numeric constant, and (string) field when we consider the content as a string constant.

By the syntax of a GR, as previously presented, it is always possible to differentiate a numeric field from a string field. So the GR processing algorithm, if it expects a numeric constant in a particular form field, can test the ASCII string contained in this field to know if it holds a rational number. In the affirmative case, the string is translated to the numeric value and the processing (comparison or mathematical operations) is done on this value. In the negative case, an error is detected and the GR fails (the incorrect form can be signaled out).

If the GR algorithm expects a string constant, it will process the ASCII string in the form field in per-character basis, even if it really holds a rational number.

It seems nevertheless proper that these two kinds of form fields must be differentiated at form definition time. So the OFS user interface could acknowledge the user about the type of data content each form field can accept, and it could make a control on data that the user tries to insert in such fields.

## 2.2.1 Global Query semantic.

In a Global Query the following parts can be distinguished:

- Part A) *Query\_object* ("query\_item")
- Part B) *Query\_source* ("what", "max\_transfer")
- Part C) *Query\_qualification* ("where\_clause")

### *Part A:*

The query object consists in the specification of data to be collected and of the kind of processing required on them.

We have three types of query objects.

#### A,1) *Form\_oriented\_query:*

We can require either the image (*image*) of the form, that is the content of all the form fields, in such a way that the form can be fully recomposed in the requesting OFS station, or a subset of the form ("field\_list": *field*(N.1,...,N.k)), that is the content of fields N.1,...,N.k, where N.i is the number of field in the form (number 0 is always the key). In the latter case, only the fields requested are shown at the OFS station which started the GR.

We define now an "item" as the set of data, collected by a GQ, taken from a single form.

In both previous cases, we can request (*station*) that another field, not included in form type definition, is added to each item collected. This field will hold the name of the station where the item was found.

In both previous cases, we can also request (*retcount*) the total number of items collected by the GQ.

#### A,2) *Unique\_count\_query:*

This kind of query object is included only because it is common in relational data query languages (it is taken from SLQ of MRS).

The option *unique* requires the elimination of duplication of items collected ("field\_list" specifies the fields wanted). Of course this option is useless if the field 0 (form key) is specified.

With the option *count* we require only the number of items identified (according to the qualification), but not collected.

The option *count\_unique* combines the two effects: it requires the number of items identified, but without duplication, and they are not collected.

#### A,3) *Mathematical\_query:*

This kind of query object is most interesting in OFS environment. A target numeric field is pointed out (*field N*) and, on the rational numeric value of this field, one or more mathematical operations are locally performed, giving at the end a single global result. Namely, the operation is applied on each local relation, according to the form qualification, and finally the same operation is applied on all local results, giving a global result.

There are two types of such a request.

In the first one, we can require the maximum (*max*) or the minimum (*min*) arithmetical value of the field, among all the forms indicated in the DDB. We can optionally require the key (*key*) of the form holding the found value, and the station name (*station*) of the OFS station where that form is located.

In the second type, we require the total number (*count*) of numerical values processed (this is also the number of forms matching the qualification, and so holding these values in the target field), plus any combination of the results of the following arithmetical operations:

*sum*: sum of these values;

*product*: product of these values;

*squaresum*: sum of squares of these values.

It must be remarked that with the *count*, *sum*, *product* and *squaresum* values, we can easily calculate several very interesting statistical indices of the target field of the forms qualified, such the confidence interval of statistical mean, of statistical variance, etc.. That is done without collecting in a single OFS station all the data involved.

#### *Part B:*

The query source contains the specification of the form type interested in the GQ (its name, among those globally defined in OFS DS).

The GQ is usually broadcasted to all connected satellites (in the DS), but in the query source it is possible to limit the range of OFS stations to which we want to route our GQ. This is done with the option *range* (*station.1,...,station.k*), showing the list of station names interested in this GQ. In effect only a subset of these stations can be effectively interested in this GQ, in the case of a few stations are not actually connected to the DS.

An answer to a GQ could consist, in some case, in a very large amount of data; so it can be useful, whether for the network, which could be overloaded, or for the OFS end-user, which could not handle such amount of data, to print off-line the result of a GQ.

This situation can be explicitly controlled issuing a GQ, by the option *maxonline N*, where N is the maximum number of items the GQ answer

can have in order to be allowed into the requesting OFS station. If the GQ answer has more than N items, it is wholly transferred to an off-line printer, and the interested OFS station is correctly warned.

If this option is not used, the system makes use of its own default value, to control the maximum amount for a GQ answer in order to be sent to the issuing OFS station.

#### *Part C:*

The query qualification contains the specifications of the forms, within the same type, which are to be involved in the GQ operation.

For the query qualification we have two implementation steps.

The first simpler one corresponds to the Case A (Syntax of GR, Par. 2.1).

In this case the qualification is completely similar to the selective search mechanism actually implemented in the OFS stations. This qualification, when present (if not present, all the forms of the indicated type are searched), is merely composed by *where* and an "ofs\_predicate": that is, the specification of the field in the form (*field N*), the specification of an operator allowed by OFS (=, >, <, >=, <=, *match*), and a string constant.

In this case we consider the form field only of string type, so all the OFS operators are applied in per-character, non numeric basis.

The second, more complex, implementation step correspond to the Case B (Syntax of GR, Par. 2.1).

In this case the qualification is almost completely similar to the qualification part of a relational query language, exactly to the "where\_clause" specification of the SLQ of MRS (Ref.3).

The only difference concerns the data types. In MRS there are two different data representations: string and word binary numeric. In the DDB of OFS there is only a data type, equal to the MRS string type, but with two possible meanings: string constant and numeric constant.

A constant, namely a quoted string, is usually assumed here as a string constant, so when we want a numeric constant, we have to specify *numeric* before the quoted string.

In this way we can differentiate, in the "predicate" B.N.F. production, the case of comparison (in per-character basis) of a string field ("field\_name") with a string constant, from the case of comparison (in arithmetic way) of a numeric field ("numeric\_field\_name") with a numeric constant.

## 2.2.2 Global Update semantic.

In a Global Update the following parts can be distinguished:

- Part A) *Update\_direction* ("what")
- Part B) *Update\_answer\_request* ("giving")
- Part C) *Update\_object* ("set\_clause\_list")
- Part D) *Update\_qualification* ("where\_clause")

### *Part A:*

The update direction contains the specification of the form type to which this GU is directed, and, optionally, the range of OFS stations we want to be involved in this global operation.

The update direction specification is completely equal to the query source specification.

### *Part B:*

The update answer request contains the specifications of the kind of answer we want at the completion of this GU.

If we do not use such option, we will simply receive a completion message at the end of this GU processing.

If we specify the *retcount* option, we will receive, as answer, the number of forms updated in the whole DS.

If we specify *retcount station key*, we will receive, as answer, the total number of forms updated and an information item for each form updated. Each item contains the station name where the form was found, and the key of this form.

Since all these items could constitute a large amount of data, here it is possible to use the option *maxonline N*, to direct the output to an off-line printer when the answer contains more than N items (of course, if this option is not used, the system uses its own default value).

### *Part C:*

The update object consists in the specifications of the changes to be made on the data contents of the specified fields of the forms accessed.

The update object is in the format:

*set field N.1 = C.1, ... , field N.k = C.k*

In this way, we require that the contents of the fields N.1,...,N.k in all the forms accessed according to the qualification, are changed respectively to the value of the constants C.1,...,C.k where each constant can be a

string constant (quoted string) or a null constant (*null*).

In OFS there are three types of form fields, defined at the moment of form type creation:

- fields filled in by the OFS system (as key, date, OFS station signature);
- fields whose content cannot be changed, after it has been filled in once;
- fields whose content can be freely changed.

It is evident that only form fields of the third type can be updated, and so they only can be specified in a GU, otherwise the GU is rejected (this check can be made in the OFS station, since all form type definitions are global in the DDB system).

*Part D:*

The update qualification contains the specification of the forms, within the same type, which are to be involved in the GU operation.

The update qualification is exactly equal to the query qualification.

NOTE:

For every correct GR answer, there are shown the interval time limits (lower and upper bound) of the consistency point, measured in the host node.

### 3 Design and implementation of OFS DDB System.

- Design and implementation of the OFS Distributed Data Base System performing Global Requests.

#### 3.1 Logical and physical models.

- Logical and physical models of the Distributed System structure.

##### 3.1.1 Logical model.

We have already outlined (Par. 1.3) the logical structure of the DDB underlying the OFS Distributed System (Fig.1). In Fig.2 there is shown the logical structure of the OFS Distributed System itself, now a real DDBS.

That is essentially the logical model according to the concurrent and cooperating functions are organized, in such a way that it is possible to perform, besides the other parallel distributed activities, like creating and moving forms, the GRs too. We can observe that the DDB shown in Fig.1 completely fits the DS structure shown in Fig.2.

Let us describe the functional model of this DDBS (Fig.2). As it results from previous discussion (Par. 1.3) the DS is composed by a host node, called *logical\_host*, and several satellite nodes, called *logical\_satellites*, which are connected, through a logical communication subsystem, in a 1:N way. That is, each logical satellite is connected to, and only to, the logical host: so a star type logical structure <sup>(Ref. 14)</sup> comes out (Fig.3). Obviously, the star type topology of the logical communication network does not imply that the physical communication network must have the same topology; no limitation is imposed on the physical network structure. In each logical satellite two main functional modules are present (a functional module can be either just a process or a group of tightly cooperating processes): the OFS-Satellite Local Functions (OFS-SLF) module and the OFS-Satellite Distributed Functions (OFS-SDF) module. The OFS activities

of local nature are concentrated in the OFS-SLF module, whereas the OFS activities of distributed nature, that is involving the whole DS, are concentrated in the OFS-SDF module.

In OFS-SLF are fully performed all activities allowed in the previous OFS stations (Ref.4), as well as the new activities which are expected to be implemented in OFS stations, such the "sketches" (Ref.5), but only if these activities do not involve the rest of the DS, that is if they are not of distributed kind. For the latter ones, the OFS-SLF must require the collaboration of the OFS-SDF, through the communication path within the satellite.

Distributed activities are here intended the previous distributed activities of an OFS station, such "creating" (requiring a unique global key for a new form), "shipping" (sending a form to a DB-tray, in the logical host, of another satellite), "mailing" (fetching forms from the DB-tray, in the logical host, owned by the satellite), "tracing" (retrieving a trace of movements of a form among the satellites, from the LOG global relation in the logical host) forms, as well as the new distributed activities corresponding to the GRs, such GQs and GUs.

In the OFS-SLF module there is a User Interface (UI) submodule which handles the interaction with the end-user (if this satellite can support an on-line end-user and it is not an automatic station) in a "environment specification by successive menus" way.

The introduction of GR in UI consists simply in the introduction of a new entry in the highest level menu, specificating the GR environment. In this environment the GR can be entered in accordance to the syntax specified in Par. 2.1. At this level a change is forecasted to switch from this Relational DB like access language for GR to a form oriented access language, similar to Query By Example (Par. 1.3).

In OFS-SDF two different sets of activities can be distinguished: the set of activities caused by distributed requests (any possible distributed activity, as previously specified) from the local OFS-SLF, which in order to be completed are to be routed to the corresponding OFS-HDF in the logical host, and the set of activities caused by a Global Request from another satellite in the DS, concerning also this station, which is routed to this module from the corresponding OFS-HDF in the logical host. This latter set of activities of OFS-SDF consists mainly in performing that GR on the DBs located in this satellite (that is, on the satellite relation).

It is possible, for this second kind of activities by OFS-SDF, to interfere with the concurrent activities in progress on OFS-SLF in accessing the DBs in the satellite. So a problem of concurrent access to shared local DB rise up.

The access control is realized by special locks associated to every relation in every local DB; these special locks do allow concurrent query operations but do not allow concurrent updates, neither query and update operations.



The communication path between OFS-SLF and OFS-SDF, in each satellite, is organized on three levels.

The first one is a synchronization line which allows these modules to asynchronously interrupt each other, sending a simple signal with no data associated.

The second one is a two ways direct data communication line, with internal first-in-first-out queuing policy. This path is used for fast communication of limited amounts of unstructured data, such request from OFS-SLF and short answers from OFS-SDF. Unfortunately, this facility (called "pipe" under the UNIX operating system) is not available in all the satellite implementations, as further explained in Par. 3.1.3.

Thus, in the least powerful satellites, the third level communication path is also used for this second kind of communications.

The third level consists in a set of data structures, on secondary storage, called "Local Communication Data Structures" (LCDS), which are shared by both OFS-SLF and OFS-SDF.

This path is the most useful for exchanging relevant amounts of structured data, especially complex answers to OFS-SLF or complex distributed requests (if allowed) to OFS-SDF.

Each OFS-SDF module, in a logical satellite, is tightly coupled, from the functional viewpoint, to a OFS-HDF module, in the logical host. There is a biunivocal correspondence between an OFS-SDF module, in the  $i$ .th satellite, and the corresponding OFS-HDF module in the host; so that module is called OFS-HDF. $i$ .

These two modules, OFS-SDF and OFS-HDF. $i$ , although located in two different components of DS (a satellite and the host), essentially work together to perform the same distributed activities.

Nevertheless, while the link connecting these two modules is so important for their behaviour, it is also the most difficult type of link to be correctly implemented because this is the only communication path, in the whole DS, which can cross the communication network connecting physically distinct DS nodes.

Thus, in each OFS-SDF and in the corresponding OFS-HDF, there is a Physical Network Interface (PNI) submodule (a couple for each link, one at each link end) aiming to correctly handle possibly different communication link types, and so to insulate OFS-SDF and OFS-HDF behaviour from the knowledge of the communication link peculiarities. PNI functions and types are fully described in Par. 3.1.2.

Since the OFS-HDF. $i$  module is the functional complement, in the host, of the OFS-SDF module in the  $i$ .th satellite, it will be intended to perform the same kind of activities of the associated OFS-SDF module. Thus the OFS-HDF. $i$  module can execute two sets of activities: a first set of activities is caused by a request from the associated OFS-SDF module, of any possible distributed type, which will involve the rest of the DS; a second

set of activities is caused by a request from another satellite, through its corresponding OFS-HDF in the host, which will involve the  $i$ .th satellite, through its OFS-SDF module.

In that first set of distributed activities, there are the following activities.

- The OFS-SDF module requests a new global key for the creation of a new form (or for copying an existing form), of  $j$ .th type, in a DB belonging to the  $i$ .th satellite. The corresponding OFS-HDF. $i$  module, in the host, makes an access to the "Global Control Data Structures" (GCDS), a set of structured data on secondary storage of the host, shared by all the OFS-HDF modules, containing all relevant control data of the global DS. Then it will pick up the new available key for the  $j$ .th global relation and it will send it back to the corresponding OFS-SDF module in the  $i$ .th satellite.
- The OFS-SDF module request to retrieve the trace of the movements of a specified form through the DS satellites.  
The corresponding OFS-HDF. $i$  makes an access to the LOG Relation in GCDS and select the tuples recording movements of that form, then it send back this answer.
- The OFS-SDF module, in the  $i$ .th satellite, requests to send a particular form, of  $k$ .th type to the  $j$ .th satellite.  
The corresponding OFS-HDF. $i$  module will insert that form in the  $k$ .th relation of the tray of the  $j$ .th satellite, that is, in the DBt. $j$  (DBt. $j$  denotes the DB-tray belonging to the  $j$ .th satellite). Then it makes an access to GCDS, inserting a tuple, in LOG Relation, recording the movement of that form.
- The OFS-SDF module, in the  $i$ .th satellite, requests to receive forms previously sent to this satellite from other ones.  
The corresponding OFS-HDF. $i$  module will fetch the forms, as requested, from interested relations in DBt. $i$ , that is, the tray of the  $i$ .th satellite. The LOG Relation is updated in GCDS, indicating that these forms have been received in the  $i$ .th satellite, and the forms fetched are sent to the OFS-SDF module.
- The OFS-SDF module sends a Global Request, either a Global Query or a Global Update, from the  $i$ .th satellite.  
The corresponding OFS-HDF. $i$  module must now perform this GR with the help of the other OFS-HDF modules in the host, and through them, with the collaboration of the other involved satellites in the DS.

How can OFS-HDF.i broadcast this GR? How can it synchronize its activity to the concurrent activities of the other OFS-HDF modules, in the host, which help it for this GR? How can it do that without blocking the parallel activities of other OFS-HDF modules executing other distributed functions, not related to this GR? How can it correctly gather all the answers, to the GR, received from the other involved satellites, through their corresponding OFS-HDF modules? How can it be sure that the concurrent operation requested by the GR are performed in a correct and consistent way?

The answers to all those questions are the specifications of the two algorithms (Alpha and Beta) proposed for the implementation of GRs.

The second set of activities a OFS-HDF module, corresponding to the i.th satellite, can execute, simply consists in the help, as previously outlined, which some other OFS-HDF can request from this OFS-HDF.i, to perform a GR involving the i.th satellite.

In this case the OFS-HDF.i module must send the GR received to the linked OFS-SDF module, in the i.th satellite. That OFS-SDF module will perform the request on the DBs local of the satellite and will send back the local answer to the OFS-HDF.i in the logical host. Then the local answer, from the i.th satellite, will be added to the global answer of that GR, in GCDS, by the OFS-HDF.i module.

Obviously, the detailed correct operations this OFS-HDF module must perform in this case, are specified in the two algorithms (Alpha and Beta) proposed for the implementation of GRs (Par. 1.5.2 and Par. 3.2).

At this point, it is possible to make an important remark about the logical host of the DS.

The logical host in practice does not exist as a separated and independent executive entity: it is simply the sum of activities of OFS-HDF modules, corresponding to the various logical satellites of the DS. Local to the logical host we can consider the shared data (DB-trays and GCDS) on which those activities are performed and synchronized.

### 3.1.2 Physical model.

The general physical structure is shown in Fig.4.

We can obviously have many particular physical implementations for the same logical structure, but some constraints are to be observed.

It is possible to change the type of the machines and the type of the communication network, but the reciprocal interactions are to be carefully considered to obtain a global DS with acceptable performance characteristics.

An important matter is the choice of the machines for the physical host and for the physical satellites.

Each physical satellite must bear a logical satellite (that is, an OFS station with several local DB and an eventual end-user line), of remote type. Therefore, the processing power and mass storage size and speed of this machine are to be chosen considering the functional characteristics of the corresponding logical satellite (esteemed workload and DBs dimension).

The physical host is the true critical component of the DS.

In fact, if the failure of a satellite machine can be discovered and suitably handled by the host machine, without blocking the overall DS, the failure of the host machine causes the crush of the DS, and only the OFS local (not distributed) activities in the satellites can keep going on.

Moreover the physical host is a critical component for performance consideration, too.

In fact, every distributed activity of every logical satellite (creating and moving form, as well as GRs) does certainly involve the physical host of the DS. Hence, the host machine must likely be the largest and most powerful machine in whole DS.

Another important characteristic of how the logical structure is distributed in the physical structure, is that some logical satellites can be contained in the physical host. These logical satellites, of type local, are functionally identical to the logical satellites of type remote, namely running on the physical satellites; so each of them implements an OFS station, owning several DBs and holding an eventual end-user line.

The purpose of these local logical satellites is to keep some important OFS station (for example these with OFS managerial aims, to which it could be allowed the use of GU) in the same machine bearing the logical host, which performs the distributed non-local functions. In this way, such local logical satellites are not dependent from the communication network for accessing the logical host, so whichever communication link failure does not affect their correct behaviour.

Hence, as shown in Fig.4, the physical host must bear the logical host an

several local logical satellites: so its processing power and its secondary storage system have to be enough large to prevent it from becoming the bottle-neck of the whole DS.

Another important matter is the kind of communication network connecting the various node (host and satellites) in the DS.

The choice of the type of communication network depends, first of all, from the distribution scale, if local or geographically dispersed, of the global system.

Obviously, this choice also affects the determination of certain internal parameters (for example , the time the host usually waits for an answer from a remote satellite before starting the procedure for remote satellite or communication path failure) in the distributed algorithm, in the logical host as well as in the remote logical satellites. In the actual implementation the choice has been for a local environment, so switching to a geographically dispersed environment, these internal parameters are to be set again.

Often, implementing a DS, we need to use an existing communication network which can range from a true communication subsystem, with a relatively high level access interface, reflecting a high level communication protocol, to a bare set of physical data transmission links and physical ports on DS nodes, with no communication software package available.

The features of the DS dependent from the type of the communication network available are insulated in a localized module: the Physical Network Interface (PNI), as shown in Fig.2.

In fact, as shown in the logical structure of DS (Fig.2), the only interprocess communication link which does pass through the communication network is the one between the module containing the "OFS Satellite Distributed Functions", in the logical satellite, and the module containing the "OFS Host Distributed Functions", of that satellite, in the logical host.

These two modules have been provided with two specular submodules, one in the OFS-SDF module and the other in the OFS-HDF module, which do perform the requested communication services, accessing the particular communication network in conformity to its own specifications.

The PNI couples are present on the ends of every link between an OFS-HDF module and an OFS-SDF module, whether in a remote or in a local logical satellite.

Therefore the PNI submodules, implementing a common interface to all OFS-HDF and OFS-SDF modules, realize the total independence of the logical host and logical satellites from the communication network features, as well as the total identification of a local and remote logical satellites (in fact, all the differences are restricted within the PNI submodules).

The assumptions of the external interface of PNI submodules are mainly two. As first, a logical link must be permanently established between the logical host and each (connected at the moment) logical satellite, that is

between each couple of OFS-HDF and OFS-SDF modules: in this way the resulting logical architecture is of star type, which the logical host as central node and the logical satellites (both remote and local) as peripheral nodes. As second, the logical communication protocol of the fully-duplex link must be of type master/master with both the logical satellite (that is, its OFS-SDF module) and the logical host (that is, the corresponding OFS-HDF module) acting as masters, following a carefully stated end-to-end protocol.

There are two types of PNI couples. The first one, connects OFS-HDF and OFS-SDF modules of a local logical satellite; therefore, both modules are in the same host machine.

These PNI submodules do not use the communication network facilities but do simply use the interprocess communication and synchronization tools, provided by the operating system on the host machine, in order to build the necessary internal communication link.

The second type of PNI couples connects OFS-HDF and OFS-SDF modules of a remote satellite; therefore, one is in the host machine and the other is in a satellite machine.

These PNI modules strictly depend from the existing communication network.

In case of complex communication subsystem, with a communication software package providing a high level access protocol, the PNI function is simply to translate the primitives of the upper interface (with OFS-HDF or OFS-SDF modules) to the primitives of the lower interface (with the software access interface of the communication network). That is usually an easy task, since the upper interface has quite simple features, in comparison to the usual high level communication protocols.

Instead, in the opposite case where only the bare physical communication lines and the associated ports are available as communication network, the PNI submodules must implement, even if in the simplest possible way, the essential functions for the control of a communication network, as establishing the basic message switching (or packet switching, if considered necessary) protocol, error checking by control extra-code, observing acknowledgement and resetting conventions, etc.

It is also possible to have many different communication links in the network: in this case more PNI couples types can be present at the same time, to properly handle each different link.

### 3.1.3 Implementation choices.

The reasons of the basic implementation choices were discussed and stated at the moment of the original implementation of the OFS (Ref.4).

These choices are generally still valid, but some features have now been changed or improved, in consideration of what is available today, mainly in the field of the communication network.

The machines for the physical host and physical satellites are DEC mini-computers of the family PDP-11 (Ref.9).

The physical host must be a relatively large member of this family so we have chosen a PDP-11/45 with hard disc units as mass storage. It runs under the UNIX operating system, by Bell Laboratories (Ref.10), which has also been extensively used for the software development of this DS.

For the physical satellites it is possible to choose different minicomputers, even the simplest types, like the LSI-11 group (based on the microprocessor LSI-11), on the ground of the needs of the corresponding logical satellites.

Thus, for this purpose we have used several different machines.

First of all, there is a PDP-11/50, of a size similar to the host machine, mainly because it was available.

Then there are a few PDP-11/23, of smaller size, with a micro LSI-11 as CPU, and with floppy disc units as mass storage; but they are soon expected to hold physically small, moving-head disc units of Winchester technology (manufactured for example by Shugart and Pertec Corporations), which have better performance and larger size (80 M-byte) at a reasonable cost (five thousands dollar price range). These machines run under UNIX operating system, as well as the host.

For other satellites, with quite simple expected activities, we have used the PDP-11/03, of the LSI-11 group, the smallest among the PDP-11 family, with only floppy disc units as secondary storage. These machines run under the MINI-UNIX operating system, by Bell Laboratories (Ref.11), a simpler version of UNIX, upward compatible with it.

Thus, the software of the logical host can run under UNIX operating system, as well as the software of a local logical satellite, while the software of a remote logical satellite can run, in the basic version, under UNIX operating system for all PDP-11 types except PDP-11/03, and in another version, which is slower but as completely functional as the previous one, under MINI-UNIX operating system for PDP-11/03.

All the software for the OFS DDB system is written in C programming language, by Bell Laboratories (Ref.12).

The only difference, between these two version, concerns the communications between the OFS-SDF process (satellite distributed functions) and

the OFS-SLF process (satellite local functions): in the former version it is possible to use the "pipes", interprocess communication mechanisms at core memory level, available under UNIX operating system, while in the latter version, since the "pipes" are not available under MINI-UNIX operating system, it is necessary to pass through files on mass storage (which is rather slow, being constituted, in this case, by floppy disc units).

For the OFS Distributed System it is intended to use a local communication network, with central switch, which uses a master-slave internal protocol and polling synchronization mechanism, and which can offer a quite large transmission bandwidth of the order of 800 KHZ, since it uses these DMA devices on the node (Ref.13) (Ref.14).

Nevertheless, since this communication network is now at an experimental stage, we need to temporarily use teletype lines as communication links: that is, two teletype ports, one on the host machine and the other on the satellite machine, are linked together via an electric cable. The bandwidth of this communication link is much lower than the previous one: 9600 baud.

This kind of ad hoc communication link, even if it will be in most part substituted by the previously presented communication network, will be still in use for that satellite machines which cannot be connected to that network, for example because they do not support a DMA device, like the PDP-11/03.

Thus, three types of PNI submodules couples have been implemented, to connect OFS-HDF and OFS-SDF processes.

The first one, used within the host machine for connecting a local logical satellite to the logical host, exploits a synchronization signals link of UNIX operating system and shared files, to build a data communication path between the two processes.

The other two ones are used in the host machine and in a satellite machine for connecting a remote logical satellites to the logical host. The former of these two ones, is employed when the communication path is a direct teletype link, the latter when the communication path is a logical link through the previously shown communication network.

As regards to the implementation, an important remark must be made: being the UNIX (and so the MINI-UNIX) mainly a time-sharing operating system rather than a real time operating system, it can provide rather poor interprocess synchronization and communication tools.

As to the communication, the best tool is the "pipe" (only under UNIX), but it is quite far from a real queuing message mechanism.

As to the synchronization, there is also an interrupting mechanism (among process of the same user-ID); but, given the lack of semaphores, critical regions, monitors, etc., and even shared memory areas, the only really useful synchronization tool is the use of lock files; that allows a "test and set" type operation.



Thus, with a lock file associated to a common data file, it is possible to implement a critical region on that data file; nevertheless that critical region does not have any waiting queue or any other control mechanism to assure a fair access policy.

For that reason, many implementation choices on the synchronization, within the distributed algorithms, are heavily conditioned by that kind of available synchronization tools, and so they could often look neither the simplest nor the most efficient.

### 3.2 Algorithms implementing Global Requests.

- Algorithms implementing the Global Requests and the other distributed activities in the OFS Distributed Data Base System.

#### DEFINITIONS

Distributed Data Based System components:

1) *Distributed\_System*:

- 1 Logical Host: *Host*

- M Logical Satellites: *Satellite.1, Satellite.2, ... , Satellite.M*

2) *Distributed\_Data\_Base*:

- N Relations (global relations):

*Relation.1, Relation.2, ... , Relation.N*

- M DB-trays in the host (one DB-tray for each satellite):

*DBt.1, DBt.2, ... , DBt.M (DBt.i: DB-tray of the Satellite.i)*

- N local relations for *DBt.i*, the same definitions in each DB-tray (subset of global relations, located in one DB-tray):

*Relation.1(DBt.i), Relation.2(DBt.i), ... , Relation.N(DBt.i)*

- M Satellite DBs (set of local DBs located in one Satellite):

*Sat.DB.1, Sat.DB.2, ... , Sat.DB.M*

- N Satellite.i Relations, the same definitions in each Satellite (subset of global relations, located in one Satellite):

*Relation.1(Sat-i), Relation.2(Sat-i), ... , Relation.N(Sat-i)*

- *S.i* local DBs in the *Satellite.i*, for each Satellite (local DB is a single complete DB, physically unique):

*DB(1,i), DB(2,i), ... , DB(S.i,i)*

- N local relations for *DB(i,j)*, the same definitions in each local DB (subset

of global relations, located in one local DB):

Relation.1(DB(i,j)), Relation.2(DB(i,j)), ... , Relation.N(DB(i,j)).

## LOGICAL SATELLITE BEHAVIOUR

The behaviour of the logical satellites is the same in algorithm Alpha and Beta; only the behaviour of the logical host differs between the two algorithms.

### *OFS-SLF module.*

This module implements the local function of a logical satellite. In it there are located almost all the functions of an OFS station, as previously intended, whether it supports an end-user or it is an automatic station.

By the point of view of the DDBS, the behaviour of this module is to send, through the communication path internal to the logical satellite, a distributed request to the OFS-SDF module and to wait for an answer from it, through the same communication path.

These distributed requests are:

- 1) Get a key for a new created form, or keys for a set of copies of a form.
- 2) Send a form (ship) to another satellite.
- 3) Get, selectively or in batch, forms sent to this satellite from other ones (mail).
- 4) Get traces of movements of a form among the satellites (or only the actual location of a form).
- 5) Perform a Global Request, that is a Global Query or a Global Update.

### *OFS-SDF module.*

This module implements the distributed functions within a logical satellite.

The behaviour of this module consists in handling two types of asynchronous requests: the first one is a request from the local OFS-SDF module, the second one is a request from the linked OFS-HDF.i module (if this is the Satellite.i) in the host.

OFS-SDF must control these two kinds of concurrent activities, preventing incorrect interferences.

The possible requests from the local OFS-SDF have been previously listed.

The request from OFS-HDF.i (through the PNI) is essentially a GR from another satellite (either from this satellite), through the logical host, which involves this satellite.

OFS-SDF handles the first type of activities routing the corresponding request to the linked OFS-HDF.i and then sending the reply from it, when arrived, to OFS-SLF.

Only for a request of a global key for a new form the action can be different: in fact OFS-SDF keeps a batch of global keys for each form type, so, if the batch is not empty, OFS-SDF can directly give the answer to OFS-SLF. When the batch is empty, OFS-SDF must request a new one from OFS-HDF.i.

OFS-SDF handles the second type of activities performing the received GR on Sat.DB.i, that is on the set of local DBs of the satellite. If the form type involved in the GR corresponds to Relation.j, OFS-SDF precisely performs the GR on Relation.j(Sat.i).

In doing so, it must successively lock each local relation accessed, namely Relation.j(DB(k,i)), with  $k=1,2, \dots, s.i$ , to avoid illegal concurrent access with OFS-SLF (only concurrent query activities are allowed).

At the end, OFS-SDF send the answer, local to this satellite, of that GR, to the connected OFS-HDF.i.

### LOGICAL HOST BEHAVIOUR

The logical host behaviour results by the sum and interaction of the concurrent activities of the OFS-HDF.i modules, with  $i=1,2, \dots, M$ , where  $M$  is the number of logical satellites.

The behaviour of the logical host differs between the algorithm Alpha and the algorithm Beta.

#### *OFS-HDF.i module.*

This module implements the distributed activities of the  $i$ .th logical satellite, within the logical host.

It is connected to the OFS-SDF module in the Satellite.i, and tightly cooperates with it for its activities.

The behaviour of OFS-HDF.i module consists in handling two types of asynchronous requests: the first one is a request from the linked OFS-SDF module in the satellite, through the PNI; the second one is a request from another OFS-HDF.j in the logical host.

The requests from the OFS-SDF module, have been previously listed (they essentially come from OFS-SLF module of the  $i$ .th satellite).

The possible request from another OFS-HDF.j module is essentially a GR, from the Satellite.j, which does involve the Satellite.i too.

The way these two types of activities are performed and synchronized in the algorithm Alpha and in the algorithm Beta are different.

In algorithm Beta, in comparison to the algorithm Alpha, these activities are performed in a larger distributed way, in spite of a more complex global behaviour.

### 3.2.1 Algorithm ALPHA

#### *Global data structures in the logical host*

Mail trays:

-  $DBT = \{Relation.i(DBT), \text{ with } i=1, \dots, N\}$ , where  
 $Relation.i(DBT) = \{Relation.i(DBt.j), \text{ with } j=1, \dots, M\}$

- TRAY.i (with  $i=1, \dots, M$ )

The forms of all DB-trays are contained in one only DB, called DBT. Moreover there is a relation file for each tray, called TRAY.i (with  $i=1, \dots, M$ , one for each satellite), which points out the forms, in DBT, belonging to the same tray. That is, through TRAY.i there can be distinguished, in DBT, the forms of the logical DBt.i.

- DBT.LOCK.i (with  $i=1, \dots, N$ ): Lock file for Relation.i(DBT).

- TRAY.LOCK.i (with  $i=1, \dots, M$ ): Lock file for TRAY.i.

NOTE:

On the lock files there are generally permitted two operations: lock and unlock. Locking operation consists in trying to create the lock file: if it does not already exist, it is created, and the associated file is locked. If it does already exist, the lock operation is temporarily unsuccessful. The process must sleep for a while (K seconds), and must try again to lock the file, until it succeeds.

In particular cases, when explicitly stated in the algorithm, if the lock operation does not succeed the first time, the process resumes other operations instead of going to sleep, waiting on the lock mechanism.

GCDS (Global Control Data Structures):

- GKEY: Relation file containing the actual value of the global counter for the N form types, which gives global keys for new forms or copied forms (meter).

- GKEY.LOCK: Lock file for GKEY.
- LOG: Relation file containing record of the traces of form movement through the trays and the satellites.
- LOG.LOCK: Lock file for LOG (of type allowing concurrent queries).
- GR: File containing the Global Request, already parsed and analyzed (it is in a structured form, as output from YACC), in execution in the DDBS.
- GR.LOCK: Lock file for GR.
- GA: File where there are gathered the local answers, to the actual GR, from the involved satellites.
- GA.LOCK: Lock file for GA.
- GRW.LOCK: Lock file warning a special access way to DBT, in consequence of the current GR.
- DBT-REL: Relation file which can temporarily contain some forms of the type involved in a GR and "shipped" during it.
- DBT-REL.LOCK: Lock file for DBT-REL.
- GRN: Indication of the global number of the current GR (each GR has its own progressive number).
- GRTYPE: Indication of the form type (relation) involved in the GR.
- SATMAP: Relation file where each tuple contains the necessary information about each satellite in the DS:
  - number and name;
  - if connected;
  - query class;
  - update class;
  - etc..
- GRSATMAP: Relation file where each tuple contains the necessary informations about each satellite involved in the current GR:
  - name of the satellite;
  - flag for the "master" satellite, that is that one which requested the current GR;
  - if this satellite has already begun the current GR;



- if it has already finished that execution;
- etc..

- GSM.LOCK: Lock file for GRSATMAP.

*Local data structures for each OFS-HDF.i*

- DR.i: Distributed request from OFS-SDF in Satellite.i.
- DA.i: Distributed answer to OFS-SDF.
- GA.i: Local answer, from OFS-SDF, to a GR from another satellite.
- LGRN: Number of the last GR performed by this module.

*OFS-HDF.i activity for a GR from another satellite*

NOTE:

As regards to the synchronization between the activity for a GR from another satellite and the activity for a Distributed Request from OFS-SDF of Satellite.i, there is no correctness requirement about the scheduling policy. That is, the algorithm does work whether one type of activity or the other one has the priority.

Thus we decided to give the priority to the requests from the associated satellite (in order to advantage the requests becoming from the local environment). We do not allow the preemption of requests, since it can be extremely costly in the implementation. In this way, the first arrived request is performed as first; if there is a conflict, the request from the associated satellite has priority over the GR from another satellite.

0) OFS-HDF.i receives a signal from OFS-HDF.j, indicating that the Satellite.i is involved in the current GR, by the Satellite.j.

1) Access GRSATMAP in GCDS, previously locking GSM.LOCK. Signal in GRSATMAP that Satellite.i is going to begin the GR activity. If this is the last one to do so (OFS-HDF.j, which is indicated in GRSATMAP as "master" for the GR, must not be included in this check), send a synchronization signal to OFS-HDF.j. Unlock GSM.LOCK.

2) Send the Global Request (the content of GR in GCDS) to the linked OFS-SDF module, in the Satellite.i, through the Physical Network Interface (PNI) submodule.

3) Wait and receive a reply from OFS-SDF, through PNI, and put it in GA.i. This reply contains the result of the GR performed on the DBs located in Satellite.i.

4) Access GA in GCDS, previously locking GA.LOCK, and then add the local answer in GA.i to the global answer in GA. Unlock GA.LOCK.

5) Access GRSATMAP in GCDS, previously locking GSM.LOCK. Signal in it that Satellite.i has just finished its GR activity. If it is the last one to do so (this time OFS-HDS.j, that is the "master" for the current GR, must be included in this check), send a synchronization signal to OFS-HDF.j. Unlock GSM.LOCK.

- 6) Put into LGRN the number of GR just performed, contained in GRN of GCDS.
- 7) End of this activity: wait for the next requested activity.

NOTE:

In every moment it is possible that PNI signals to OFS-HDS.i that the connection, through the communication network, with the Satellite.i fell down. In this case OFS-HDS.i must signal in SATMAP that this satellite is not connected anymore, and it stops whichever activity, waiting until the line is set up again.

*OFS-HDF.i activity for a Distributed Request  
from the associated Satellite.i*

0) OFS-HDF.i receives a signal from its PNI submodule, indicating that a Distributed Request (DR) has been sent by the corresponding OFS-SDF module in the Satellite.i. The received DR has been put into DR.i.

This step 0 is common to the following parts A, B, C, D and E.

A) DR.i is a request of an allotment of keys for new forms, or for copies of a form.

1) Access GKEY in GCDS, previously locking GKEY.LOCK.

2) From the global counter contained, determine the requested allotment of global keys.

3) Increment correspondingly the global counter.

4) Unlock GKEY.LOCK.

5) Build a reply to OFS-SDF, with this allotment of keys, and put it in DA.i.

6) END: Send the reply, in DA.i, to OFS-SDF in the Satellite.i, through PNI. Erase DA.i. Then wait for the next requested activity.

(This END step is common to all the following request types).

B) DR.i is a request of the trace of movements of a form among the satellites (or only the actual location of a form).

1) Access LOG in GCDS, previously locking LOG.LOCK.

2) Get all the tuples recording movements of that forms among satellites or through their mail trays (or get only the last recorded location of that form).

3) Unlock LOG.LOCK.

4) Build a reply to OFS-SDF with these traces, and put it into DA.i.

5) END

C) DR.i is a "mail" request, that is, the Satellite.i wants to receive all forms, or only a selected one, sent to it by other satellites, fetching them from its mail tray in the host.

1) Test if GRW.LOCK is set. If it is set, go to step 10 (a GR is in execution on DBT).

2) Lock TRAY.LOCK.i.

3) If this is a "mail" request for all forms in the tray (total "mail" request), lock all DBT.LOCK.j corresponding to every form type present in TRAY.i.

If this a single form "mail" request, lock only DBT.LOCK.j of the type of form requested (previously check if that form is really in TRAY.i).

4) If this is a total "mail" request, try to fetch all the forms (from the Relation.j(DBT), whose corresponding DBT.LOCK.j were locked) pointed out in TRAY.i.

It is possible that some forms, indicated in TRAY.i, are not found in the Relation.j(DBT) of the corresponding type: in this case, those forms are located in DBT.REL, because of a GR in execution, and must not be involved in this "mail" operation.

If this is a "mail" request for one only form, try to fetch it from the Relation.j(DBT) corresponding to its type.

5) Put the forms fetched into DA.i (answer to this request) and build a trace record for each form movement (from logical DBt.i to the Satellite.i).

6) Lock LOG.LOCK.

7) Insert into LOG all trace records (tuples) for the performed form movements.

8) Unlock LOG.LOCK, all previously locked DBT.LOCK.j and TRAY.i. Unlock DBT-REL.LOCK, if previously locked.

9) END.

10) Access GRN in GCDS: if the number of the current GR is different (namely greater) from the number in LGRN (that is, OFS-HDF.i has not already performed the current GR, or is not involved in it), go to step 2.

11) Access GRTYPE in GCDS, and look at the form type involved in the current GR. It is s.

12) If it is a single form "mail" request, of type k different from s, go to step 2.

13) Lock DBT-REL.LOCK.

14) Access DBT.REL. If it is a single form "mail" request and the wanted form is in DBT.REL, fetch it and go to step 5.

15) At this point the "mail" request cannot be performed completely because there is still in execution the same GR that this module has already finished. We have two possible cases:

a) If this "mail" request does not allow an incomplete answer, at the cost of delaying it: unlock DBT-REL.LOCK and wait on GRW.LOCK until it is unlocked, then go to step 2.

b) If this "mail" request does allow an incomplete answer, for not delaying it:

- If it is a single form "mail" request, it is now impossible to be performed; so unlock DBT-REL.LOCK, build a negative answer in DA.i and go to step 9.

- If it is a total "mail" request, it can now be done partially, substituting DBT-REL to Relation.k(DBT), as shown in the following steps.

16) Lock TRAY.LOCK.i.

17) Lock DBT.LOCK.j for every form type present in TRAY.i, but with j different from k.

18) Tray to fetch all the forms, pointed out in TRAY.i, from Relation.j(DBT), whose corresponding DBT.LOCK.j were locked, and from DBT-REL, at the place of Relation.k(DBT).

It is possible that some forms, indicated in TRAY.i, cannot be fetched, since they are in Relation.k(DBT). That happens because these forms are to be processed by the actual GR, on behalf of the "master" OFS-HDF.1 (that is, the one which issued the GR).

19) Go to step 5.

D) DR.i is a "ship" request: that is the Satellite.i wants to send a form, of type k, to the Satellite.j, putting it in the mail tray, in the host, of that satellite.

1) Lock TRAY.LOCK.j.

2) Put a tuple in TRAY.j, indicating the form sent (key and type).

3) Test if GRW.LOCK is set. If it is set, go to step 11 (a GR is in execution on DBT).

4) Lock DBT.LOCK.k.

5) Put the form, contained in DR.i, in a new tuple in Relation.k(DBT).

6) Lock LOG.LOCK.

7) Insert into LOG a tuple with the trace of movement of that form (from Satellite.i to the mail tray of Satellite.j).

8) Unlock LOG.LOCK, either DBT.LOCK.k or DBT-REL.LOCK (according to which previously locked) and TRAY.j.

9) Build in DA.i a suitable acknowledgement as answer to Satellite.i.

10) END.

11) Access GRTYPE in GCDS and look at the form type involved in the current Global Request. It is s.

12) If  $s$  is different from  $k$ , go to step 4 (the form type in GR is different from the form type of the form to ship).

13) Access GRN in GCDS: if the number of the current GR is different (namely greater) from the number in LGRN (that is, OFS-HDF.i have not already performed the current GR), go to step 4.

14) Lock GSM.LOCK. Access GRSATMAP in GCDS and unlock GSM.LOCK: if the Satellite.j is not involved in the current GR, go to step 4.

15) In this case, the form to be shipped cannot be stored, at the moment, into the designed Relation.k(DBT) because of the current GR; so lock DBT-REL.LOCK.

16) Put the form, contained in DB.i, in a new tuple in DBT.REL.

17) Go to step 6.

E) DR.i is a Global Request (GQ or GU) from the Satellite.i, involving the form type  $k$ .

1) Try to lock GR.LOCK. If this lock operation does not succeed at once, go to waiting state for a while (sleep  $R$  seconds), *staying\_available* at signals, from others OFS-HDF in the host, for a possible request involving OFS-HDF.i, and hence Satellite.i, in the actual GR.

This is important in order to avoid a deadlock which could occur if OFS-HDF.i performs an usual lock on GR.LOCK at this point (in fact GR could not be released, if it requires the cooperation by OFS-HDF.i, now waiting to take possession of GR).

After the waiting period, or after having performed a GR, try again to lock GR.LOCK.

2) Increment by 1 the number of actual GR in GRN (in GCDS) and put in GRTYPE the type  $k$  of the forms involved.

3) Take the time of the beginning of the execution of GR and put it into an entry in DA.i.

4) Take the Global Request, in structured form as output of YACC in OFS-



SLF, from DR.i and put it into GR (in GCDS).

5) Send a message to the linked OFS-SDF (in Satellite.i), through PNI, to signal that the GR previously sent to the host is now in execution in the whole DDBS, and so can be performed also locally in that satellite, on Relation.k(Sat.i).

6) Create DBT-REL, now empty, in accord to the Relation.k definition.

7) Lock GRW.LOCK, to signal to the other OFS-HDF that now a GR is in execution on DBT and a special procedure might be requested for "shipping" forms.

8) Access SATMAP (in GCDS) and create the relation file GRSATMAP, now empty.

9) For each Satellite.j that is requested to cooperate in the GR, also implicitly (for example if all the Satellites in the DS are indicated by default) the following checks are to be made:

- Is the Satellite.j actually connected?

- Only if the GR is a GQ: is the query class of Satellite.j lower or equal to the query class of Satellite.i? (Note that a satellite can perform a GQ only on satellites with query class lower or equal).

- Only if the GR is a GU: is the update class of Satellite.j strictly lower than the update class of Satellite.i? (Note that a satellite can perform a GU only on satellites with update class strictly lower).

If all these checks are affirmative, the Satellite.j can be involved in the GR, so put a new tuple in GRSATMAP with the indication of this satellite.

If also only one of these checks is negative, the Satellite.j cannot be involved in the GR, so put a suitable acknowledgement in DA.i (answer to the requesting Satellite.i).

Put a tuple in GRSATMAP for the Satellite.i, with the indication that this is the one which sent the GR, that is, it is the "master" for this GR.

10) Send a synchronization signal to all OFS-HDF.j involved in the GR (that is, for every Satellite.j pointed out in GRSATMAP).

Now the various OFS-HDF.j can start their concurrent operations to build their GA.j for this GR (see the part: "OFS-HDF.i activity for a GR from another satellite", step 0).

11) Wait a synchronization signal from the last OFS-HDF.j, involved in this GR, which is ready to handle it (see the part: "OFS-HDF.i activity for a GR from another satellite", step 1).

12) Lock TRAY.LOCK.j, for each Satellite.j involved in the GR (pointed out in GRSATMAP).

13) Lock DBT.LOCK.k and DBT-REL.LOCK.

14) Tag every form in Relation.k(DBT) which is pointed out in whichever TRAY.j, corresponding to a Satellite.j involved in the GR (as indicated in GRSATMAP).

15) Perform the GR on the tagged tuples of the Relation.k(DBT) and put this local answer (for forms in mail trays involved) in TEMP temporary file.

16) Now in DBT-REL there are all the forms, of the type involved in the GR, sent to satellites involved in the GR, by satellites which have already performed, in their DBs, the GR on the same forms: so OFS-HDF.i, the "master" for this GR, does not perform this GR again on these forms. After that, the function of DBT-REL is finished.

Thus, put all tuples of DBT-REL into Relation.k(DBT), erase DBT-REL and unlock GRW.LOCK (no need anymore of a possible special procedure access to DBT, for parallel "shipping" form operations).

17) Unlock DBT-REL.LOCK, DBT.LOCK.k and all TRAY.LOCK.j previously locked.

18) Wait and receive in GA.i the answer to the GR performed locally on the Satellite-i, that is on Relation.k(Sat.i). For that, wait a synchronization signal from PNI, indicating a message arrived from the linked OFS-SDF.

19) Lock GSM.LOCK and access GRSATMAP.

20) If the "master" OFS-HDF.i is the last module in the host, among those involved in the GR, which has received the answer from the linked satellite, as indicated in GRSATMAP, go to step 22.

21) If other OFS-HDF.j have not finished yet, unlock GSM.LOCK and wait a synchronization signal from the last OFS-HDF.j finishing the activity involved in this GR (see the part: "OFS-HDF.i activity for a GR from another satellite", step 5).

22) Now all the data related to the GR are owned by OFS-HDF.i. Take the time of the end of distributed execution of GR (to determine the consistency point interval - Par. 1.1 and Par. 1.5.2) and put it into an entry in DA.i.

23) Add TEMP (local answer from the mail trays) and GA.i (local answer from the Satellite.i) to GA, where the data for the Global Answer are not gathered. Erase TEMP and GA.i.

24) Perform other possible operation requested by GR on the items in GA (for example for mathematical queries, "unique\_count" queries, etc.).

25) Count the items in GA. They are R.

26) RMAX is the maximum number of items for the Global Answer, as indicated in GR (if not indicated, a default system value is assumed), in order to be accepted by the requesting Satellite.i.

If R is greater than RMAX, the (formatted) GA is spooled to an off-line printer, connected to the host.

If R is less or equal to RMAX, the (formatted) GA is added to DA.i.

Note that DA.i, also in the first case, contains the indication of the satellites requested to be involved to this GR but which were not be able to do that, and the indication of the time internal for the consistency point.

27) Erase in GCDS: GR, GA, DBT-REL, DBT-REL.LOCK (if present), GRATYPE, GRSATMAP.

28) END.

### 3.2.2 Algorithm BETA

In several parts, the Algorithm Beta is very similar to the Algorithm Alfa. The first main difference is in the implementation of mail trays in the logical host: here there is a local DB for each mail tray (DBt.i is the mail tray of Satellite.i).

The second main difference is in the way the Global Requests are performed on such mail trays. In Algorithm Alpha the "master" OFS-HDF.i, that is the one which started the current GR, performs the GR on all the trays, gathered in a single DB, that is DBT. Instead, in Algorithm Beta, each OFS-HDF.j, involved in the GR, performs this GR on its mail tray, that is on DBt.i.

We will list the resulting differences.

#### *Global data structures in the logical host*

Mail trays:

- DBt.i (with  $i=1, \dots, M$ ) = {Relation.j(DBt.i), with  $j=1, \dots, N$ }  
DBt.i contains the forms of the mail tray for the Satellite.i.
- DBt.i(Rel.j).LOCK (with  $i=1, \dots, M$  and  $j=1, \dots, N$ ): Lock files for each relation in each mail tray.

GCDS (Global Control Data Structures).

The only difference with the Algorithm Alpha is that DBT-REL, DBT-REL.LOCK and GRW.LOCK are not present here.

#### *Local data structures for each OFS-HDF.i*

They are the same ones like in Algorithm Alpha.

*OFS-HDF activity for a GR from another satellite*

NOTE:

As regards to the synchronization between different activities in OFS-HDF.i, we use, for the same reasons, the same policy as in Algorithm Alpha. That is, the Distributed Request from the associated satellite has the priority over the Global Request from another satellite, without allowing the preemption.

0) Like step 0 in Alpha (Satellite.i is involved in the current GR by OFS-HDF.j).

1-2) Like steps 1-2 in Alpha.

3) Access GRTYPE in GCDS, and look at the form type involved in the current GR. It is k.

4) Wait a synchronization signal from OFS-HDF.j (the "master" for this GR), if not already received. It indicates that each satellite involved has already started its activity for this GR (see the section: "OFS-HDF.i activity for a Distributed Request from the associated Satellite.i", part E, step 10). Now the GR can be performed locally on the mail tray DBt.i.

5) Lock DBt.i(Rel.k).LOCK and DBT-REL.LOCK.i.

6) Perform the GR on Relation.k(DBt.i), and put this local answer (for forms in this mail tray) in GA.i.

7) Now the function of DBT-REL.i (that is, holding all the forms, of the type involved in the GR, sent to Satellite.i by satellites which have already performed, in their DBs, this GR on the same forms) is finished: OFS-HDF.i did not perform this GR on the forms contained in DBT-REL.i, since other satellites have already done that.

Thus, put all the tuples of DBT-REL.i into Relation.k(DBt.i), and erase DBT-REL.i.

8) Unlock DBt.i(Rel.k).LOCK and DBT-REL.LOCK.i.

9-12) Like steps 3-6 in Alpha.

13) END.

*OFS-HDF.i activity for a Distributed Request  
from the associated Satellite.i*

0) This common step is like the common step 0 in Alpha

A-B) The parts A and B are like the parts A and B in Alpha.

C) DR.i is a "mail" request.

1) If this is a total "mail" request (for all the forms in the mail tray), lock all DBt.i(ReI.j).LOCK, for j=1, ... , N.

If this is a single "mail" request (for a particular form in the tray), lock only DBt.i(ReI.k).LOCK, where Relation.k is the type of the form.

2) If this is a total "mail" request, fetch all forms from all Relation.j(DBt.i), with j=1, ... , N.

If this is a single "mail" request, fetch the requested form Relation.k(DBt.i), corresponding to its type.

3-5) Like steps 5-7 in Alpha.

6) Unlock LOG.LOCK and all previously locked DBt.i(ReI.k).LOCK.

7) END.

D) DR.i is a "ship" request of a form, of type k, to the Satellite.j.

1) Lock DBt.j(ReI.k).LOCK.

2) Put the form, contained in DR.i, in a new tuple in Relation.k(DBt.j).

3) Lock LOG.LOCK.

4) Insert into LOG a tuple with the trace of the movement of that form (from Satellite.i to the mail tray of Satellite.j).

5) Unlock LOG.LOCK and DBt.j(Rel.k).LOCK.

6) Build in DA.i a suitable acknowledgement as answer to Satellite.i.

7) END.

E) DR.i is a Global Request (GQ or GU) from the Satellite.i, involving the form type k.

1-5) Like steps 1-5 in Alpha.

6-9) Like steps 8-11 in Alpha.

10) Access GRSATMAP, previously locking GSM.LOCK. Send a synchronization signal to all OFS-HDF.j involved in the GR, as indicated in GRSATMAP. Now the various OFS-HDF.j can start the GR on their mail trays, namely on Relation.k(DBt.j) (see the part: "OFS-HDF.i activity for a GR from another satellite", step 4).

11) Lock DBt.i(Rel.k).LOCK.

12) Perform the GR on the Relation.k(DBt.i) and put this local answer (for forms in this mail tray), into GA.i.

13) Unlock DBt.i(Rel.k).LOCK.

14-18) Like steps 18-22 in Alpha.

19) Add GA.i (local answer from the mail tray DBt.i and from the Satellite.i) to GA, where all the data for the Global Answer are now



gathered. Erase GA.i.

20-23) Like steps 24-27 in Alpha.

24) END.

## Table of Contents

1 Discussion of Global Requests problem in OFS. ....	2
1.1 Global Query requirements in the general case. ....	2
1.2 Office Form System environment. ....	5
1.3 DDB special features in OFS environment. ....	6
1.4 GR special features in OFS environment. ....	10
1.5 Requirements for the algorithm implementing GR. ....	13
1.5.1 Correctness requirement ....	14
1.5.2 Behaviour requirement ....	16
2 Syntax and semantic of Global Requests in OFS. ....	18
2.1 Syntax of Global Request. ....	18
2.2 Semantic of Global Request. ....	24
2.2.1 Global Query semantic. ....	25
2.2.2 Global Update semantic. ....	28
3 Design and implementation of OFS DDB System. ....	30
3.1 Logical and physical models. ....	30
3.1.1 Logical model. ....	30
3.1.2 Physical model. ....	35
3.1.3 Implementation choices. ....	38
3.2 Algorithms implementing Global Requests. ....	41
3.2.1 Algorithm ALPHA ....	46
3.2.2 Algorithm BETA ....	59

## REFERENCES

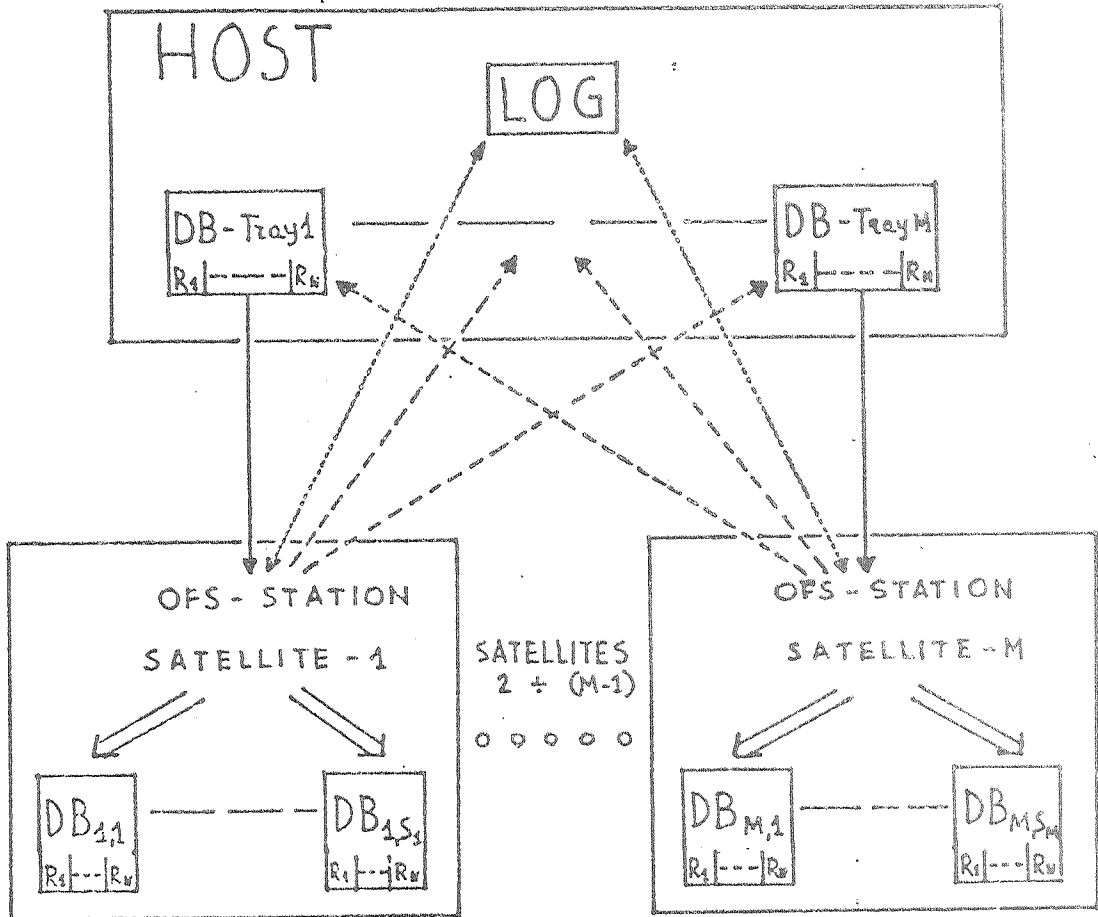
- Ref.1) M. Stonebraker: "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES". Proc. 3rd Berkeley Workshop on Distributed Data Bases and Computer Networks, pp 235-258. 1978.
- Ref.2) P. Bernstein, D. Shipman, J. Rotnie, N. Goodman: "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the General Case)". Technical Report CCA-77-09 (Computer Corporation of America). December 15, 1977.
- Ref.3) A) R. Hudyma, J. Kornatowski, I. Ladd: "Implementing a Microcomputer Database Management System (MRS)". Technical Report CSRG-101 (University of Toronto). May, 1979.
- B) J. Kornatowski: "The MRS User's Manual". CSRG - University of Toronto. July, 1979.
- C) I. Ladd: "A Distributed Database Management System Based on Microcomputers". M.Sc. Thesis, Dept. of Computer Science (University of Toronto). Jan., 1979.
- Ref.4) A) D. Tsichritzis: "A Form Manipulation System". Technical Report CSRG-101 (University of Toronto). May, 1979.
- B) S. Gibbs: "OFS: An Office Form System for a Network Architecture". M.Sc. Thesis, Dept. of Computer Science (University of Toronto). Sept., 1979.
- C) C. Cheung, J. Kornatowski: "The OFS User's Manual". CSRG - University of Toronto. March, 1980.
- D) S. Gibbs: "OFS Implementation Manual". CSRG - University of Toronto. Febr., 1980.
- Ref.5) O. Nierstrasz: "Automatic Procedures in OFS: the Sketches". (Working paper)

CSRG - University of Toronto. May, 1980.

- Ref.6) D. Tschritzis: "OFS: An Integrated Form Management System". Technical Report CSRG-111 (University of Toronto). April, 1980.
- Ref.7) M. Zloof: "Query By Example: a Data Base Language". IBM Systems Journal - Vol. 16, No. 4. 1977.
- Ref.8) S. Johnson: "YACC - Yet Another Compiler Compiler". Comp. Sci. Tech. Rep. No. 32, Bell Telephone Laboratories. July, 1975.
- Ref.9) Digital Equipment Corporation: "PDP-11/03, /23, /45, /50 Processor Handbooks".
- Ref.10) D. Ritchie, K. Thompson: "The UNIX Time-Sharing System" and "UNIX Implementation". The Bell System Technical Journal - Vol. 57, No. 6, Part 2. Aug., 1978.
- Ref.11) H. Lyckloma, D. Bayer: "UNIX on a Microprocessor". The Bell System Technical Journal - Vol. 57, No. 7, Part 2. Aug., 1978.
- Ref.12) B. Kernighan, D. Ritchie: "The C Programming Language". Bell Telephone Laboratories - Prentice-Hall Software Series. 1978
- Ref.13) A) R. Hudyma: "The Hardware Design of Distributed Office Workstations". Technical Report CSRG-111 (University of Toronto). April, 1980.
- B) Computer Technology: "Engineering Specification HEX-L11 - Interprocessor Linking System". Technical Manual. Nov. 21, 1979.

Ref.14) F. Rabitti: "Ricognizione di Sistemi Distribuiti in una Classificazione Generale di Sistemi di Elaborazione". Nota Interna B79-29 Istituto di Elaborazione della Informazione, Pisa - Consiglio Nazionale delle Ricerche. Dicembre 1979.

Fig.1






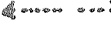
-  : DBMS access (query, update, insert, delete, etc...)
-  : Receive forms (mail)
-  : Send a form (ship)
-  : Record form movement through trays (in the host), and retrieve such 'traces' upon request

Fig. 2

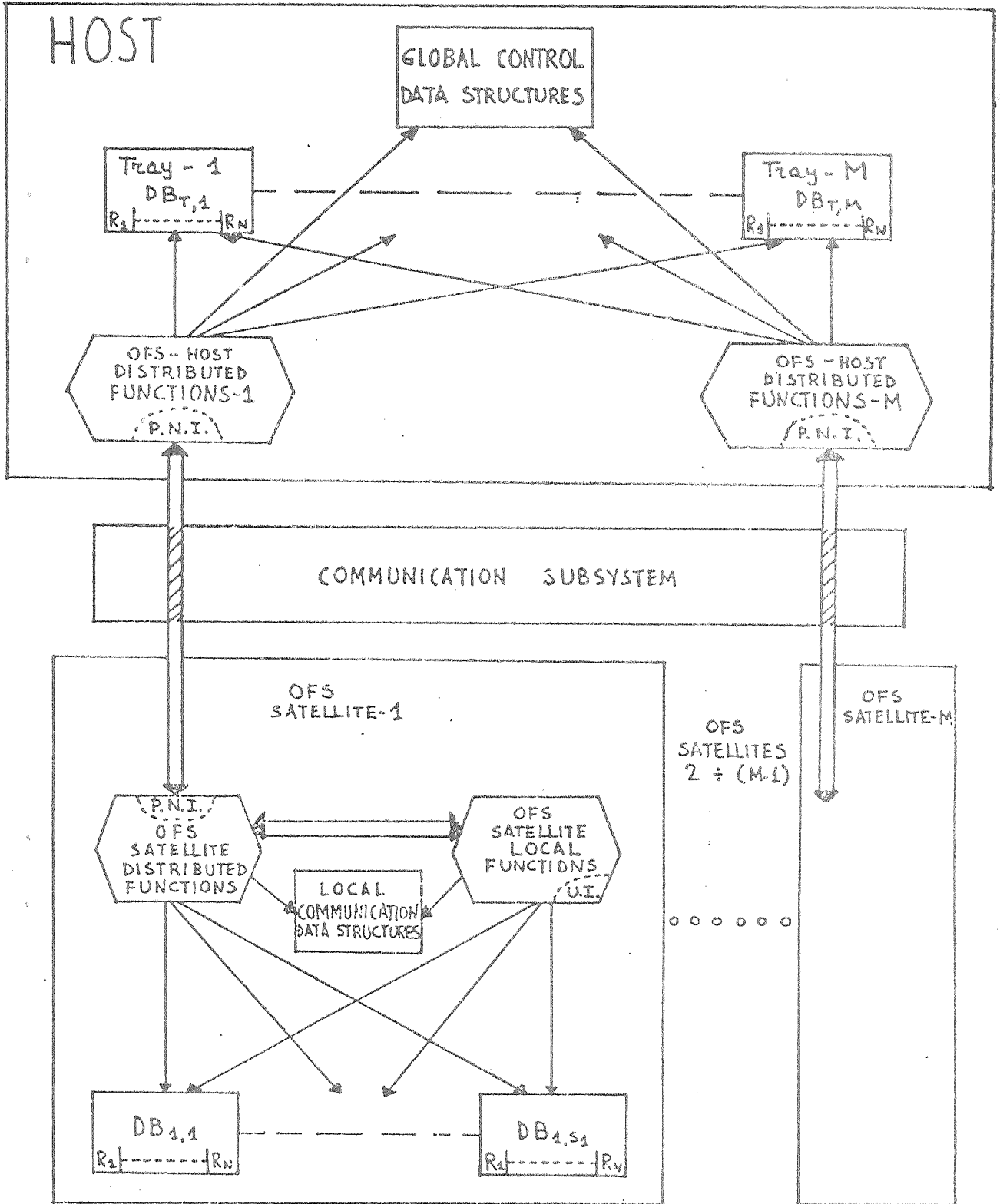


Fig.3

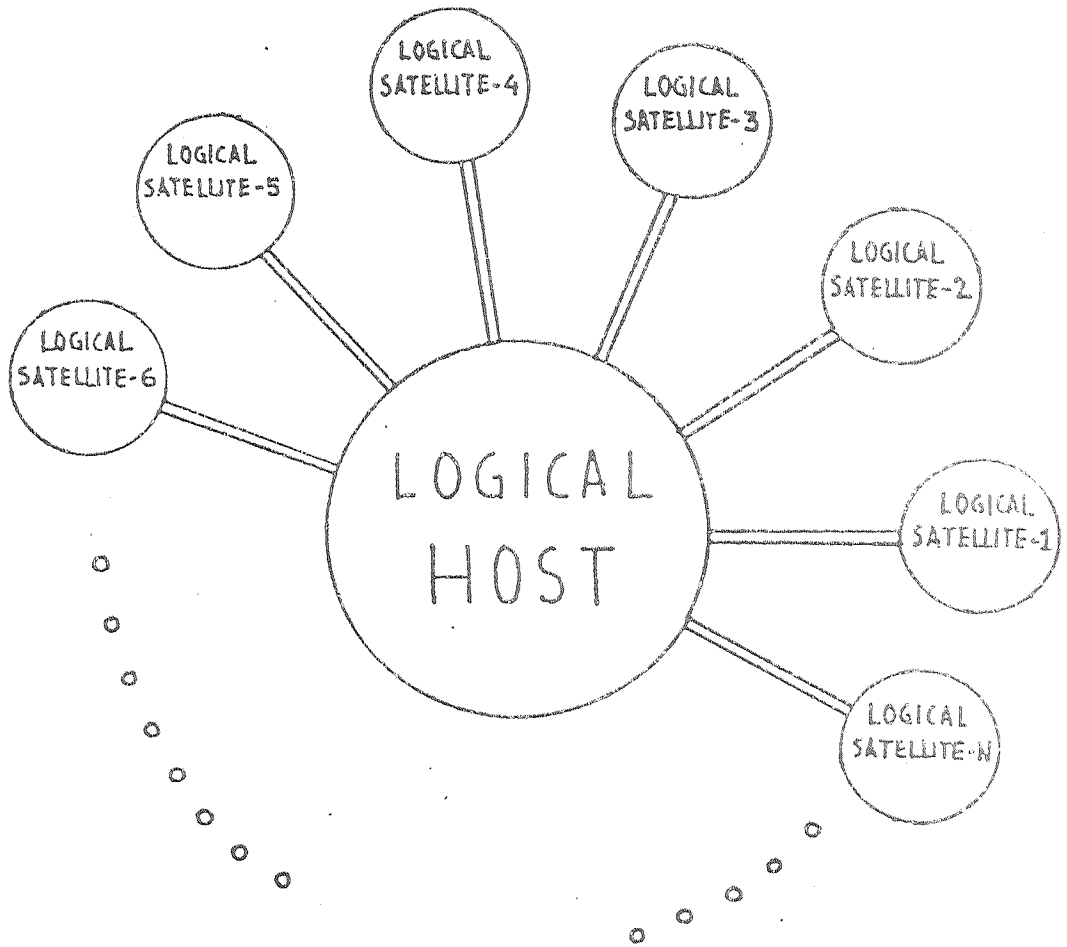




Fig.4

