# DISTRIBUTED PROCESSING OF DATA DYNAMICS

BY

COLETTE ROLLAND , CHRISTIAN RICHARD

UNIVERSITE PARIS I PANTHEON-SORBONNE
12, PLACE DU PANTHEON

75231 - PARIS CEDEX 5 - FRANCE

ABSTRACT

*We are concerned by distributed data base (DDB) consistency. We claim that a better DDB consistency requires to integrate in Distributed Base Management Systems a system of automata to manage its dynamics processing according to the description of data dynamics. We present a specification language in order to allow the dynamics DDB description according to the data dynamics modeling presented at the 4th VLDB Conference. We develop the system of automata able to perform in a distributed way the DDB dynamics processing and discuss the solution from concurrency and consistency points of view.*

## I - INTRODUCTION : How to increase distributed data base consistency

We claim that a better distributed data base (DDB) consistency requires to integrate in Distributed Data Base Management Systems (DDBMS) a system of automata to manage its dynamics processing according to the description of data dynamics.

The DDB consistency is reached to day by solutions to concurrency and integrity problems. These solutions are incomplete and very expensive in distributed environment. Furthermore , inconsistency in DDB results essentially in an incomplete management of data evolution as the user transactions are running on the distributed data base.

Our proposition consists in integrating consistency control in DDB evolution management function carried out by the DDBMS. To fulfil this function, the DDBMS manages DDB evolution according to the description schema of the DDB dynamics structure. It disposes of an inventory of updating operations types and possible events types that allows it to control both every event that occurs in the DDB and the resulting operations execution.

### I-1 DDBMS UPDATING MANAGEMENT POLICY

DDB updating management is currently worked out by transactions, under the following hypothesis :
1) DDB evolution is due to transactions executions : the transaction is the DDB transformation unit the DDBMS takes into account.
2) Users can define and trigger transactions execution at every moment : the transactions are known by the DDBMS only at their execution time.
These hypothesis determine a DDB evolution management policy grounded on transactions execution control covering both concurrency control and integrity control.



I-11 Concurrency control methods synchronize the execution of a set of transactions generated by users located in distinct sites but operating on a common set of data. Concurrency control methods are only justified in a distributed environment. We consider two classes of concurrency control methods :

- methods based on resources allocation and locking mechanims. They are derived from solutions used in operating systems field for generalized deadlocks problems [ 6 ], [ 7 ], [ 24 ].

- methods based on concurrent transactions temporal serialization techniques [ 22 ], [ 25 ], [ 4 ]. They occasion transactions execution according to a defined serialization order.

I-12 Integrity control methods aim to maintain semantic consistency. They must operate in distributed or not distributed data base context. We recognize two classes of integrity control methods :

- methods leading to the execution of a set of integrity constraints after every transaction execution, with if necessary a backtracking to the data initial state [ 3 ],

- methods grounded on consistency tests inserted in transactions. These tests are evaluated at the running transaction time and could stop it if necessary. These tests are defined through a semantic analysis of the transaction and are closely related to format programs proofs [ 26 ], [ 10 ]. They are either inserted by hand [ 6 ], [ 11 ], or automatically generated when it is possible [ 10 ].

I-13 Conclusive evaluations

a) Integrity controls and concurrency controls are both necessary and complementary : concurrency control, indispensable in a distributed environment, is not sufficient; integrity control must be set on to achieve semantic consistency.

b) Experience has pointed out that integrity control techniques through integrity constraints evaluation are rather expensive and little efficient. Techniques issued from formal programs proofs seem difficult to work up according to the opinion of some of their authors [ 8 ]. Consequently users cannot be guaranteed that the controls carried out by the DDBMS are efficient and sufficient.

c) Solutions currently retained to day, lead to define in a dynamic way a central control policy, as the DDB is operated by the running transactions. These solutions are not always a judicious ones. One can particularly remarks that :

- a transaction is analyzed as many times as its execution is required. Probably an analysis in terms of types should be sufficient.

- the DDBMS has to manage every transaction in a global way, even if it is a priori known that the transaction acts only upon local data. In a distributed environment up dating processing and integrity and concurrency controls should be, when convenient, distributed as data are.

### I-2 OUR DDB EVOLUTION MANAGEMENT POLICY

As a consequence of these evaluations we propose for concurrency and integrity control to ground DDBMS action on the following objectives :
1) The DDBMS has to know DDB evolution rules. In other words, it has to know data dynamics as well as data structure.
2) Consistency control is worked out through the control of DDB evolution guidance in the course of time.
The DDB description we propose is the modelling expression of both data and their dynamics through the concepts of a model presented in the fourth international conference on VLDB [ 17 ]. It issues in a dynamic conceptual schema which is an integrated representation of objects classes, operations classes and events classes. It comprises simultaneously a static sub-schema equivalent to the data schema and a dynamic sub-schema representing interconnections among data, operations and events. The description obtained is relational type.



The DDB evelution management is worked out in a distributed way by a distributed set of interdependent automata that we name Distributed Dynamics Processors (DDP). Each automata is responsible for a site DDB part. It is able to take into account a subset of events that may occurs in this site and to process them according to its local control schemata. The local control schemata is a projection of the DDB global dynamic schema refering to the data of the site. Each DDP is structurally synchronized with others processors. The structural synchronization operates when an event the DDP locally processes depends on events occurring on others sites and that must be synchronized with it. Every automaton is able to recognize every event related to the DDB and to transfer its control to the relevant automaton. The whole automata system makes the DDB evolve in a consistent way. Its manages automatically in a distributed way the DDB evolution in the course of time by a dynamic control that is, as needed, or stictly local or inter local.

Section 2 presents the DDB dynamics description. Section 3 presents the distributed dynamics processors (DDP) system and compare the proposed solution to existing ones.

## II - THE DDB DYNAMIC DESCRIPTION

The DDB dynamic description is the expression through a relational type language named ISDEL of the DDB dynamic conceptual schema, of integrity constraints that complete this schema and of the rules that define the projections of the DDB global view on local views. We recall the dynamic model already presented in the fourth international conference on VLDB and present ISDEL.

### II-1 DDB DYNAMIC CONCEPTUAL SCHEMA

The DDB dynamic conceptual schema defines a global view of the DDB described through an unique schema that simultaneously models data, their structure and their dynamics. It represents the distributed enterprise dynamics as the set of facts that trigger the real objects transformations and their connections.

## II-11 Conceptual model

The DDB dynamic conceptual schema is defined with three concepts of a causal type model.

. The c-object concept represents a time consistent aspect of a real world objects class or an association of objects classes. It defines an atomic state of the DDB. It is the representation of the larger set of properties of a class having an identical dynamic behaviour.

. The c-operation concept represents a real world operations class. It is defined by reference to the c-object concept. It is the representation of an elementary transformation that can happen to a c-object : the occurrences of a c-operation represent real operations that modify one or several objects of a same c-object.

. The c-event concept represents a real world events class. It is defined by reference to the c-object and the c-operation concepts. It is the representation of an elementary noteworthy state-change that could affect a c-object which triggers one or more operations : the occurrences of a c-event are events that ascertain a precise type of objects state change corresponding to a unique c-object and that trigger operations corresponding to one or several c-operations. Several state change types of a c-object can be c-events. The occurrences of several c-events can trigger operations of a unique c-operation. The triggering can be iterative and conditional

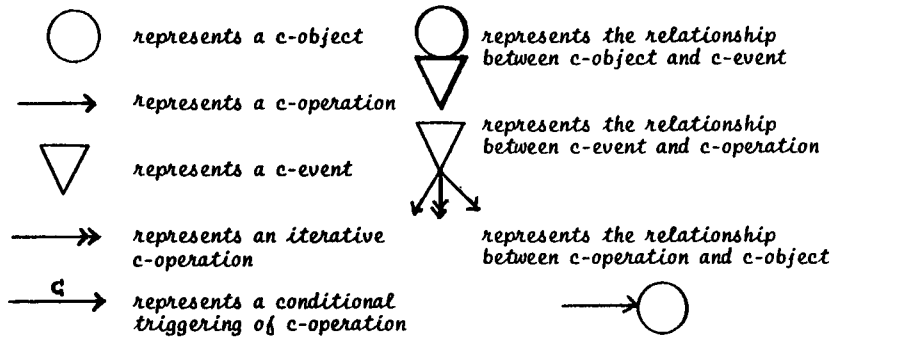
## II-12 The conceptual schema

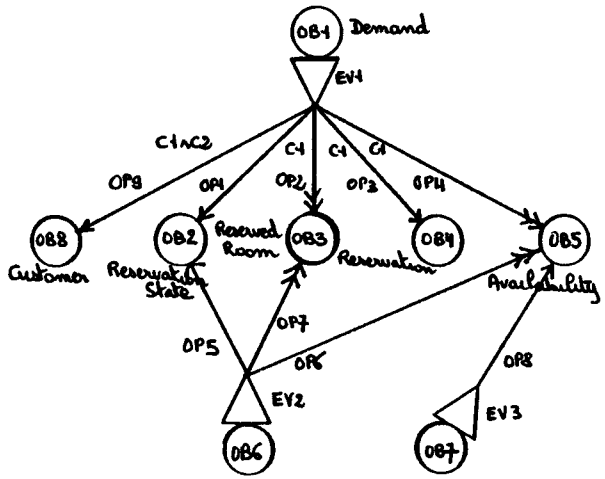
It is the integration of :

- the data schema corresponding to the conceptual data structure (the collection of c-objets).
- the dynamics schema corresponding to the conceptual dynamics structure of the DDB (the collection of c-operations and c-events). It represents the distributed enterprise dynamics in terms of causal associations among c-objects, c-operations and c-events. It defines all the events types that can cause the DDB evolution and all the corresponding updating operations types independently of both the operations physical distribution and the events ascertainment devices distribution. It describes in fact the synchronization structure of the DDB updating operations.

## II-13 Example

This example concerns a reservation system of hotel rooms in ski resorts. Figure 1 gives the collection of the DDB c-objects and a graphic representation of the corresponding dynamic schema based on the following conventions :




DEMAND (ndem, demname, demaddress, dnresort, dhotcat, dateb, datee roomsnb, dcatroom)

SKI-RESORT (nresort, resortname)

HOTEL (nhot, hotname, hotcat, nresort)

RESERVATION (nres, ndem, resbegdate, resenddate, nhot, roomsnber, ncust)

RESERVATION-STATE (ndem, demdate, state)

CANCELLATION-DEMAND (nres, canceldate, reason)
numres = nres, canceldate

RESERVED-ROOM (nroom, resbegdate, nres)
numresroom = nroom, resbegda

AVAILABILITY (nroom, avbegdate, avenddate
numavroom= nroom, avbegdate

ROOM (nroom, creationdate, catroom, nroom = nhot, nroom

CUSTOMER (ncust, custname, custaddress)

Figure 1 : An example of dynamic schema

| C-OBJECTS | C-OPERATIONS | C-EVENTS | CONDITIONS |
|---|---|---|---|
| OB1 : Demand | OP1 : Reservation state creation | EV1 : Demand arrival | C1 : The reservation demand can be satisfied |
| OB2 : Reservation state | OP2 : Reserved room creation | EV2 : Cancellation demand arrival | C2 : The customer does not exist in the DDB |
| OB3 : Reserved room | OP3 : Reservation creation | EV3 : Room modification arrival | |
| OB4 : Reservation | OP4 : Availability decreasing | | |
| OB5 : Availability | OP5 : Reservation state updating | | |
| OB6 : Cancellation demand | OP6 : Availability increasing | | |
| OB7 : Room | OP7 : Reserved room suppression | | |
| | OP8 : Availability increasing | | |

COMMENTS :

. The EV1 type event represents the arrival of a demand and corresponds to the creation of a new occurrence of OB1 in the DDB. EV1 triggers alternative operations. Either the demand refusal (OP1 setting the demand in a "refused "state"") if the condition C1 is false, or the booking corresponding to the creation of one occurrence of OB4 by OP3, creation of several occurrences of OB3 by OP2 and OB5 by OP4 and creation of one occurrence of OB8 by OP9 (condition C2 true), if the condition C1 is true.

. The EV2 type event represents the arrival of a cancellation demand and corresponds to the creation of a new occurrence of OB6 in the DDB. EV2 triggers operations that lead to the cancellation of a booking (suppression of the corresponding reserved rooms (OB3) by OP7, updating of the availability OB5 by OP6 and setting of the reservation in a "cancelled" state by OP5).

. Finaly, the arrival of a room modification is an event (EV3) that causes availability (OB5) updating (OP8)

## II-14 Conceptual schema description

The conceptual dynamic schema is described through a collection of relations completed by a set of integrity constraints. The time aspects integration in the model [ 21 ] leads to describe the three concepts by seven relation types (or "modes"). The COB mode describes the c-object concept. The COPPER and COPTEXT modes describe the c-operation concept and the CEVPER, CEVASCERT, CEVPRED, CEVTRIG modes describe the c-event concept. The dynamic conceptual schema is therefore a collection of relations belonging to one of these seven types. Otherwise, the dynamic conceptual schema must be completed by integrity constraints to make sure DDB consistency. Two kinds of integrity constraints are necessary [ 14 ]: static constraints that precise properties of the relations, as in current data base approach and dynamic constraints corresponding to the complete description of c-operations and c-events. The last describe namely the operations texts, the predicates defining events, the conditions and triggering factors. Moreover, notice that most of the integrity constraints associated to a data base are, as natural, yet expressed in the dynamic schema and consequently taken into account by the processors. So, a DDB dynamic description involves less integrity constraints than a data base schema.

## II-15 Projections of the DDB dynamic conceptual schema

The DDB global view described by the DDB dynamic conceptual schema is projected over several local views described as sub-schema of the DDB dynamic schema that we name local dynamic schema. Every local dynamic schema can be interpreted as :

- from a static point of view, it describes the collection of local data

- from a dynamic point of view, it describes the control structure of local data evolution, i.e the set updating operations locally managed through a program base, the set of events that can be recognized and locally processed and the set of causal links among events, operations and data.

## II-2 CONCEPTUAL LANGUAGE ISDEL

ISDEL must satisfy two main objectives :
1) to allow dynamic conceptual schema description, ie description of both relations collections and associated integrity constraints.

2) to be the central imput of the processors system.
To reach the first objective, we have chosen a relational complete language which is both the description and manipulation language of the DDB.
To reach the second objective, ISDEL includes constructions derived from programming languages, and namely PASCAL [28] and extensions close to those developed by Schmidt in PASCAL-R [23] and Wasserman in PLAIN [27]. With these constructions, we can define through assertional expressions the c-operations texts, the predicates, triggering conditions and factors associated to c-events. These expressions are interpreted and afterwards generated under the shape of processing modules stored in the programs base and workable by the processors.
We will limit our presentation to four basic constructions of ISDEL (assertional expressions, RELATION predefined type, FOREACH control structure and language extensions) and to the DDL description architecture in introducing in the example frame the most characteristic clauses.

### II-21 Four ISDEL basic constructions

#### II-211 Assertional expressions

ISDEL assertions are built under the general shape of current relational expressions. The usual logical operators are refered by the key words AND, OR, NOT, EXIST, ALL.

Examples :

a) provide all the reservations of the system :

(x) : reservation ⟨ x ⟩

b) provide all the reservation numbers corresponding to reservations issued since 01/01/81 for the "Squaw Valley" hotel :

(x.nres) : reservation ⟨x⟩ AND hotel ⟨y⟩ AND EXIST ⟨y⟩
(y-nhot = x-nhot AND y-hotname = "Squaw Valley" ) AND
(x-resbegdate = 01/01/81).

#### II-212 RELATION predefined type

This type has been introduced to allow local and temporary relations building from DDB relations and their manipulation without DDB modification. It has the same structure as a PASCAL RECORD. The declaration of a RELATION type variable obeys the following syntax (we systematically use 'BACCHUS' notation):

⟨relation type⟩ :: = RELATION ⟨field list⟩ END;
⟨field list⟩ :: = ⟨identifier⟩:⟨type⟩ |
⟨identifier⟩:⟨type⟩; ⟨field list⟩

Example : declaration of an intermediate relation : resort-reservation (nres, nhot, ncust).
VAR resort-reservation : RELATION nres : INTEGER ; nhot : INTEGER ; ncust : INTEGER END.

The t-uples set of a RELATION variable is built from the t-uples of existing relations.

Example : Build the "resort-reservation" relation from the t-uples of the "reservation" relation corresponding to the reservations issued for the period begining on 01/01/81 and ending on 01/02/81 for the hotels of the Squaw Valley ski-resort :

Resort-reservation : ⟨x-nres, x-nhot, x-ncust⟩ : reservation ⟨x⟩ AND ski-resort ⟨y⟩ AND hotel ⟨z⟩ AND EXIST⟨z⟩ (z-nhot = x-nhot) AND EXIST ⟨y⟩ (y.nresort = z.nresort AND z.nresort = "Squaw Valley") AND (x-resbegdate = 010181 AND resenddate = 010281).

#### II-213 FOREACH control structure

The FOREACH control structure has been introduced to express manipulation of t-uples of one relation. It is analogous to constructions of [14], [5], [16], and it obeys the following syntax :

FOREACH ⟨variable⟩ IN⟨relation⟩[UNTIL ⟨expression⟩] DO⟨instruction⟩ END;

This structure expresses the repetition of ⟨instruction⟩ either as many times as there are t-uples in ⟨relation⟩ or until the stop condition described by ⟨expression⟩ is fulfilled. For every repetition of ⟨instruction⟩ a new t-uple of the relation is allocated to ⟨variable⟩. We consider that ⟨variable⟩ has the same relation type as the relation refered by ⟨relation⟩ and consequently the same attributes. This is its only declaration and its range is the ⟨instruction⟩ group following the DO clause. ⟨Relation⟩ refers either a relation defined in ISDEL or a variable of RELATION predefined type.

Example : Compute the number of reservations issued in Squaw Valley in January 1981.

```
VAR number : INTEGER;
FOREACH reserv IN resort-reservation DO number : = number + 1  END;
```

### II-214 Language extensions

To make easier relations manipulation, ISDEL involves embodied usual functions [ 14 ],[ 5 ],[ 16 ]: TOTAL (adding up of an attribute values), COUNT (counting of the t-uples of a relation) AVERAGE (arithmetic average of a list of values), MAX (greatest value of an attribute), MIN (smallest value of an attribute), UP and DOWN (order functions).

## II-22 ISDEL description architecture

An ISDEL description of a dynamics conceptual schema is constituted of the successive and non ordered descriptions of the schema relations. The description of a relation is worked out through description clauses leading

1) to assign a type to every relation
2) to name it and to provide the list of all its attributes
3) to specify every attribute
4) to identify and to describe every integrity constraint :

```
<relation description> ::= IS-TYPE <is-type-ident> <relation-specification> END
                           expression of (1)
<relation specification> ::= <rel-spec> | <rel-spec> , <specification-relation>

<rel-spec> ::= <relation-name> | <field-name-list> | <domain-spec> | <relation-name>
                                                                     expression
             | <field-name-list> | <domain-spec>  <assertion-spec>
               of (2)              expression of(3) expression of (4)
```

We now review aspects (1), (3), (4) in introducing the main characteristic clauses.

### II-221 Relations types and modes

```
<is_type-ident> ::= COBJECT MODE COB | COPERATION MODE <cop> | CEVENT MODE <cev
<cop> ::= COPPER | COPTEXT
<cev> ::= CEVPER | CEVASCERT | CEVPRED | CEVTRIG
```

Every relation described through ISDEL is identified by an IS-TYPE, associating the relation with one of the three phenomena categories represented : C-OBJECT, C-OPERATION, C-EVENT. This type is completed by a MODE to specify the particulo aspect of the phenomenon described by the relation.

### II-222 Domains

```
<domain-spec> ::= DOMAIN <domain-spec-def> ;
<domain-spec-def> ::= <domain-def> | <domain-def> ; <domain-spec-def>
<domain-def> ::= <field-name> : <domain-exp> | <field-name> : SAME OF <relation r
<domain-exp> ::= CHAR (integer) | INTEGER (integer) | REAL (integer,integer) |
                 DATE (integer) | BOOLEAN
```

The DOMAIN clause specifies the domain of values associated to every relation attribute. In addition to usual predefined domains (INTEGER, REAL, BOOLEAN, CHAR), ISDEL allows DATE type use. This type is defined with regard to a calend [ 14 ]: a date is a projection over the calendar. Moreover, an attribute domain can be defined as that of another attribute (SAME OF).

### II-223 Assertional expressions of integrity constraints

Integrity constraints are introduced by the ASSERT clause :

```
<assertion-spec> ::= ASSERT <ident-spec-list>
<ident-spec-list> ::= <spec-list-elem> | <spec-list-elem> , <ident-spec-list>
<spec-list-elem> ::= identifier : <static assert> | identifier : <dynamic asse
                     identifier : LIKE identifier
```

### II-2231 Static integrity constraints

```
<static assert> ::= <field-name-list> COMPOSKEY <field-name>|<field-name>
                    KEY; | CONSTRAINT [<field-name-list>] :<text-constraint>;
```

KEY defines the relation key field; COMPOSKEY expresses the composition in fields of a key identifier; CONSTRAINT defines an integrity constraint over or between terms of attributes specified by text-constraint, whose result is transmited by the assertion identifier.

Example : Description of the C-object "Demand" (OB1)

```
IS-TYPE COBJECT MODE COB demand (ndem,demname,demaddress,dmresort  ,dhotcat,
                        dateb,datee,roomsnb, dcatroom)

DOMAIN  ndem : INTEGER (5); demname : CHAR (14);   dmresort : CHAR (10);
        demaddress : CHAR (60);dhotcat  : INTEGER (1); dateb : DATE (8);
        datee : DATE (8); roomsnb : INTEGER (2); dcatroom : INTEGER (2);

ASSERT  as1 : ndem KEY;
        as2 : CONSTRAINT (dateb,datee) :
              BEGIN IF (datee <dateb) THEN as2 := FALSE ELSE as2 :=TRUE END

END
```

### II-2232 Dynamic integrity constraints

```
<dynamic assert>  ::=<fieldname><id-text><dyn-assert-end> |<predicate>
<dyn-assert-end>  ::=(<relation name>) : <constraint-text>
<predicate>::= <initial-predicate>, <final predicate>|<fieldname>,
                <fieldname> NEW
<initial predicate>  ::= <field-name> INITPRED < dyn-assert-end>
<final predicate> ::= <field-name> FINPRED <dyn-assert-end>
<id-text> ::=  CONDITION | FACTOR | OPERATION
```

All the constraints are expressed under a same description schema. Text-constraint is an assertion block framed by BEGIN and END symbols. It corresponds to a procedure block whose imput parameter is a relation (relation of the schema or RELATION type variable) referred by <relation-name> and whose result (if existing) is transmited by the assertion identifier <field name>.

Key word CONDITION refers an integrity constraint describing a c-event triggering condition whose result (true or false) is transmited by the assertion identifier. Key word OPERATION refers a constraint describing a c-operation text. Key word FACTOR refers a constraint describing the set of the t-uples for which there will be an iterative operation triggering. This set is allocated to the corresponding <field name>. In this case, the assertion identifier takes the same part as a RELATION type variable. Key words INITPRED and FINPRED represent the description of a constraint defining the initial and final predicates associated to a c-event. The description is simplified when the c-event corresponds to the creation of a c-object occurrence (option NEW).

Examples : . 1) Description of OP2 triggering by EV1

```
IS-TYPE CEVENT MODE CEVTRIG trig-ev1-op2 (nev1, nop2, c1)
DOMAIN   nev1 : SAME OF  Demand arrival;
         nop2 : SAME OF  Reservation;
         c1   : CHAR (4);
ASSERT   as-ev1-op2 : c1 CONDITION (demand) : / Description of the triggering
condition
   BEGIN
      VAR avail-room : RELATION navailroom : INTEGER; navailhot : INTEGER;
                       ndate 1 : DATE; ndate 2 : DATE  END;
          avail-room :<(x-nroom, x-nhot,x-avbegdate,x-avendate) :
                availability <x> AND room <y> AND hotel <z> AND EXIST< z >
                (z-nresort = demand-dnresortAND z-hotcat = demand-dhotcat
                AND EXIST <y > (y-nhot = z-nhot AND y-catroom = demand-dcatroo
                AND EXIST <x > (x-nroom = y-nroom AND
                                x-avbegdate =<demand-dateb AND x-avenddate>
                                demande-datee)));
          IF COUNT ((x navailroom) :avail-room<x> )> = demand-roonsnb
             THEN c1 : = TRUE  ELSE  c1 : = FALSE;

   END
END
```

.2) Description of OP7 triggering by EV2

```
IS-TYPE CEVENT MODE CEVTRIG  trig-ev2-op7  (nev2, nop7, §5)
DOMAIN  nev2 : SAME OF  Cancellation-demand-arrival;
        nop7 : SAME OF Reserved-room-suppression;
          §5 : CHAR (2);

ASSERT  as-ev2-op7 : §5 FACTOR (Cancellation-demand) :
  / Definition of the triggering factor occurrences /
     BEGIN
      §5 :< (x nroom, x-resbegdate,x-nres) : reserved-room <x>
            AND EXIST< x > (x-nres = Cancellation-demand-nres),
     END;
END
```

.3) Description of OP7 operation

```
IS-TYPE COPERATION MODE COPTEXT Cancel (nop7, textop 7)
DOMAIN  nop7 : INTEGER (4);
        textop7 : CHAR (4);
ASSERT  as-op7 : textop7 (VAR : §actor : RELATION nroom§ : INTEGER (4);
                                    resbegdate§ : DATE (8); nres§ : DATE (8)
        BEGIN
            DELETE (§actor-nroom§, §actor-resbegdate§,
                  §actor-nres§) IN reserved-room; END
```

The DDB conceptual description is necessary to the Distributed Dynamics Processors System functioning that we will now present.

## III - DISTRIBUTED DYNAMICS PROCESSORS SYSTEM

The Distributed Dynamics Processors (DDP) system is composed of a set of interconnected automata that performs in a distributed way the DDB dynamics processing. The system works as an unique and centralized processor that would control the DDB evolution. We introduce the concepts of dynamics processing and DDB distributed dynamics processing. We present the DDP system architecture and the architecture and functions of a processor.

We demonstrate computational equivalence of the proposed solution and finally discuss our solution comparatively to the current ones for consistency keeping.

### III-1 DB DYNAMICS PROCESSING AND DDB DISTRIBUTED DYNAMICS PROCESSING PRINCIPLES

#### III-11 DB Dynamics Processing

DB dynamics processing consists in managing Data Base dynamics evolution control through an automaton named Dynamics Processor. The dynamics conceptual schema serves as a control schema for the automaton. As soon as the automaton perceives a state change (creation, modification or deletion of a c-objet relation t-uple), it takes it into account in verifying if this state change is an event, selecting and triggering the resulting operations and storing the induced state changes. We have yet demonstrated in the 5th Conference on VLDB [19 ] that this processor carries out updating operations synchronization and works out a consistent DB evolution. For this, the processor has to treat the event sequences in a consonant way with both the chronology of their appearance and the conceptual schema.

#### III-12 DDB Distributed Dynamics Processing

DDB distributed dynamics processing consists in bringing several processors into play to manage the evolution of a Distributed Data Base. This solution is better in terms of efficiency, costs, reliability and vulnerability. But the functioning of the whole set of processors must be equivalent to that of a single processor. When several processors share the control and execution of the events sequences occurring randomly over the different sites of the distributed system, each of them works according to its local dynamics schema that serves as a control schema. In such context, two events $ev_i$ and $ev_j$ treated by two distinct processors can be processed in a sequence different of that determined by a single processor working globally. This difference has no consequences if the two events are "independent", but if they are not, it can issue in inconsistencies. Therefore, we have to determine the sequence that always keeps the DDB consistent. To solve this problem, we have defined the concept of situational dependency allowing to discriminate in two classes the events of a given site. The events with a "local context" stamp do not depend on other sites. Conversely, the events with a

"global context" stamp must be synchronized with other events on other sites and related to the global times scale.

III-121 Situational dependency. Let EVi and EVj be c-events respectively related to local schema of sites $S_i$ and $S_j$. Let $O_i^{\bullet}$ the set of c-objects whose occurrences are created or modified by the operations triggered by EVi occurrences. Finally, let ${}^{\bullet}Oj$ be the set of c-objects whose occurrences are used by the operations triggered by EVj occurrences. Then $EV_j$ is situationaly dependent of $EV_i$ (denoted $EV_i \xrightarrow{S} EV_j$) if and only if ${}^{\bullet}O_j \wedge O_i^{\bullet} \neq \emptyset$. Therefore, an event $ev_j$ is situationaly dependent of an event $ev_i$ if $ev_i$ triggers at least one operation that modifies the objects used by at least one operation triggered by $ev_j$.

III-122 Situational dependencies graph. We can deduce from the DDB conceptual expression the whole set of situational dependencies among the DDB c-events. Let $t_n$ be the binary relation defined over the site $S_n$ as $t_n \subset E_n \times P_n$ in which $E_n$ (respectively $P_n$) refers the c-events (respectivly c-operations) set of site $S_n$ : $t_n\ (EV_i, OP_i^k)$ is true iff $EV_i$ triggers $OP_i^k$. Let $m_n$ be the binary relation defined over $S_n$ as $m_n \subset P_n \times O_n$ in which $O_n$ refers the c-objects set of site $S_n$ : $m_n\ (OP_i^k, OB_k)$ is true iff $OP_i^k$ modifies $OB_k$. Finally, let $u_n$ be the binary relation defined over $S_n$ as $u_n \subset P_n \times O_n$ : $u_n\ (OP_i^k, OB_k)$ is true iff $OP_i^k$ uses $OB_k$. Then, the situational dependency of $EVj \in Sj$ with regard to $EVi \in Si$ is equivalent to :

$(EVi \xrightarrow{S} EVj) \Longleftrightarrow (\exists OP_i^k, OP_j^{\ell}, OB_k)\ (t_i\ (EVi, OP_i^k) \wedge m_i\ (OP_i^k, OB_k) \wedge u_j\ (OP_j^k, OB_k) \wedge t_j\ (EVj, OP_j^k))$

or $\boxed{(\dagger)\quad (EVi \xrightarrow{S} EVj) \Longleftrightarrow (d_j^{-1} \circ u_j^{-1} \circ m_i \circ d_i)\ (EVi, EVj)}$ in which $\circ$ refers the relations composition operator. Then, the application of rule (†) allows the situational dependencies graph definition.

III-123 Global context stamped c-event - local context stamped c-event. A c-event EVj belonging to the local schema of a site Sj is a global context stamped c-event iff it is the target of at least one situational dependency. Conversely, it is a local context stamped c-event iff it is the target of no situational dependency. A local context stamped c-event depends only on the context of its site. It can be treated as soon as it occurs on the site independently of what happens on the other sites. Conversely, a global context stamped c-event must be treated according to the global chronology of events arrival over all the sites of the system. More precisely, it must be processed only when all the events on which it is situationally dependent and which are chronologically anterior have been processed.

III-124 Synchronization between processors. To achieve a consistent distributed functioning, the processors must be synchronized. The synchronization structure issues from the situational dependencies graph. As soon as one processor $P_i$ manages at least one event $EV_i^k$ source of situational dependency with $EV_j^m$ managed by $P_j$, it must dispose of mechanisms allowing execution locking / unlocking of every $EV_j^m$ by $P_j$. Notice that this approach allows a "structural" definition of the synchronization from the analyses of the DDB conceptual description.

III-125 Example.

The example presented in paragraph II-13 concerns a set of renting agencies working over a common set of resources (rooms and hotels) to manage ski resorts sojourns reservations. As all the agencies work in the same way, the conceptual schema of the figure 1 is both the global conceptual description of the system and the local dynamics schema of a site. An analysis of this conceptual description leads to the following situational dependencies between two sites i an j :

$EV1_i \xrightarrow{S} EV1_j$;
$EV1_j \xrightarrow{S} EV1_i$; $EV2_i \xrightarrow{S} EV1_j$; $EV2_j \xrightarrow{S} EV1_i$; $EV3_i \xrightarrow{S} EV1_j$;
$EV3_j \xrightarrow{S} EV1_i$.

Consequently, EV1 is global context stamped EV2, EV3 are local context stamped.

According to the distributed dynamics processing principles, we have to synchronise only global context stamped events. Then, as soon as an EV2 type event (cancellation demand arrival) occurs on a site, the DDP of this site has to lock the executions of every EV1 event occuring on the other sites all over EV2 processing. Else, the system may refuse a reservation demand because it has no available rooms while EV2 issues in the corresponding rooms release. Likewise, as soon as an EV3 type event occurs on a site, its DDP has to lock the execution of EV1 occuring on other sites all over EV3 processing because EV3 issues in modifications of availabilities that EV1 subsequently uses to accept or refuse a reservation demand. Last, as soon as an EV1 type event occurs on a site, its DDP has to lock the executions of every EV1 type event occuring on other sites. Else the process of two EV1 occurrences by two distinct DDP at the same time may issue in the reservation of the same room by two distinct persons for the same period.

### III-2 DISTRIBUTED DYNAMICS PROCESSORS SYSTEM ARCHITECTURE

The Distributed Dynamics Processors (DDP) system that works out DDB distributed dynamics processing is constituted of n interconnected DDP distributed over n sites (figure 2). Each DDP makes the local DDB evolve according to local dynamics schema rules. Therefore, each DDP is empowered to process a subset of the events occuring over the DDB and to select and execute through local DBMS the resulting updating operations stored in the local Programs Base (PB). DDP communicate on the one hand to transfer on the pertinent site a state change that cannot be processed on the site where it occurs and on the other hand to synchronize their functioning when it is necessary to global consistency achievement.

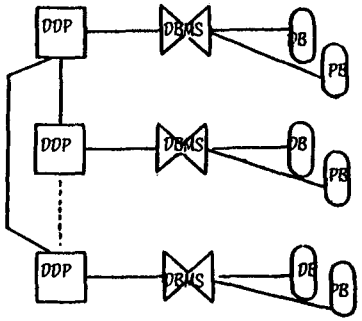



Figure 2

Notice that this architecture allows a complete heterogeneity of the local DBMS, Data Bases and Programs Bases.

### III-3 DISTRIBUTED DYNAMICS PROCESSOR ARCHITECTURE AND FUNCTIONS

The functional architecture of a DDP is a four layers architecture corresponding to the four functions worked by the DDP [13]

- The INTERFACE function takes into account DDB state changes introduced on its site and eventually transfers their control to another site.

- The EVALUATION function allows the evaluation of all the consequences of a state change by the DDP (is it an event? Which are the operations to trigger? Which are the updated c-objects?).

- The SYNCHRONIZATION function is specific of a distributed functioning of the processors. It works out synchronization between the processors so that for certain events their appearance order related to a unique global times scale might be reestablished.

- The EXECUTION function works out the "event-processing" (execution of the operations and DDB updating).

The DDP brings these functions into play in the order of this presentation, according to its own logic. It operates according to both its local dynamics schema and the situational dependencies graph, and acts over the local data base. In order to fulfil its task, it manages its own informations set that we name "references base" (figure 3).

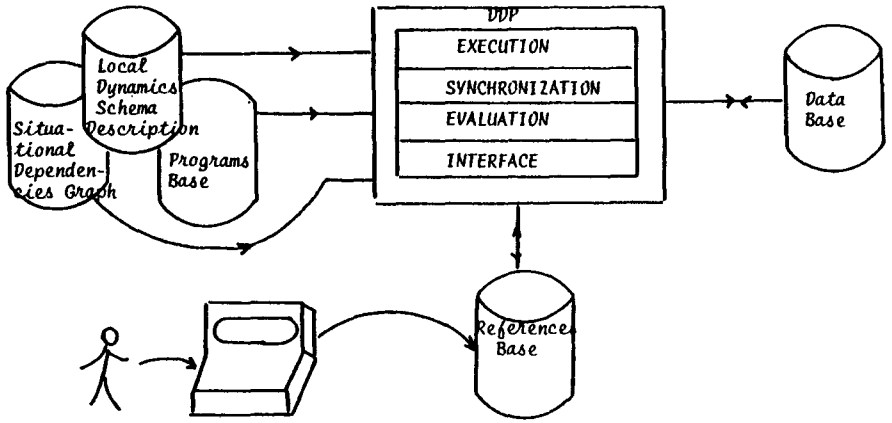



Figure 3

## III-4 DISTRIBUTED DYNAMICS PROCESSORS FUNCTIONING SPECIFICATION

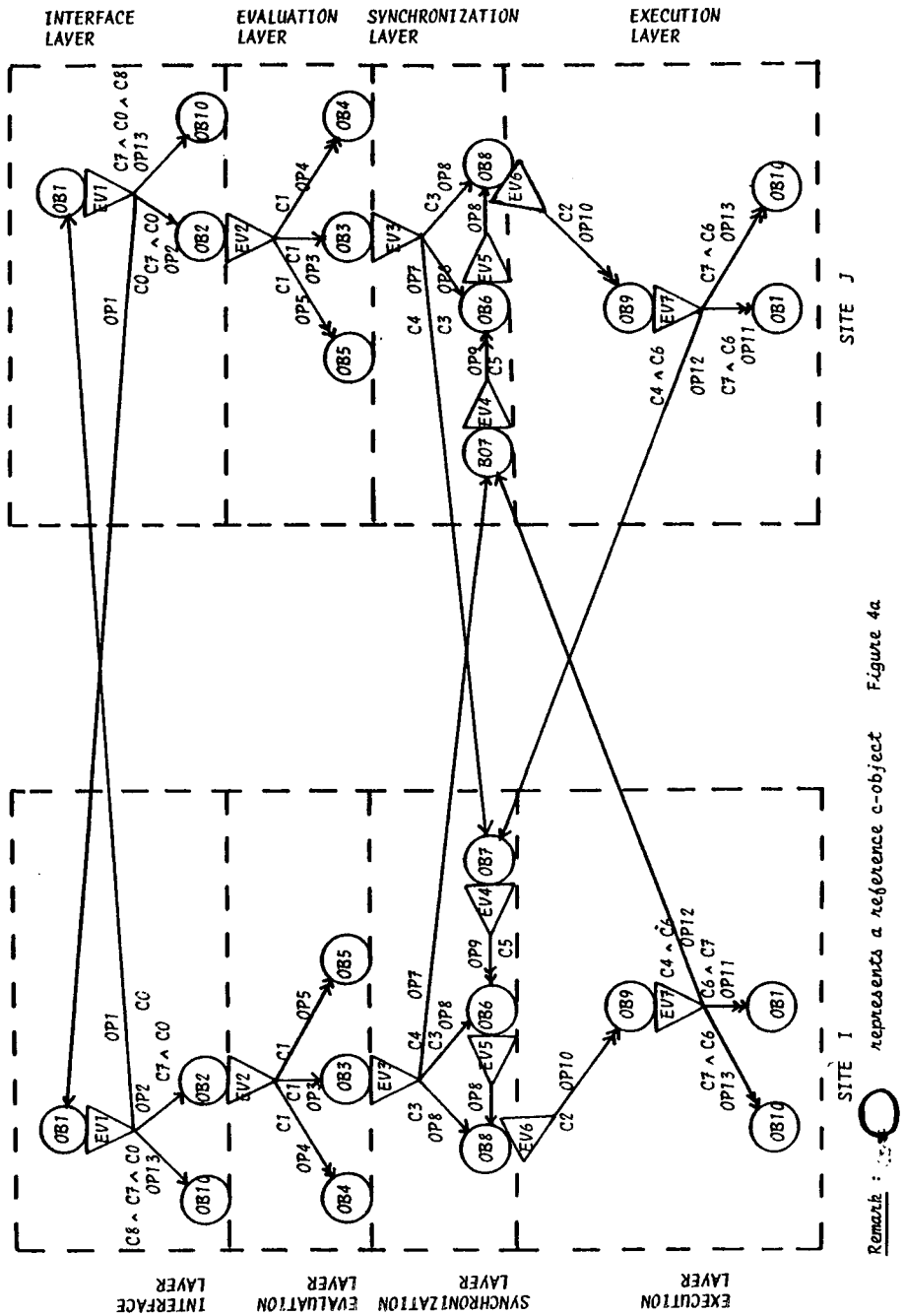
To describe the Distributed Dynamics Processor (DDP) consists in both defining the sequencing logic of the functions it brings into play and specifying every function.

### III-41 DDP functioning logic

We describe DDP functioning logic through the dynamics causal model. We present its conceptual schema through graphic notations in the case of two mutually synchronized DDP (figure 4). This schema points out the sequencing logic of the layers and the actions corresponding to each layer. The c-operations of this schema represent the actions performed by the DDP. Its c-objects are either the references used by the DDP or the DDB updated c-objects and its c-events describe the references state changes starting the DDP. For a better understanding of this schema, we present a relational description of the references :

- State change to take into account reference : OB1 (sc-ref, ob-ident, ob-ident-val, ob-attributes-val)
- State change locally taken into account reference : OB2 (lsc-ref, sc-ref, lob-ident)
- Produced c-event reference : OB3 (ev-ref, ev-ident, ev-ident-val, lsc-ref)
- C-operation to be triggered reference : OB4 (op-ref, op-ident, ev-ref, trig-cond-text, factor-text, op-text)
- C-object to be modified reference : OB5 (ob-ref, ob-ident, op-ref, ob-extension)
- C-event waiting for execution reference : OB6 (wev-ref, ev-ref, dep-occ-nb, gt-w-ev, w-ev-ident)
- Trace reference : OB7 (tr-ref, ext-ev-ident, ev-ident, gt-ext-ev, ext-ev-site, state)
- C-event to process reference : OB8 (pev-ref, ev-ref)
- Executed c-operation reference : OB9 (eop-ref, op-ref, op-ident-val, mod-ob-ident, mod-ob-att-val)

in which gt refers a global time defined either by clocks synchronization techniques [ 12 ] or without clocks synchronization by, for example, a circulating ticket over a virtual ring [ 15 ]. Then the DDP functioning conceptual schema is :




Remark : ○ represents a reference c-object Figure 4a

LEGEND

C-OBJECTS

OB1 : State change to take into account reference
OB2 : State change locally taken into account reference
OB3 : Produced c-event reference
OB4 : C-operation to be triggered reference
OB5 : C-object to be modified reference
OB6 : C-event waiting for execution reference
OB7 : Trace reference
OB8 : C-event to process reference
OB9 : Executed c-operation reference
OB10 : DDB c-object relation

C-OPERATIONS

OP1 : Transfer to the relevant site
OP2 : OB2 reference creation
OP3 : Event search
OP4 : Operation to be triggered search
OP5 : Object to be modified search
OP6 : Event processing delay
OP7 : Arrival of an event source of situational dependency signalling
OP8 : Event processing authorization making out
OP9 : Re-evaluation of event processing delay
OP10 : Operation execution
OP11 : OB1 reference creation
OP12 : End of the processing of an event source of dependency signalling
OP13 : DDB updating

C-EVENTS

EV1 : State change arrival
EV2 : State change to take into account locally arrival
EV3 : Event arrival
EV4 : External event processing trace arrival
EV5 : End of waiting
EV6 : Operation execution authorization
EV7 : An operation has been executed

CONDITIONS

C0 : This state change must be taken into account on this site
C1 : The state change is a c-event occurence
C2 : The triggering conditions are fulfilled
C3 : There are non processed and anterior occurrences of c-events sources of situational dependencies with the currently processed event
C4 : The processed event is a source of situational dependencies with this site
C5 : This external c-event occurrence is anterior to the waiting event
C6 : All the operations assouated to this c-event whose triggering conditions were fulfilled have been executed
C7 : The associated integrity constraints are fulfilled
C8 : External c-object



## III-42 Functions specification

The DDP functions correspond to the c-operations of the previous schema. We use a high level language to describe them simply.

First Layer INTERFACE function

Comment :

This function is activated as soon as a state change of a DDB c-object (described by reference OB1) must be analyzed by the DDP. This state change can be submitted by a local user (OB1 creation through Input/Output devices), transfered by another DDP (OB 1 creation by OP1) or produced by the local DDP (OB1 creation by OP11). The analysis consists either in transfering reference OB1 contents to the relevant site if the related c-object evolution is not managed on this site (OP1) or in both updating the c-object if the associated integrity constraints are fulfilled (OP13) and creating reference OB2 (OP2).

Specification :

```
BEGIN   FORONE  SC-ref  x1  DO
        BEGIN   Search-site (ob-ident, ob-ident-val, ob-site);
                IF ob-site ¬= site THEN BEGIN network-convey ((SC-ref x1),
                                                ob-site); Delete SC-ref x1));
                                    END
                               ELSE BEGIN creation := true; Search-integrity-
                                                constraints (ob-ident, constraint
                                                register, const-ident)
                               WHILE constraint-register ¬= NIL  DO
                               BEGIN bool := false; exec.constr-ident (bool);
                                  IF ¬bool THEN creation := false;
                                END;
                               IF creation THEN BEGIN create-object (x1) ;
                                                create-reference (lsc-ref,
                                                                  x1);
                                                END;
                               END;
        END;
END;
```

Second Layer : EVALUATION function

Comment :

This function works out the three basic primitives of dynamics processing : (1) to recognize if a c-object state change (described by OB2) is an event belonging to a c-event (operation OP3) (2) to select the c-operations to execute as the event consequences (operation OP4) and (3) to determine the c-objects whose states are modified by the c-operations execution (operation OP5). These three primitives use the local dynamics schema description and create the corresponding references : OB3 (produced c-event reference), OB4 (c-operation to be triggered reference) and OB5 (c-object to be modified reference).

Specification :

```
BEGIN FOR ONE lsc-ref x1 DO
    BEGIN  Search-event (lob-ident, event-register, ev-ident);
           WHILE event-register ¬ = NIL DO
                BEGIN Search-initial-predicate (ev-ident, init-pred-ident);
                      Pinit := false;
                      exec. init-pred-ident (Pinit);
                      IF Pinit THEN BEGIN Search-final-predicate (ev-ident,
                                          fin-pred-ident);
                                          pfin : = false;
                                          exec. fin-pred-ident (pfin);
                                          IF pfin THEN BEGIN event-register:=
                                                NIL;
                                                Create-reference (ev-ref,
                                                      ev-ident,ev-ident-val,
                                                      lsc-ref);
                                    END;
                         END;
                   Next (event-register);
           END;
        Search-operation (ev-ident, op-register, op-ident);
        WHILE op-register ¬ = NIL DO
           BEGIN Search-trig-cond (op-ident,condition);
                 Serach-trig-fact (op-ident,factor);
                 Search-op-text (op-ident,optext);
                 Create.reference (op-ref,op-ident,ev-ref,condition,
                                   factor,optext);
                 Next (op-register);
           END;
        FOREACH op-ref x2 OF ev-ref DO
           BEGIN  Search-object (op-ident, ob-ident);
                  Create.reference (ob-ref, ob-ident,op-ref,ob-extension);
           END;
    END;
END;
```

Third layer : SYNCHRONIZATION function

Comment :

This function works out events processing synchronization among DDP . It consists in :
1) making out the authorization to process an event $ev_i$ (described by OB3) if it belongs to a local context stamped c-event (condition C3) expressed through reference OB8 creation by OP8,
2) signalling $ev_i$ arrival to all the depending sites (those for which there is $ev_j$ such as $ev_i \xrightarrow{S} ev_j$) through trace references OB7 creation by OP7, if $ev_i$ is a source of situational dependencies,
3) locking $ev_i$ execution through a waiting mechanism (creation of a c-event waiting for execution reference OB6 by OP6) if $ev_i$ belongs to a global context stamped c-event (condition C3). The number of OB7 references corresponding to event $ev_i$ defines the number of events with which it must be synchronized.

$ev_i$ execution unlocking (EV5 arrival issuing OP8 execution) occurs as soon as all the relevant sites have signalled that the locking events processing has been terminated (OB7 updating by OP12).
Finally, let us notice that OP7 and OP12 requires the network-convey function.

Specification :

```
COBEGIN events-synchronization || signal-processing COEND;

PROCEDURE events-synchronization :
   BEGIN FORONE ev-ref x1 DO
         BEGIN Search-dep-source (ev-ident, source-register,source-ident);
               IF source-register ¬ = NIL THEN
                  BEGIN Exec. globalchronology (gt); state := 0;
                        WHILE source-register ¬ = NIL DO
                           BEGIN Search-site (source-ident, dep-site);
                                 Create-Trace (tr-ref,ev-ident,source-ident,
                                               gt,site,state);
                                 Network-convey (tr-ref, dep-site);
                                 Delete-Trace (tr-ref);
                                 Next (source-register);
                           END;
                  END;
               Search-situat-dep (ev-ident, sit-dep-register, sit-dep-ident);
               IF sit-dep-register = NIL THEN Create-reference (pev-ref, ev-ref)
                     ELSE BEGIN Exec.globalchronology (gt);
                                depnb := 0;
                                WHILE sit-dep-register ¬ = NIL DO
                                BEGIN Search-Trace (ev-ident, trace-register,
                                                    trace-ident);
                                      FORLAST trace-ident OF trace-register DO
                                           IF trace-ident ¬ = NIL THEN
                                              IF ((trace-ident.gtextev  gt ) AND
                                                 (trace-ident.state = 0)
                                                 THEN depnb := depnb + 1 ;
                                      Next (sit-dep-register);
                                END;
                                Create-reference (wev-ref, ev-ref, depnb, gt, ev-ident);
                          END;
         END;
END;

PROCEDURE signal-processing :
  BEGIN FORONE tr-ref x1 DO
     BEGIN IF state = 1 THEN
              BEGIN Search.Waiting.event (ext-ev-ident, waiting-register,
                                          waiting-ident);
                    WHILE waiting-register ¬ = NIL DO
                          BEGIN IF (waiting-ident. wev-gt< gt-ext-ev) THEN
                                BEGIN nb:= waiting-ident. dep-occ-nb - 1;
                                  IF nb = 0 THEN Create-reference (pev-ref,
                                                   waiting-ident.ev-ref)
                                            ELSE Create reference (wev-ref,
                                                   waiting-ident.ev-ref,nb,
                                                   waiting-ident.gt-w-ev, waiting-
                                                   ident.w-ev-ident);
                                END;
                          next (waiting-register)
                    END;
              END;
     END;
END;
```

Fourth Layer : EXECUTION function

Comment :

This function works out the event processing, i.e (1) the controlled execution (OP10) of the operations references by OB4, (2) the creation (OP11) if OB1 references to communicate the new state changes to take into account to the DDP.

Specification :

```
BEGIN FORONE pev-ref x1 DO
      BEGIN FOREACH op-ref x2 OF ev-ref DO
            BEGIN bool :=    false
                  Exec. op-ref.tri-cond-text (bool);
                  IF bool THEN BEGIN Exec.op-ref.factor-text (tuple -register,
                                                              tuple-ident);
                                     WHILE tuple-register ¬ = NIL DO
                                     BEGIN Exec.op-ref.op-test (tuple-ident,
                                           op-ident-val, mod-ob-ident,
                                           mod-ob-att-val);
                                        Create-reference (eop-ref,op-ref,
                                           op-ident,val,mod-ob-ident,
                                           mod-ob-att-val);
                                        Next (tuple-register);
                                     END;
                        END;
      END;
      FOREACH eop-ref x2 OF ev-ref DO
      BEGIN   creation : = true;
              Search-integrity-constraint (mod-ob-ident, constraint-register,
                                           const-ident)
              WHILE constraint-register ¬ = NIL DO
                    BEGIN bool := false;
                          Exec.const-ident (bool);
                          IF ¬bool THEN creation := false;
                    END;
              IF creation THEN BEGIN Create-object (mod-ob-ident,mod-ob-att-val);
                                     Create-reference (sc-ref,mod-ob-ident,
                                                       mod-ob-att-val);
                                     END;
      END;
      Search-dep-source (ev-ref.ev-ident,source-register,source-ident);
      IF source-register ¬ = NIL THEN
         BEGIN State : = 1; Exec.globalchronology (gt);
               WHILE source-register ¬ = NIL DO
                     BEGIN Search-Site (source-ident,dep-site);
                           Create.Trace (tr-ref,ev-ref.ev-ident,source-ident,
                                         ev-ref.val-ident-ev,gt,site);
                           Network.convey (tr-ref), dep-site);
                           Delete.Trace (tr-ref);
                           Next (source-register);
                     END;
         END;
   END;
END;
```



## III-5 SOLUTION CORRECTNESS DEMONSTRATION

We ground this demonstration on the serializability theorem presented by Bernstein and Goodman in [ 2 ].

### III-51 Some recalls about serialization conditions

An execution E of transactions $T_1,\ldots,T_n$ is a serial execution iff no transaction ever execute concurrently in E. Therefore, it preserves DDB consistency. An execution E is serializable iff it is computationally equivalent to a serial one, i.e. if it produces the same output and has the same effect on the DDB as some serial execution. Consequently, it constitutes a correct execution [ 2 ]. Let $\longrightarrow_{rw}$, $\longrightarrow_{wr}$, $\longrightarrow_{ww}$ be three binary relations defined over E as : $T_i \longrightarrow_{rw} T_j$ iff on a site, $T_i$ reads some data item into which $T_j$ subsequently writes; $T_i \longrightarrow_{wr} T_j$ iff on a site, $T_i$ writes into some date item that $T_j$ subsequently reads; $T_i \longrightarrow_{ww} T_j$ iff on a site, $T_i$ writes into some data item into which $T_j$ subsequently writes. Let $\longrightarrow$ be the binary relation defined as $\longrightarrow = \longrightarrow_{rw} \cup \longrightarrow_{wr} \cup \longrightarrow_{ww}$. Then E is serializable iff $\longrightarrow$ is a cyclic (Serializability theorem

### III-52 Solution correctness

We show how the situational dependency relation ($\xrightarrow{S}$) allows relation $\longrightarrow$ of [ 2 ] retrieval and we show that for every execution E this relation $\xrightarrow{S}$ is a cyclic

Let us first recall that relation $\xrightarrow{S}$ can be expressed as a composition of the binary relationst(trigger), m (modify) and u (use ) : $(EV_i \xrightarrow{S} EV_j) \Longleftrightarrow (\exists\, OP_i^k, OP_j^k, OB_k)\ (d_i(EV_i, OP_i^k) \wedge m_i(OP_i^k, OB_k) \wedge u_j(OP_j^k, OB_k) \wedge t_j(EV_j, OP_j^k))$.

SO, $(T_i \longrightarrow_{rw} T_j) \Longleftrightarrow (\exists\, OP_i^k, OP_j^k, OB_k)\ (u_i(OP_i^k, OB_k) \wedge m_j(OP_i^k, OB_k))$ : if $T_i \longrightarrow_w T_j$, there is an operation $OP_i^k$ belonging to $T_i$ that reads a data item (an object) $OB_k$ that $OP_j^k$ belonging to $T_j$ subsequently modifies. Let $EV_i$ (resp. $EV_j$) be the c-event that triggers $OP_i^k$ (resp.$OP_j^k$). Then $T_i \longrightarrow_w T_j$ $\Longleftrightarrow T_j\ (EV_j, OP_j^k) \wedge m_j(OP_j^k, OB_k) \wedge u_i(OP_i^k, OB_k) \wedge t_i(EV_i, OP_i^k) \Longleftrightarrow EV_j \xrightarrow{S} EV_i$.

Likewise, $T_i \longrightarrow_{wr} T_j \Longleftrightarrow (\exists\, OP_i^k, OP_j^k, OB_k)\ (m_i(OP_i^k, OB_k) \wedge u_j(OP_j^k, OB_k) \Longleftrightarrow T_i(EV_i, OP_i^k) \wedge m_i(OP_i^k, OB_k) \wedge u_j(OP_j^k, OB_k) \wedge t_j(EV_j, OP_j^k) \Longleftrightarrow EV_i \xrightarrow{S} EV_j$. Last $T_i \longrightarrow_{ww} T_j \Longleftrightarrow (\exists\, OP_i^k, OP_j^k, OB_k)\ (m_i(OP_i^k, OB_k) \wedge m_j(OP_j^k, OB_k) \wedge u_j(OP_k^k, OB_k))$ [because $T_j$ writes after $T_i$] $\Longleftrightarrow t_i(EV_i, OP_i^k) \wedge m_i(OP_i^k, OB_k) \wedge u_j(OP_j^k, OB_k)$

$\wedge t_j(EV_j, OP_j^k)\ [\wedge m_j(OP_j^k, OB_k)] \Leftrightarrow EV_i \xrightarrow{S} EV_j$.

Therefore as $\longrightarrow = \xrightarrow[rw]{} \cup \xrightarrow[wr]{} \cup \xrightarrow[ww]{} \xrightarrow{S}$ .

We xill now show that $\xrightarrow{S}$ is a cyclic for every execution E. Let us recall that a transaction $t_j$ can be viewed as a sequence of elementary operations : $T_j = OP_j^1 \ldots\ldots OP_j^{nj}$. As each one of these operations is triggered by an event $ev_j^k$, $T_j$ can be viewed as a sequence $\sigma_j$ of events $ev_j^k : \sigma_j = ev_j^1 \ldots\ldots ev_j^{kj}$ such as $ev_j^1 \xrightarrow{+} ev_j^2 \xrightarrow{+} \ldots \xrightarrow{+} ev_j^{kj}$ (in which $\xrightarrow{+}$ refers the chronological dependency relation [ 18 ]. Then an execution E of transactions $T_1, \ldots, T_n$ will be a set $\underline{\sigma}$ of n events sequences $\sigma_j$ corresponding to the transactions : $\underline{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$ in which $\sigma_j = ev_j^1 \ldots\ldots ev_j^{kj}$. Let us split up every sequence $\sigma_j$ in sub-sequences $\sigma_j^1, \ldots\ldots \sigma_j^{xj}$ such as 1) the first event of every sub-sequence is a global context stamped event and 2) the other events of the sequence are local context stamped events : $(\sigma_j^i = ev_j^{i1} \ldots\ldots ev_j^{iPij}) \wedge (\xrightarrow{S})^{-1}(ev_j^{11}) \neq \emptyset \wedge (\forall k \in [2, P_{ij}]) (\xrightarrow{S})^{-1}(ev_j^{ik}) = \emptyset)$. Then, the system $\underline{\sigma}$ becomes $\underline{\sigma}' = \{\sigma_1^1 \ldots \sigma_1^{x1}; \sigma_2^1 \ldots \sigma_2^{x2}; \ldots; \sigma_n^1 \ldots \sigma_n^{xn}\}$. As the local context stamped events have no influence on the serialization order, it is sufficient only to consider the initial events of every sub-sequence. Therefore, it is sufficient to analyze the situational dependencies graph defined over the set $E_\sigma = \{ev_1^{11}, ev_1^{21}, ev_1^{x_1 1}; ev_2^{11} ev_2^{21} \ldots ev_2^{x_2 1}; \ldots; ev_n^{11} ev_n^{21} \ldots ev_n^{x_n 1}\}$ that becomes, after suppression of events repetitions in several sub-sequences, $E'_\sigma = \{ev_\sigma^1, ev_\sigma^2, \ldots, ev_\sigma^n\}$. Then, execution E will be serializable iff the graph $\varphi(E'_\sigma, \xrightarrow{S})$ is acyclic. Let us first recall that the chronological dependencies graph $\varphi(E, \xrightarrow{+})$ is acyclic for any sequence of chronologically dependent c-events occurrences [ 18 ]. Now, $ev_i \xrightarrow{S} ev_j \Leftrightarrow t_i(ev_i, OP_i^k) \wedge m_i(OP_i^k, OB_k) \wedge u_j(OP_j^k, OB_k) \wedge t_j(ev_j, OP_j^k) \Rightarrow ev_i \xrightarrow{+} ev_j$ in the DDB global conceptual schema because by definition of $\xrightarrow{+}$, if an operation $OP_j^k$ uses an object created or modified by an operation $OP_i^k$, their triggering events are in chronological dependency. Then suppose the graph $\varphi(E'_\sigma, \xrightarrow{S})$ to be cyclic for a given execution E, i.e. for a given occurrence of $E'_\sigma$. Then $(\exists (ev_i^1, ev_i^2, \ldots, ev_i^k) \in E'^k_\sigma) (ev_i^1 \xrightarrow{S} ev_i^2 \xrightarrow{S} \ldots \xrightarrow{S} ev_i^k \xrightarrow[\sigma_S]{} ev_i^1) \Rightarrow ev_i^1 \xrightarrow{+} ev_i^2 \xrightarrow{+} \ldots \xrightarrow{+} ev_i^k \xrightarrow{+} ev_i^1$. Consequently, the chronological dependencies graph $\varphi(E'_\sigma, \xrightarrow{+})$ involves a cycle and that is inconsistent with the result previously recalled. Then for all the occurrences of $E'_\sigma$ representing an execution E, the graph $\varphi(E', \xrightarrow{S})$ is acyclic and that proves the proposed solution correctness.



## III-6 EVALUATION OF THE PROPOSED SOLUTION

The proposed solution is grounded on a "structural" definition of both consistency control and synchronization control. The dynamics conceptual schema defines all the events and operations that can occur over the DDB. Consequently, it defines all the consistency control the D-DBMS has to work out. The analysis of this schema distribution over the sites allows the definition of all the synchronization controls the D-DBMS has to performs. This solution requires a more important design work, but in our point of view, it simplifies and improves DDB operating.

A) The dynamics schema definition replaces the integrity constraints determination work. One can expect that the precise concepts we have developed make this work easier and more precise. Moreover, there are theorems to verify dynamics conceptual schema consistency and correctness. Without the help of such solutions, it seems very difficult to find out all the integrity constraints necessary to check DDB semantic consistency. Besides, consistency control will be more occurate and less expensive.

B) The situational dependency concept allows to strictly define all the conflicts that can occur during events processing and to build up the cooperation among DDP that perform it. These solutions are equivalent to concurrency control solutions worked out in current D-DBMS but they seem to present two main advantages :

1) Events stamping is defined over the types and used for all the occurrences of the type. If we refer to the example, every time an event belonging to EV1 type occurs on a site, the DDP knows that it must bring the mechanisms of the synchronization layer into play because EV1 is a global context-stamped c-event. In a current solution, for each transaction corresponding to a demand processing, the D-DBMS has to re-analyze the transaction and to re-define its processing strategy. Therefore, our solution increases D-DBMS efficiency and decreases synchronization cost.

2) Local events stamping leads to the same kind of advantages : it is defined once at a time and it allows the immediate processing of local events. Conversely, the current solutions lead to analyze all the transactions whatsoever they may be.

C) The proposed solution allows an effective distribution of both programs and evolution control issuing in efficiency improvment. Else, the D-DBMS has to perform at a global level all the queries evaluation functions to determine if all the operations embeded in a query are local operations. Otherwise, the execution control remains global even if the subqueries are processed locally by the local DBMS . Consequently, they result in efficiency decreasing. There, if we refer to the previous example, the D-DBMS has to analyze at a global level each transaction corresponding to a "cancellation demand" to recognize its local character and consequently trigger and control its local execution. Our solution permits a more complete decentralization of the data management and a greater independence of the processors towards a better adequacy of computer systems in regard with users needs [ 1 ].

D) The proposed solution involves a minimal number of lockings. The c-events and c-operations describe the elementary operations and events that can occur over the DDB. The synchronization due to updating operations logic and the synchronization related to distribution are defined at the sharpest level. In current approaches, the transactions are synchronized as a whole: n operations embodied in a transaction can be needlessly locked because only one of them is conflictual. Consequently, parallelism ratio decreases whilst locking number needlessly increases.



## IV - CONCLUSION

Starting from the purpose of DDB distributed dynamics management in D-DBMS, we have draun two major conclusions :

1) A complete description of data dynamics allows a more consistent DDB management. The DDB consistency is due to the Distributed Dynamics Processors (DDP) system that controls every state-change of the DDB objects and triggers and synchronizes all the transformations bearing on the objects and that synchronizes processes actions. So it minimizes the risks of either semantic inconsistency due to a bad transactions programming or erroneous transactions concurrency management.

2) To specify a software in two parts, a control part (the dynamics schema) and an operational part (the processor) is useful and efficient. In our case, DDP system efficiency in due to the distribution of both programs and control : every DDP works independently and sometimes cooperates with one or several other DDP, but it is not controlled by a global scheduler, and to the existence of a control schema defined in terms of types allowing to build up once at a time the concurrency control mechanisms (synchronization between processors) and the consistency control mechanisms (events management).

Finally, notice that this approch regains automata theory and discrete processors theory [ 9 ]. We have demonstrated in [ 20 ] the equivalence of our approch with this theory.

1 ASSOCIATION OF COMPUTING MACHINERY
"Proceedings of the Workshop on Data Abstraction, Data Bases and Conceptual Modelling"
Pingree Park, Colorado, June 1980, ACM editors.

2 BERNTEIN, P.A. - GOODMAN, N.
"Timestamp based algorithms for concurrency control in distributed data base systems"
Proceedings of 6th international Conference on VLDB, Montreal, Sept. 1980.

3 BADAL, D.Z. - POPEK, G.J.
"Costs and performance analysis of Semantic Integrity Validation Methods"
ACM/SIGMOD 79, Boston, 1979.

4 BERNSTEIN, P.A. - SHIPMAN, D.W. - ROTHNIE, J.B.
"Concurrency control in a System for Distributed Database (SDD-1)"
ACM - T.O.D.B.S., Vol.5, Nb.1, March 1980.

5 DATE, C.J.
"An introduction to Database Systems"
Addison Wesley editor, 1977.

6 ESWARAN, K.P. - GRAY, J.N. - LORIE, R.A. - TRAIGER, J.L.
"The notion of Consistency and Predicate Lock in a database system"
Comm. ACM, Vol.19, Nb.11, Nov. 1976.

7 ELLIS, C.A.
"A robust algorithm for updating duplicate databases"
Proceedings of 2nd Berkeley Workshop on Distributed Data management and Computers Network
May 1977.

8 GARDARIN, G.
"Integrity, Consistency, Concurrency, Reliability in Distributed Data Base ManagementSystem" in "Distributed Data Bases",
C. Delobel and W. Litwin editors, North-Holland, Publ. C°, April 1980.

9 GHISHKOV, V.M. - LETICHEVSKII, A.A.
"Theory of algorithms and discrete processors"
Advances in information systems science, Vol.1, Ch.1, 1973

10 GARDARIN, G. - MELKANOFF, M.
"Proving consistency of database transactions"
Proceedings of 5th international conference on VLDB, Rio de Janeiro, Oct. 1979.

11 HAMMER, MM. - SARIN, S.K.
"Efficient monitoring of database assertions"
ACM/SIGMOD 78, International Conference on Management of Data, Dallas, June 1978.

12 LAMPORT, L.
"Time, clocks and the ordering of events in a distributed system"
Comm. ACM, Vol.21, Nb.7, July 1978.

13 LE BIHAN, J. et AL.
"SIRIUS : a french nationwide project on distributed data bases"
Proceedings of 6th international conference on VLDB, Montreal, Sept. 1980.

14 LEIFERT, S.
"Un langage de specification des systèmes d'information. Un outil pour leur gestion".
Thèse du docteur-ingenieur - Université de Nancy I - Mai 1980.

15 LE LANN, G.
"Algorithms for distributed data sharing which use tickets"
Proceedings of 3rd Berkeley Workshop on distributed data management and computers network, August 1978.

16 PIROTTE, A. - KONDON, F.
"A comprehensive formal query language for a relational data base : FQL"
RAIRO - informatique / Computer Science, Vol.11, Nb.2, 1977.

17 ROLLAND, C. - FOUCAUT, O.
"Concepts for the design of an information system conceptual schema and its utilization in the Remora project",
Proceedings of the 4th international conference on VLDB, Berlin, 1978.

18 RICHARD, C.
"Une approche conceptuelle des problèmes de synchronisation"
Thèse de 3ème cycle, Université de Nancy I, Oct. 1979.

19 ROLLAND, C. - LEIFERT, S. - RICHARD, C.
"Tools for information system dynamics management"
Proceedings of the 5th international conference on VLDB, Rio de Janeiro, Oct. 1979.

20 RICHARD, C. - ROLLAND, C.
"Architecture de Système d'Information réparti et théorie des processeurs discrets".
SIRIUS Research Report, January 1981.

21 ROLLAND, C. - RICHARD, C. - LEIFERT, S.
"A proposal for information system design and management"
ACM SIGDA, August 1980.

22 ROSENKRANTZ, D.J. - STEARNS, R.E. - LEWIS, P.M.
"System level concurrency control for distributed data base systems"
ACM - T.O.D.B.S., Vol.3, Nb.3, June 1978.

23 SCHMIDT, J.W.
"Some high level language constructs for data of type relation"
ACM - T.O.D.B.S., Vol.2, Nb.3, 1977.

24 SCHNEIDER, F.B.
"Ensuring consistency in a distributed data base system by use of distributed semaphore" in " Distributed Data Bases",
C. Delobel and W. Litwin editors, North-Holland Publ. C°, April 1980.

25 THOMAS, R.H.
"A majority consensous approach to concurrency control for multiple copy data base"
ACM - T.O.D.B.S., Vol.4, Nb.2, June 1979.

26 TODD, S.
"Automatic constraint maintainance and updating defined relations"
Proceedings of IFIP Congress, Toronto, August 1977.

27 WASSERMAN, A.I.
"USE : a methodology for the design and development of interactive information systems"
Proceedings of IFIP Congress, Oxford, 1979.

28 WIRTH, N.
"The programming language PASCAL"
Acta Informatica 1, app. 35-63, 1971.