A CONCURRENCY CONTROL MECHANISM WHICH USES SEMANTIC KNOWLEDGE ABOUT THE APPLICATIONS

Amauri Marques Da Cunha
Universidade Federal do Rio de Janeiro
Rio de Janeiro - *Brasil*

1 - INTRODUCTION

The very fundamental idea which is the kernel of the flexible mechanism described in this work, is that the use of the semantic knowledge about the defined operations on the system objects, may lead to a greater parallelism in the entire system. This approach was studied from the theoretical point of view in |KUNG & PAPADIMITRIOU 79|. In this work, the authors demonstrated that a Concurrency Control mechanism will allow more parallelism, the more it knows about the semantics of the application.

For example, when the mechanism has knowledge of the objects' meaning and the defined operations on these objects, it may generate a greater number of correct **execution histories** |KUNG & PAPADIMITRIOU 79|, then theoretically it will permit more concurrency in the system. The concurrency increased in this way, will be called **potential concurrency**, because we cannot guarantee that it will be advantageous in all real situations, without doing other practical considerations like overheads for example.

The issue of utilizing knowledge of the application to enhance Concurrency Control performance, was studied in several works. The most interesting propositions are |LAMPORT 76|, |GARCIA-MOLINA 83|, |LYNCH 83|, |SCHWARZ & SPECTOR 84|, |ALLCHIN & McKENDRY 83| and |WEIHL 83|. With regard to other works, the originality of the one presented here, is the proposition of a classification in semantic knowledge levels, of all the operations carried out on the system objects. This classification is defined in a naturally hierarchical way which allows, besides the specification of a Concurrency Control mechanism for each level of Semantic Knowledge, the grouping of these specifications in only one flexible mechanism. This mechanism is able to simultaneously manage the requests from all of the semantic levels. When we only have operations with a high degree of Semantic

Knowledge (SK), the mechanism will allow a high degree of potential concurrency. The potential concurrency increases dynamically according to the SK-level of the operations under execution on an object, if we use the proposed methodology and mechanism.

This paper is organized as follows. Section 2 presents the assumptions about the Computer System, and section 3 defines the SK-levels of the operations on the objects. Section 4 describes the necessary mechanisms for each level, and the 5th section is devoted to the important topic of recovery. Section 6 introduces the Concurrency Control Flexible Mechanism with SK utilization and gives an example. Section 7 presents a discussion about our approaches. Finally, in the 8th section we give our conclusion.

2 - ASSUMPTIONS ABOUT THE SYSTEM

The transaction concept has always been successfully used in the Data Bases domain. This concept has been studied and generalized during recent years -see |GRAY 81|. A good example of an extension to this concept, is the one of nested transactions |MOSS 81| and |BEERI et al. 83|.

More recently, several extensions have been proposed for the use of the transaction concept in the construction of distributed operating systems, which are not exclusively conceived for the Data Bases environment, but for more general applications. The reader will find in |SPECTOR & SCHWARZ 83|, a discussion on the subject. The new propositions that appear to be more interesting, are the following:

- The ARGUS Project at MIT |LISKOV 82| and |WEIHL & LISKOV 85|;
- The operating system CLOUDS at Georgia Institute of Technology |ALLCHIN & McKENDRY 83|;
- The ARCHONS Project at Carnegie-Mellon University |SHA et al. 83a| and |SHA et al. 83b|;
- The TABS prototype at Carnegie-Mellon too |SCHWARZ & SPECTOR 84| and |SPECTOR et al. 84|.

With the exception of SHA's works at Carnegie-Mellon, all of these propositions, explicitly modelize the object as **abstract types.** This modelization allows the inclusion of a greater variety of operations in the transactions, which represents a powerful extension for the manipulation of shared objects other than Data Bases.

This approach was adopted in the conception of the mechanism described in this work. We suppose that every system object is an instance of an **abstract data**

type, whose complete and formal specification is produced at the system design time |LISKOV & ZILLES 74|.

All the possible operations on a given object are rigorously defined, and therefore all of the operations' properties will be well known by the transactions that utilize the object. Each object is encapsulated by a kind of monitor that ensures its specification is respected. Additionally, we suppose that each site has only one object (this assumption is made only to simplify the exposition), and that each site has a kernel of the distributed operating system playing the role of transaction manager, as well as treating the synchronization problems, recovery, deadlock, and intersite communications. Other precisions about this approach may be obtained in |SCHWARZ & SPECTOR 84| and |WEIHL & LISKOV 85|.

This work deals particularly with the problem of transactions synchronization on objects. Its goal is to allow the creation of the greatest possible number of transactions **execution histories** on an object, without violation of the consistency constraints. For the intersite synchronization, one will use other techniques, for example the global serialization using timestamping ordering. We will not study these techniques here. For discussion of this subject, the reader could see for example |BERNSTEIN & GOODMAN 82|.

Finally, we chose the **two-phase locking** technique of |ESWARAN et al. 76|, as the basic technique for the proposed mechanism. If necessary, it is possible to combine this basic technique with a multitude of possible variations like the utilization of multiple versions, timestamping, etc. Even the adaptation to utilize the optimistic approach - which is radically opposed to the pessimistic two-phase locking - is easily feasible. Therefore, the mechanism presentation from this point of view, does not mean any loss of generality.

The deadlocks resolution, that is indispensable when one uses locks, is not explicitly treated here. Nevertheless, we will suppose the mechanism has the capability to accept requests to abort transactions, which might be generated by deadlock resolution procedures.

3 - SEMANTIC KNOWLEDGE LEVELS OF THE OBJECT OPERATIONS

This SK-levels classification, was mainly conceived to show the feasibility of a mechanism which is sensitive to the SK-level of the requests of operations made by transactions. A similar classification was proposed and studied from the theoretical point of view by |KUNG & PAPADIMITRIOU 79|.

Nevertheless, observing the more recent propositions of SK utilization to optimize the Concurrency Control, principally those which use the **abstract data types** approach such as |SCHWARZ & SPECTOR 84|, |WEIHL & LISKOV 85|, |WEIHL 83| and |ALLCHIN & McKENDRY 83|, we can remark that all of them have a tendency to mix the Concurrency Control mechanism with the object encapsulation. We mean by this that there is no clear separation between the Concurrency Control algorithms, and the object encapsulating procedures that implement its operations. In this model, the Concurrency Control mechanism has the right to observe the **object contents,** i.e. its internal structure and even the values therein stored, to decide about the compatibility of any two operations. This model was explicitly employed in |WEIHL 83|.

The work |SCHWARZ & SPECTOR 84| intented to separate the two functions, using the compatibility matrixes between the operations. Nevertheless, these compatibility matrixes of the examples given in |SCHWARZ & SPECTOR 84|, do not avoid the obligation that the Concurrency Control has to access the object content (value). So, this model is fundamentally the same as |WEIHL 83|.

To define the SK-levels, we initially established the following two categories according to whether the Concurrency Control mechanism does or does not have the right to observe the object contents to make its decisions. If it has this right, or more particularly if the requests of operation on the object, convey semantic information that allows the mechanism to efficiently use the object contents to decide about their compatibility, we can say that these requests are of a higher semantic level than the others. We consider that these requests are classified at the SK-level-three.

We chose to create a SK frontier at this point, because we consider that any mechanism which is capable of dealing with the SK-level-three will be more

resource consuming than the others which deal with the inferior levels. It should be used for an application, only if it has an advantageous price/performance ratio.

The requests that are of a level inferior to the SK-level-three, convey less semantic information. We consider the SK-level-one is attached to the fundamental operations defined on the object. We state further the SK-level-two, for the operation requests that convey more information than the fundamental operations of the SK-level-one. The additional information is represented by parameters added to the fundamental operations. These parameters amplify the quantity of available information to the Concurrency Control mechanism, leading to more parallelism in the system. The SK-level-zero and four are also defined, simply for completeness. They have already been identified in |KUNG & PAPADIMITRIOU 79| as the higher and lower possible levels of semantic knowledge respectively. In the following we present a SK-levels classification, illustrated by several examples, representing a pragmatic tentative in detailing the one already stated in |KUNG & PAPADIMITRIOU 79|:

SK-LEVEL-ZERO - the request of operation on the object, does not convey any Semantic Knowledge.

> **Example 0.1** - a request of an exclusive lock on the object, like for instance the classical lock WRITE (W) from the Data Bases domain.
>
> **Example 0.2** - in a Real Time System, after the occurrence of an alarm, an emergency transaction is launched. Before evaluating the extension of the problem indicated by this alarm, the transaction has to lock in an exclusive mode a certain number of objects.

SK-LEVEL-ONE - the request of operation on the object, is only composed by one of the fundamental operations that were defined at conception time of the **abstract type**. We recall that the object is an instance of such an **abstract type**. From this formal definition of the operations, one extracts a **compatibility table** between the operations, which gives all the ordered pairs of commutatives operations.

Example 1.1 - the operations READ (R) and WRITE (W) from the Data Bases.

representation of a compatibility table:

column: operation that owns a lock

line: operation that requests a lock

	R	W
R	Yes	No
W	No	No

Example 1.2 - suppose we have a **"strictly FIFO queue"**, with two types of operations defined on: INSERT (I) or DELETE (D) an element. By its own definition of a queue strictly FIFO, this queue does not admit the commutativity between two type I operations. The insertions of two elements have to be executed and validated, in the same order they arrived. The same applies for the type D operations. So, two operations of the same type cannot be compatible.

Furthermore, if the Concurrency Control mechanism only knows the requests arriving to it - in the present case I or D - and if it does not have the right to observe the interior of the object, then it cannot allow simultaneous executions of the I and D operations on the same object. In this situation it would not be capable of ensuring consistency. To demonstrate this, it is sufficient to suppose an initially empty queue. When two operations, one I and the other D, arrive, we can see that the execution order of these two operations is extremely important, because each different execution order gives a different result as well. Then, in this case, we cannot guarantee the commutativity between I and D. For this reason, its compatibility table will not allow any parallelism:

	I	D
I	No	No
D	No	No

Example 1.3 - suppose we have a **"weak FIFO queue"**, with the same operations as in example 1.2. As opposed to the strictly FIFO queue, the **weak FIFO queue** is defined as the one in which the Insert operations can be validated in an order different to the arriving order. The same thing applies to the Delete operations. It means that two type I operations are always commutatives, as are two type D operations. Otherwise, two operations of different types remain incompatible by the reasoning of the example 1.2. This gives the following compatibility table:

	I	D
I	Yes	No
D	No	Yes

This **"weak FIFO queue"** was adapted from |SCHWARZ & SPECTOR 84|, and it was also used in |WEIHL & LISKOV 85| with the name of **"semi-queue"**.

SK-LEVEL-TWO - at this level, we have all the level-one knowledge and, further, for each incompatibility, the possible existing commutativities according to a parameter added to the fundamental operation. This parameter does not change the general nature of the operation, however it conveys more detailed information about the operation meaning, creating in fact one sub-operation for each possible parameter. The parameter absence in a request of operation on an object, simply means that the request is SK-level-one.

Example 2.1 - a bank account object, on which are defined the operations READ (R) and WRITE (W) of the example 1.1, at the SK-level-one. Among all the semantic possibilities of the W operation, we suppose that the more frequent are the **deposits (d)** of any amount of money and the **withdrawals (w)** that do not exceed a certain threshold, as for instance the ones coming from an ATM - Automatic Teller Machine. This kind of withdrawal, which is already subject to constraints, is always paid by the bank. Then we will have the same compatibility table as in example 1.1 for the SK-level-one. When the request specifies a sub-operation W(d) or W(w), it will be necessary to see the following SK-level-two compatibility table:

	W(d)	W(w)
W(d)	Yes	Yes
W(w)	Yes	Yes

We observe in this example, that the perfect commutativities between the (sub-)operations d and w, lead to an important increase of the parallelism. In such a system, it is worthwhile specifying a second SK-level for the WRITE operations.

Example 2.2 - we retake the previous example 2.1. Now we suppose that the occurrence of simultaneous (sub-)operations W(w) will be a rare event in the system. So, the prohibition of the W(w)/W(w) commutativity will penalize only slightly the performance of the new system, with regard to that of example 2.1. In other respects, we estimate that the serialization of all W(w) (sub-)operations, that are made imperative if all pairs W(w)/W(w) become incompatible, will greatly facilitate the design of the other system modules. As an example we can mention protection against errors and frauds. Hence, such a system simplification would bring about a decrease in the number of routines, as well as a reduction in their complexity, leading to an improvement of the overall performance. Here is an illustration of utilization of SK-levels, that may give more flexibility to the System design, in addition to the potential gains of parallelism. In this case, we use the compatibility table below, instead of the table in example 2.1.

	W(d)	W(w)
W(d)	Yes	Yes
W(w)	Yes	No

Example 2.3 - suppose we have a directory object, where each element (entry) is identified by a key formed by a character string. Its fundamental operations are MODIFY (M) an element - which may be insert or delete an element, or simply modify its contents - or LOOKUP (L) an element, or still DUMP (D) the entire directory. At the SK-level-one, the M and L operations cannot indicate the key of the

target element. The D operation is only defined at this level. Here is the compatibility table of the SK-level-one:

	M	L	D
M	N(*)	N(*)	N
L	N(*)	Yes	Yes
D	N	Yes	Yes

(*) the incompatibilities marked with an asterisk, may direct the decision to the SK-level-two, provided that the key of the target element is present as an operation parameter.

Nevertheless, we will not have here a compatibility table to characterize the SK-level-two. The relations M==>M (here the symbol ==> means precede), L==>M and M==>L do not create dependencies |SCHWARZ & SPECTOR 84|, when each operation is realized on a different element of the directory. Therefore it is sufficient to keep a list that contains all the parameters of the M or L operations which own a lock on the object. The verification of the commutativity of one (sub-)operation M(k) or L(k) just arrived (k = key of the directory element addressed by the operation), with the ones already in execution, is done by a simple search on the list (table) of accepted locks.

We recall that the commutativity L(k)/L(k'), even if k'=k, is detected at the SK-level-one, so the test does not have to be done at the SK-level-two.

SK-LEVEL-THREE - at this level, we have the sum of the level-one and level-two knowledge and, in addition, we know that the operations commutativity depends on the object value.

Example 3.1 - a bank account object that has, among the operations defined on it, a general operation of money withdrawal that brings together all the kinds of possible withdrawals. In this case, two withdrawal operations are commutative only if there is enough money in the account.

Example 3.2 - we retake the "strictly FIFO queue" of example 1.3. The incompatibilities I/D and D/I may be removed by a procedure of the Concurrency Control mechanism, that is capable of observing the object contents to make its decision in the following manner:

From example 1.2, we already know that when the queue is empty, there is no commutativity between I and D. There are other equivalent situations, and to show it let us state the variables:

n_q = number of elements (validated) of the queue

n_D = number of D operations that own a lock on the object

n_I = number of I operations that own a lock on the object

Effectively, the limit-situation is reached when $n_q=n_D$, i.e. at the moment the queue is potentially empty. In reality, the queue only becomes truly empty, if all of the D-operations that own a lock on the queue, are validated (we cannot forget that aborts are still possible before validation). Anyway, since we have $n_q<n_D$, we cannot guarantee that a just arrived I-operation, will be compatible with all the E-operations that already possess the lock. To demonstrate this, we suppose that this lock I is granted. If this operation I is validated before the validation of the last D-operation, we will have a different result from the one that would be obtained if the validation of I had been done after the validation of the last E. In this case, the commutativity hypothesis is contradicted, and so the assertion is demonstrated.

It is interesting to observe that, even if the different results mentioned in the above demonstration, do not seem to cause consistency problems, they have to be strictly prohibited. Why?!!... Because the Concurrency Control mechanism has to guarantee a global serialization of the transactions, to ensure the system's global consistency. The possible local commutativities, that allow more parallelism and hence more potential concurrency, are admitted on the condition that it permits (or does not prevent) the construction of a global execution order of the transactions |GARDARIN & MELKANOFF 82|. Thus, the operations on an object that have the risk of giving different results from the semantic point of view, according to their execution (validation) ordering, have all to be serialized at the local level.

Returning to the example, if we have a D-operation arriving and other I-operations that already own a lock, we consider that the limit-situation still depends on the variables n_q and n_D. Effectively, the worst case happens when all the D-operations are validated before all the I-operations. Thus to grant a D-lock on the object concerned, supposing that it already has other I-locks granted, it is necessary that the condition $n_q > n_D$ be true. Furthermore, if we have a request for an I-lock, and the object has already granted other D-locks, we will grant the lock only in the case of $n_q >= n_D$ (we notice that the equality is admitted here).

Example 3.3 - we take again the **"weak FIFO queue"** of example 1.3. Following the reasoning used in example 3.2, we can deduce that even the compatibility D/D, which seemed to be completely natural, could only be decided at the SK-level-three. If we suppose there are only D-locks granted, an arriving request for a new D-lock, can only be accepted if $n_q > n_D$ or if n_q=zero. We summarize the Concurrency Control rules on this object as follows:

a - compatibility table SK-level-one:

	I	D
I	Yes	No
D	No	No

b - SK-level-three procedures:

b.1 - incompatibility of SK-level-one I/D: the request for the I-lock, is granted only if $n_q >= n_D$;

b.2 - D/I: the request D is granted if $n_q > n_D$;

b.3 - D/D: the request is granted if $n_q > n_D$ or if n_q=zero.

Observation - in the given examples, we only studied one exceptional situation of the queues: the possibility that it becomes empty. Clearly, this is the most important exception. However, in practice, we could rarely suppose a queue to be infinite. In these cases, we would also have to treat the possibility of having a full queue. If this happens, we could use reasoning which is entirely analogous to the reasoning used in examples 3.2 and 3.3. For instance, the compatibility I/I will not be

possible at the SK-level-one; it will also be decided at the SK-level-three, using the maximum dimension of the queue.

SK-LEVEL-FOUR - we can imagine that it will be possible in the future, to store and/or analyse the complete Semantic Knowledge about all the transactions and all the objects within a distributed system. With this knowledge, the Concurrency Control mechanism could find all the possibilities of commutativity between operations, as soon as the operations arrive. This highest SK-level, corresponds to the **"optimal schedulers"** that utilize the maximum information possible about the transaction system |KUNG & PAPADIMITRIOU 79|. This type of scheduler has the capability to generate all the possible correct execution histories (logs), and therefore it is optimal with respect to the parallelism.

4 - THE NECESSARY MECHANISMS FOR EACH LEVEL

We need to make some comments about the SK-level classification, before the presentation of the mechanisms.

Initially we notice that there is no sense in classifying at the SK-level-one, a fundamental operation that needs an exclusive access to the object. From the point of view of the approach introduced here, this operation should in reality be located at the SK-level-zero. Even at the SK-level-two, we do not have to define a sub-operation that is incompatible with all the others at the same level. If this happens, we can say that in fact it can be reduced to a SK-level-one, or even to a different SK-level-zero operation.

The WRITE (W) and READ (R) operations, classic in the Concurrency Control study for the Data Bases, already represent a certain SK-level. With our approach, the W operation - that requires an exclusive access to the object - would be classified at the SK-level-zero. At the SK-level-one, we would only have the R operation, which is always compatible with itself.

Finally it is interesting to observe that the SK-level classification can be augmented by the definition of other intermediate levels. It is easy to conceive

for example that the level-two operations may still be divided in other sub-operations by the adjunction of other parameters. Each parameter added to a basic operation, may establish a new sub-level, with its corresponding compatibility tables. The classification proposed here, seems to be a general framework, and is able to accept some refinements concerning how to express the Semantic Knowledge degrees of transaction systems.

4.1 - The Concurrency Control mechanisms used at each SK-level

SK-LEVEL-ZERO - the guarantee of exclusive access, may be obtained by managing a semaphore for each object.

SK-LEVEL-ONE - this is a well-known case, and there is a vast literature about it in the Data Bases domain. From the definition of the fundamental operations, we establish a compatibility table between the operations. The utilization of this table with the two-phase locking protocol |ESWARAN et al. 76|, ensures global consistency. Obviously the mechanism will also manipulate a table which keeps the granted locks.

SK-LEVEL-TWO - it is a generalization of the SK-level-one. According to the object semantics, we can distinguish two types of implementations.

In the first - as in example 2.1 of the previous section - we have one or several compatibility tables of the SK-level-two, each one corresponding to a pair of incompatible operations at the SK-level-one. A SK-level-two table, furnishes the information about the compatibility between the different existent sub-operations, derived from the fundamental operations. The Concurrency Control algorithm, in deciding about the compatibility between a certain pair of sub-operations, must first find the appropriate compatibility table, and then it applies the two-phase locking protocol as in the SK-level-one.

The second type of implementation is for instance, the one in example 2.3. Here, instead of consulting a compatibility table for the SK-level-two, the algorithm stores in a list, the parameter of each sub-operation that has a granted lock. This supplementary storage may be added to the proper lock table to facilitate the manipulations. An arriving lock request, will only be accepted if its parameter is not in the list (table) of granted locks. Naturally the logical

complement of this algorithm, should be available as a variation of this type of implementation; in this case it should only accept the sub-operations having the same parameter as another already in the granted locks table. Obviously, this second type of implementation fits only in the situations where the parameters' diversity cannot be limited, or is too great, as in example 2.3. Otherwise, the first type with its compatibility tables would be sufficient.

The choice of the type of implementation and its possible variations, has to be done as a preliminary, at the very moment of the conception and encapsulation of the abstract object.

SK-LEVEL-THREE - at this level, the mechanism has the right to observe the object contents to make its decisions. Example 3.1 of the previous section, illustrates this idea very well.

Each object that is able to accept SK-level-three operations, will be initialized with a special-purpose procedure, that verifies the commutativity of an arriving request of operation on the object, with all the others which already own a lock at that moment. This procedure may go as far as consultating the semantic information conveyed by all the other operations that are present in the lock table. This procedure may even execute the just arrived operation, using the current (validated) object value, to accomplish its verification.

However, there are simpler situations, as in examples 3.2 and 3.3. Effectively, the necessary information on these objects, can be summarized within counters for these examples. If these counters are stored in main memory, this access to the object will not be much more expensive than the (mandatory) access to the lock table.

SK-LEVEL-FOUR - we are not yet capable of presenting the specifications for this level. Nevertheless, previous work |LAMPORT 76| has demonstrated its feasibility.

5 - THE RECOVERY PROBLEM

The need to abort a transaction, may appear at random. The main events that give rise to an abort are:

- faults
- deadlocks
- Promptness Control
- user request

Apart from in very special-purpose systems, normally the Concurrency Control mechanism has to be able to deal with aborts in a way that preserves consistency, and has to allow transactions recovery. The classical way to abort a transaction, is to undo it, by resetting all the objects that were modified by the transaction, to the initial state encountered by the transaction. However, when we use semantic knowledge that induce the commutativity of operations that **modify the object state**, we cannot simply reset the initial state, because this would lead the system to an inconsistent state. We recall that the commutativity of the operations on an object, allows that the local order of execution (and validation) of the commutative operations, is different from the global serialization order of the transactions which those operations belong to.

Thus, it is necessary to have a supplementary hypothesis to solve this problem. We make the same supposition as that proposed by |GARCIA-MOLINA 83|, i.e., we suppose that each operation involved in a commutativity relation, has a counter-operation that is capable of undoing from the semantic point of view the original operation. This counter-operation has to be imperatively commutative with the other operations and counter-operations belonging to the same commutative set of its original operation.

A counter-operation may be represented either by the inverse operation on the abstract object, or by another compensating operation. In any case it must not demand the allocation of objects other than the object already concerned, because on this object, the transaction has always the suitable lock, even during the abort execution.

This additional hypothesis is, of course, very strong and, consequently, greatly reduces the applicability of the proposed semantic levels. We note that even the

existence of commutative operations, was an already very restrictive characteristic, to which we are obliged to add the existence of the corresponding counter-operations. On the other hand, there are certainly many particular systems, where the mechanisms proposed here, considerably increase the concurency all over the system - see for instance |GARCIA-MOLINA 83|, |LYNCH 83|, |SCHWARZ & SPECTOR 84|, |WEIHL & LISKOV 85|, and |WEIHL 83|.

6 - THE CONCURRENCY CONTROL FLEXIBLE MECHANISM

Observing the description of the Semantic Knowledge levels zero, one, two or four, one notices that there is a perfect hierarchy between these levels. The SK-level-zero does not permit any parallelism on the object. The SK-level-one has a compatibility table that establishes the possible permitted concurrencies between the operations defined on the object. The SK-level-two only exists for each pair of SK-level-one incompatible operations. Hence, we clearly see the possibility of simultaneously controlling the two levels: the one and the two. It is sufficient to have a hierarchical structure on which the locking is made, and that this structure be attached to the existent compatibility tables within each level. This reasoning can equally be applied to the SK-level-three, that can be reached either bv a SK-level-two incompatibility or by another SK-level-one incompatibility. To correctly solve all concurrency cases that may occur on the object, it will be sufficient to control an unique tree of locks.

Thus, we can use an unique mechanism to control the concurrency on a object, disregarding the SK-level of the transactions that utilize this object. And further, such a flexible mechanism, will allow more parallelism as the transactions (operations) Semantic Knowledge is higher. It dynamically adapts itself, in a natural way with respect to the arriving transactions (operations), i.e., if it treats a group of requests that conveys a high SK-level, it will allow more potential parallelism. On the other hand, when an arriving group of requests (of operations) has at least one low semantic level request, the degree of concurrency automatically decreases by the mechanism functioning during all the period this request owns its lock.

6.1 - An example of the Flexible Mechanism

We suppose an object that accepts the SK-levels one and two. The fundamental operations defined on it, are the current ones usually made in the Data Bases Systems domain:

R - only has the right to read the object

W - has the right to modify the object

The SK-level-one is described by the following compatibility matrix CM1:

CM1:

	R	W
R	1	0
W	0	0

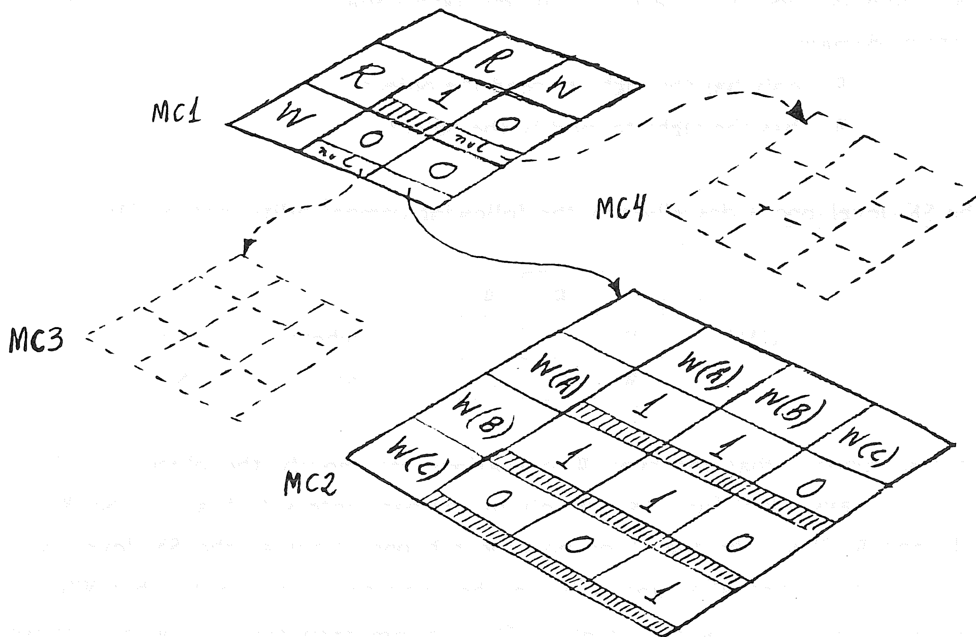
where 1=Yes
and 0=No

We also suppose that the type W operations can modify the object, in three different ways, identified respectively by the parameters A, B and C. So, W(A), W(B) and W(C) will be the three possible sub-operations at the SK-level-two. We still admit that W(A) and W(B) can be executed in any order, but W(C) is commutative only with itself. The commutativities between these sub-operations, are expressed by the compatibility matrix CM2:

CM2:

	W(A)	W(B)	W(C)
W(A)	1	1	0
W(B)	1	1	0
W(C)	0	0	1

To show clearly the hierarchical aspect, we represent the matrixes together, adding a pointer (that may contain the nil value) to the SK-level-two within each CM1 element. The following figure represents this:


MC1
R
R
W
1
0
W
0
0
MC4
MC3
MC2
W(A)
W(A)
W(B)
W(C)
W(B)
W(C)
1
1
1
0
0
1
0
0
1

If the incompatibilities R/W and W/R had some possibilities of compatibility at the SK-level-two, they could be represented for example, by the matrixes CM3 and CM4 of the above figure.

The operations requested by the transactions, always have a well specified SK-level. In our example, if a W operation does not convey a parameter, it will be considered a SK-level-one request. During all the time this operation has a lock, it has to prohibit the simultaneous execution of other W's, even if they are of the SK-level-two. Therefore, a SK-level-one lock, can block one or several possibilities of the SK-level-two - each set of (semantic) possibilities being represented by a SK-level-two compatibility matrix.

On the other hand, each SK-level-two W operation will only allow (possibly) other concurrent W's of the same SK-level. As long as there is at least one SK-level-two W, no other SK-level-one W (and no other R either) can be executed.

This example shows the need of a hierarchical locking table for this kind of mechanism. The annex shows an implementation of the proposed mechanism, and also describes its more important aspects.

7 - A DISCUSSION ABOUT OUR APPROACHES

The two principal approaches used in the conception of the proposed mechanism, are discussed in this section.They are the shared abstract types approach to model the system's objects, and the semantic knowledge levels approach that utilizes semantic knowledge about the application.

7.1 - The Shared Abstract Types Approach

This approach was well studied in |SCHWARZ & SPECTOR 84|. The same approach has been developped at the MIT with the name of atomic data types |WEIHL & LISKOV 85|.

7.1.1 - ADVANTAGES

The first and principal advantage is inherent to this approach. Effectively, the manner of modeling objects as abstract types instanciations, is already oriented towards the distributed model of computing, therefore facilitating the development of distributed systems in general. We recall that the object is the unit of permanent data within a system. This unit is used to control the concurrent transactions. On the other hand, the manner of modeling objects in the classical Data Bases approach, requires the utilization of special and often very expensive techniques to pass from the centralized to a distributed framework.

The following advantages, are listed according to the classification of required characteristics of a Real Time Distributed System, described in |LE LANN 83|.

FLEXIBILITY: this approach encapsulates the objects with some modules, which constrains the objects to behave according to their specifications. Hence, there is an obvious modularity in this approach. To change or adapt the behaviour of an object, to fit new specifications, we can envelop it with a new encapsulation module |WEIHL & LISKOV 85|. We can still encapsulate several different objects to unite them. This modularity considerably increases the system's flexibility.

CORRECTNESS: an abstract type specification may be done through a mathematical formalization. The proof of correctness of all the operations on an object, is somewhat inherent to this approach. Consequently, this rigor facilitates the transactions construction, and also the establishment of a formal demonstration of the application's correctness |LISKOV & ZILLES 74|.

RELIABILITY: the abstract type approach makes possible an atomic treatment of the object, as the one studied in |WEIHL & LISKOV 85|. Thus, adding recovery procedures and even fault tolerant procedures to the object encapsulation |WEIHL & LISKOV 85|, we could ensure a great reliability at the local level. Its extension to the global level will depend on other mechanisms, which will cope with the global atomicity and resilience, and will enhance the local reliability.

PROMPTNESS: in some particular cases, this characteristic may be improved. We take the example of an object that admits compensating operations, like the ones mentioned in section 5. We can imagine that it will be possible in this case, to anticipate the validation of an operation that modifies the object. If the possibility of compensation does not endanger the consistency, therefore we can obtain an improvement on the promptness.

7.1.2 - DISADVANTAGES

SPECIFICATION OF THE ABSTRACT TYPES: this is an inherent disadvantage of this approach. We are obliged to conceive and develop (and still program), the abstract types that are necessary to the application. This handicap is represented in practice by a high development cost in comparison with the other classical approaches. We already know that in the future, we will have languages and/or distributed operating systems, which will offer some primitives to assist the specification of abstract types, and even possibly, the complete encapsulation for the more frequently used types |WEIHL & LISKOV 85|. Though this disadvantage tends to decrease with time, it will not be eliminated.

OVERHEAD: obviously, this approach considerably increases the overhead for all the object utilizations. The first experiments reported, for example the TABS system from Carnegie-Mellon University |SPECTOR et al. 84|, confirm this assertion. However, we are sure that at least for the small transactions or in general for applications that are simple, this approach is not worthwhile. On the other hand, for the distributed systems of great complexity, that have large size transactions, we can catch a glimpse of good results for the parallelism degree, for the management of the complexity, etc. It is still necessary to work out a lot of studies and researches to find the trade-offs.

7.2 - The Semantic Levels Approach

7.2.1 - ADVANTAGES

I - the first advantage is to allow much more potential concurrency in numerous cases, like for example the ones mentioned in section 3.

II - this approach only adds an overhead to the Concurrency Control mechanism, if it has to cope with the highest SK-levels. Indeed, take the example of an implementation of the SK flexible mechanism presented in section 6. Observing it, we ascertain its modularity with respect to the SK-levels. The semantic lock table, the storing of the compatibilities, and the procedures corresponding to the highest SK-levels (especially two and three), are only used to treat the operations requests with these highest levels. As soon as we are sure of having only SK low levels operations (as zero or one for example), we can remove practically all that were added to manage the other levels, bringing the mechanism to the same overhead of a classical mechanism.

This reasoning can also be applied to the dynamic behaviour of the system. Suppose we have on a site, the complete mechanism to cope with all the SK-levels. To manage the requests of operations of the SK-level-zero or SK-level-one, the mechanism will not have in practice any increase in its time overhead with regard to the classical mechanisms. Only in the case of the arrival of higher SK-level requests, will the mechanism consume more execution time. Nevertheless, at this moment, we will have other gains from the point of view of the parallelism.

III - the utilization of SK-levels, allows us to obtain some profits with a common situation in the practice of transaction systems. Particularly in the Real Time Systems, the transactions are specified, developed and catalogued previously, to be dynamically launched later during the system's life |DA CUNHA & TEIXEIRA 84|. In order to keep a sufficient degree of generality for the transactions, we can think they will be catalogued with the lowest possible SK-level about the objects. Otherwise, at the moment of a transaction dispatching, the transactions manager will have more precisions about the current state of the system,

allowing in this manner to augment the SK-level of the transaction. Thus, we can envisage the "refinement" of the SK of a transaction already catalogued, dynamically from the moment it has to be launched. This "refinement" may also occur at the moment of dispatching a local action on a site, because at this moment it could take advantage of the known results of the (already executed) previous actions.

The example 0.2 in section 3, illustrates this situation, with the emergency transactions of a Real Time System.

IV - the above advantage refers to the strategy of static allocation of resources. The strategy of dynamic allocation can however take advantage of SK-levels utilization. Here, the moment when the transaction requests the lock(s), i.e., the moment of dispatching the local action on a site, it is precisely when there is the highest possible SK, and then this is the best moment to decrease the probability of conflicts, and consequently increase the potential parallelism. In attaining a diminution of the probability of conflicts occurrence, one favours the dynamic allocation of objects by the decreasing of the aborts percentage, which is the principal handicap of this strategy.

7.2.2 - DISADVANTAGES

A major disadvantage of this approach, is the limitation of its utilization to only certain kind of applications. We refer the reader to section 5, for a discussion of this subject.

8 - CONCLUSION

We recall that the theoretical base of the proposed mechanism, is the work developed in |KUNG & PAPADIMITRIOU 79|, where the authors demonstrated the existent relation between the Concurrency Control permissiveness and the semantic knowledge about the application. Using this idea, we defined four levels of Semantic Knowledge (SK) that are hierarchically organized. We suppose the objects are modeled as abstract types. Thus, each request of an operation on an object, conveys a well-known SK-level, which is used by our SK

Flexible Mechanism. The higher the SK-levels of the requests received by the mechanism on an object, the more parallelism it allows on this object. Using our Flexible Mechanism, the parallelism increases and decreases automatically, according to the SK-levels of the requests on the object.

In the literature, we find a similar idea in the protocol proposed by |GRAY et al. 76|, to deal with the granularity of objects in a Data Base. Our proposed framework can also be applied to control the concurrency on variable granularity objects - see example 2.3 of section 3. Nevertheless, the hierarchical protocol used in |GRAY et al. 76|, is based on the more "physical" concept of object granularity, whereas ours is based on the logical nature of the operations on the object. So, our proposition is applicable to more general situations - see section 3. We can still note that the **"intention modes"** of |GRAY et al. 76|, when applied to a node, prevent undesirable accesses to the underlying structure. There is an analogous situation in our mechanism, when a low SK-level operation acquires a lock on a object and therefore blocks undesirable accesses even if they have a high SK-level (see section 6).

The work |SCHWARZ & SPECTOR 84| had a lot of influence on our proposition. Our approaches are significantly the same. From the description of the examples studied in |SCHWARZ & SPECTOR 84|, it emerges that their Concurrency Control mechanism can explicitly utilize the parameters of the fundamental operations defined on the object, in a way that is entirely analogous to our SK-level-two. Furthermore, their mechanism needs in several cases, to observe the object contents, as in our SK-level-three. This appears only implicitly in the paper |SCHWARZ & SPECTOR 84|, where the mechanism is always presented in the form of a compatibility table. In this manner, they cannot treat correctly the "weak FIFO queue" of example 3.3 of section 3, in the cases where the queue may be contingently empty (or contingently full).

Hence, the utilization of compatibility tables between the operations - even adding parameters to these operations - does not seem sufficient to correctly express all the possible cases of concurrency on the abstract types. Effectively, it is still necessary to add special-purpose procedures, that are capable of "refining" the verification of the commutativity, in all the possible situations that are not provided by the compatibility table. This is done in a very natural way in our SK-levels proposition, but is not explicitly treated in the proposition

|SCHWARZ & SPECTOR 84|.

The paper |WEIHL 83| introduces the notion of **dynamic atomicity**. It models the Concurrency Control mechanism (scheduler) and the encapsulation of the object (modeled itself as an abstract type), into a single module. This **dynamic atomicity** clearly corresponds to our SK-level-three. The framework of |WEIHL 83| is less flexible than ours, because it does not allow simpler (and less expensive) treatments.

Finally, we comment on the propositions of the CLOUDS Operating System |ALLCHIN & McKENDRY 83| with respect to ours. CLOUDS proposed four conceptual levels for the Concurrency Control mechanism, with the same idea that the higher the level, the more semantic information is available to the mechanism, and so the more parallelism is allowed. There is a fundamental difference between the conceptual levels of CLOUDS and the SK-levels. The CLOUDS conceptual levels are mutually exclusive by definition, i.e., they cannot simultaneously coexist within the same object and in the same application. This is due to the fact that they were not defined in a hierarchical manner, and so the mechanism cannot traverse, in a way that preserves the consistency, from one level to another, as we can do with the SK-levels. In this sense, our proposition allows much more flexibility.

We hope the framework proposed here, may lead to a more comprehensive study of the Concurrency Control on objects modeled as abstract types.

REFERENCES

|ALLCHIN & McKENDRY 82| J.E. ALLCHIN et M.S. McKENDRY
"Object-based Synchronization and Recovery"
Georgia Institute of Technology, Technical Report GIT-ICS-82/15, (septembre-1982) 20p.

|ALLCHIN & McKENDRY 83| J.E. ALLCHIN et M.S. McKENDRY
"Synchronization and Recovery of Actions"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.31-34

|BEERI et al. 83| C. BEERI, P.A. BERNSTEIN, N. GOODMAN, M.Y. LAI et D.E. SHASHA
"A Concurrency Control Theory for Nested Transactions"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.45-62

|BERNSTEIN et al. 80| P.A. BERNSTEIN, D.W. SHIPMAN et J.B.ROTHNIE JR.
"Concurrency Control in a System for Distributed Databases (SDD-1)"
ACM Transactions on Database Systems, Vol.5, No.1 (mars-1980) pp.18-51

|BERNSTEIN & GOODMAN 81| P.A. BERNSTEIN et N. GOODMAN
"Concurrency Control in Distributed Database Systems"
Computing Surveys, Vol.13, No.2 (juin-1981) pp.185-221

|BERNSTEIN & GOODMAN 82| P.A. BERNSTEIN et N. GOODMAN
"A Sophisticate's Introduction to Distributed Database Concurrency Control", Proc. 8th Int. Conf. on Very Large Data Bases, Mexico (septembre-1982) pp.62-76

|CAREY & STONEBRAKER 84| M.J. CAREY et M.R. STONEBRAKER
"The Performance of Concurrency Control Algorithms for Database Management Systems", Proc. 10th Int. Conf. on Very Large Data Bases Singapore (août-1984) pp.107-118

|DA CUNHA & TEIXEIRA 84| A.M. DA CUNHA et M.J. TEIXEIRA
"SIGMA - Um Executivo Distribuido para Ambientes Automatizados em Tempo Real", Proc. 5° Congr. Bras. Automatica
Campina Grande (septembre-1984) pp.517-522

|ESWARAN et al. 76| K.P. ESWARAN, J.N. GRAY, R.A. LORIE et I.L. TRAIGER, "The Notions of Consistency and Predicate Locks in a Database System", Communications of the ACM, Vol.19 No.11 (novembre-1976) pp.624-633

|GARCIA-MOLINA 83| H. GARCIA-MOLINA
"Using Semantic Knowledge for Transaction Processing in a Distributed Database", ACM Transactions on Database Systems, Vol.8 No.2 (juin-1983) pp.186-213

|GARDARIN & LEBEUX 77| G. GARDARIN et P. LEBEUX
"Scheduling Algorithms for Avoiding Inconsistency in Large Databases" Proc. 3rd Int. Conf. on Very Large Data Bases,
Tokyo (octobre-1977) pp.501-506

|GARDARIN & MELKANOFF 82| G. GARDARIN et M. MELKANOFF
"Concurrency Control Principles in Distributed and Centralized Databases"
INRIA - Rapport de Recherche No.113 (janvier-1982) 91p.

|GRAY et al. 76| J.N. GRAY, R.A. LORIE, G.R. PUTZOLU et I.L. TRAIGER
"Granularity of Locks and Degrees of Consistency in a Shared Data Base"
Proc. IFIP Working Conf. on Modelling in Data Base Management Systems
Freudenstad (janvier-1976) pp.695-723

|GRAY 81| J.N. GRAY
"The Transaction Concept: Virtues and Limitations",
Proc. 7th Int. Conf. on Very Large Data Bases,
Cannes (septembre-1981) pp.144-154

|KORTH 83| H.F. KORTH
"Locking Primitives in a Database System",
Journal of the ACM, Vol.30 No.1 (janvier-1983) pp.55-79

|KUNG & PAPADIMITRIOU 79| H.T. KUNG et C.H. PAPADIMITRIOU
"An Optimality Theory of Concurrency Control for Databases"
Proc. ACM SIGMOD Int. Conf. on Management of Data,
Boston (mai-1979) pp.116-126

|LAMPORT 76| L. LAMPORT
"Towards a Theory of Correctness for Multi-user Data Base Systems"
Technical Report CA-7610-0712, Massachusetts Computer Associates Inc.
(octobre-1976)

|LE LANN 83| G. LE LANN
"On Real-Time Distributed Computing"
Proc. IFIP-83 Congress, Paris (septembre-1983) pp.741-753

|LISKOV 82| B. LISKOV
"On Linguistic Support for Distributed Programs"
IEEE Transactions on Software Engineering, Vol.SE-8 No.3 (mai-1982)
pp.203-210

|LISKOV & ZILLES 74| B. LISKOV et S. ZILLES
"Programming with Abstract Data Types"
Proc. ACM SIGPLAN Symposium on Very High Level Languages
Santa Monica (mars-1974) pp.50-59

|LYNCH 83| N.A. LYNCH
"Multilevel Atomicity - A New Correctness Criterion for Database Concurrency Control"
ACM Transactions on Database Systems, Vol.8, No.4 (décembre-1983)
pp.484-502

|MOSS 81| J.E.B. MOSS
"Nested Transactions: An Approach to Reliable Distributed Computing"
Thèse Ph.D. MIT, Technical Report MIT/LCS/TR-260 (avril-1981) 178p.

|SCHWARZ & SPECTOR 84| P.M. SCHWARZ et A.Z. SPECTOR
"Synchronizing Shared Abstract Types"
ACM Transactions on Computer Systems, Vol.2, No.3 (août-1984)
pp.223-250

|SHA et al. 83a| L. SHA, J.P. LEHOCZKY, E.D. JENSEN et N. PLESZKOCH
"Data Consistency and Transaction Correctness - A Modular Approach to Non-serializable Transactions"
CARNEGIE-MELLON University, draft (16-août-1983) 28p.

|SHA et al. 83b| L. SHA, E.D. JENSEN, R.F. RASHID et J.D. NORTHCUTT
"Distributed Co-operating Process and Transactions"
in Distributed Computing Systems - Synchronization, Control and Communication, Y.PAKER ed., ACADEMIC PRESS (1983) pp.23-50

|SHA 84| L. SHA
"Modular Concurrency Control and Failure Recovery ...Consistency, Correctness and Optimality"
Thèse en préparation à l'Université CARNEGIE-MELLON, draft (29-août-1984) 12p.

|SPECTOR & SCHWARZ 83| A.Z. SPECTOR et P.M. SCHWARZ
"Transactions: A Construct for Reliable Distributed Computing"
ACM Operating Systems Review, Vol.17, No.2 (avril-1983) pp.18-35

|SPECTOR et al. 84| A.Z. SPECTOR, J. BUTCHER, D.S. DANIELS, D.J. DUCHAMP, J.L. EPPINGER, C.E. FINEMAN, A. HEDDAYA & P.M. SCHWARZ
"Support for Distributed Transactions in the TABS Prototype"
CARNEGIE-MELLON University,
Technical Report CMU-CS-84-132 (juillet-1984) 25p.

|WEIHL 83| W.E. WEIHL
"Data-dependent Concurrency Control and Recovery"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.63-75

|WEIHL & LISKOV 85| W.E. WEIHL et B. LISKOV
"Implementation of Resilient Atomic Data Types"
ACM Transactions on Programming Languages and Systems, Vol.7,No.2 (avril-1985) pp.244-269