SBMAHTIC IHflBftlTI TBAIBACTIORS IH DESIGHDATABASES

by

CharlM M. lutMn & Gil«« M.S. Lafue

DBC-15-12-82

April, 1982

Semantic Integrity Transactions in Design Databases

by [1]

Charles M. Eastman
Institute of Building Sciences
and Design Research Center
Carnegie "Mellon University
Pittsburgh, Pa. 15213

and

Gilles M.E. Lafue[2]
Department of Computer Science
Rutgers University
New Brunswick, N.J. 08903

This paper views engineering design as a task involving simultaneously defining and assigning values to a database. Such a database (called a *design database),* represents the artifact being designed throughout many phases, from early specifications all the way to manufacturing instructions. This paper examines some of the semantic inegrity requirements of design databases in relation with current research, and proposes a refinement of the notion of integrity transaction, i.e., a structure to manage the semantic integrity of a database without necessarily guaranteeing it all the time. Semantic integrity is also discussed within the larger framework of confidence in data values of design databases.

1. Introduction

A major concern in database research has focussed on how to define schemas in terms of record types and relationships between these types (see all data models). But once a database schema is defined, assigning values to the database can pose problems. Although, such problems have received attention, many remain unsatisfactorily solved, especially for *design databases.*

A design database is a database which supports the design of a complex artifact from early specifications, to full design development and to manufacturing instructions. Many efforts throughout the world are being directed toward the development of such databases to support the design of complex artifacts such as buildings [9], ships [4], process plants [21], aircraft [23], or digital circuits [22].

Like most database applications, those in engineering design aim to integrate multiple stand-alone applications, including engineering analyses, design checking programs, automatic synthesis programs and production of drawings and numerical control tapes for fabrication. It is expected that, for a given project, a common database will be used by multiple groups of designers. These designers will interactively build up the database, i.e., assign its contents. For example, in building design, it is expected

[1]This work has been supported in part by the National Science Foundation, grant number MCS-78-22328, and in part by the U.S. Army Corps of Engineers, contract number 79-C-0022.

[2]This work has been partly supported by the National Science Foundation, grant number MCS-81-101QCL-

that architects, structural and mechanical engineers, and interior layout people will all be accessing portions of the same database, organized to support each group's design activities. The reason for closely integrating each group's work through a common database is to enhance consistency across the decisions and resulting data that each group is generating. For example, a common database for building design allows automatic checking of the geometrical layout (to eliminate spatial conflicts) among all building subsystems, and automatic (re-)computation of the structural loads (when major equipment is moved by the architects or mechanical engineers). A design database is aimed at integrating the diverse information previously held on drawings and written specifications into a single repository, allowing improved management, consistency checking, and control, while automating the conversion of data for various applications.

The question of how to assign data values is particularly important for a design database, because the main purpose of such a database is, by nature, to be built, that is, to be both logically defined and filled with values (these two acts are often intermingled in time). Its archival role after design completion is secondary[3] . This is in contrast to other kinds of databases which are more or less static repositories of information, and are mostly meant to be consulted.

The confidence that the contents of a database represents a possible real world situation, may be hard to measure. Several factors can affect this confidence. One important factor is the respect of *semantic integrity constraints,* that is, constraints which restrict data values so that the set of values of a database is semantically consistent with the reality being modelled.

Two major types of integrity constraints can be conceptually distinguished. Constraints of the first type specify the cardinalities of relations between data types, that is, the number of instances of various types which can exist in relation with each other. An example of this type is that a pipe has exactly two end connections. We shall call these *existence integrity constraints.* Constraints of the second type state the relationships between the values of the related data. For example, the diameter of the pipe should be constrained to be equal to that to which it is connected. We shall call these *value integrity constraints.*

Database schemas are traditionally based on existence constraints only. Moreover, the relation cardinalities they use are generally no more precisely defined than "one" or "many" (Abrial's model is an exception in this respect [1]). Thus, a database can respect the rules of-the particular data model with which its schema is defined, and still be meaningless. More integrity constraints than those implicit in schemas are needed.

Examples of integrity constraints in design are numerous. The task of design varies in the different design fields, and within the same field, on different projects. But, common to these tasks, is the requirement to specify a set of elements and their composition. The specifications must satisfy a wide and sometimes ill-defined set of criteria. Such criteria express both that the artifact must be *feasible* (i.e., realizable), and also achieve some required *performances.*

Feasibility involves a large set of criteria, generally defined as constraints, that correspond, for instance, to physical laws. Examples include prohibiting intersections in space and time (no spatial conflicts), requiring correspondence between the topological connectivity of a system and its geometric properties, conservation of fluid, sound, statical or thermal flows (the flows into a pipe or a finite element

[3]Other database systems usually not regarded as design databases, can also exhibit these characteristics, such as decision support systems for corporate strategy planning.

must equal the flows out of it). Feasibility normally relies on a number of criteria reflecting current technology or practice, e.g., limitation to what is currently technologically available, minimum tolerances in fabrication, requirements for testing. Feasibility must also reflect resource limitations, e.g., in terms of material, dollars, space, or labor [11].

The performances of an engineering artifact are usually determined by analysis programs, such as structural analyses, analyses of thermal conductivity, or by simulation, e.g., of a digital circuit, or of pedestrian flow in the design of a transportation facility. Performance is sometimes evaluated intuitively, e.g., the visual qualities of a building. Criteria regarding these performances are compared to the performances predicted of a proposed design by the analyses. The design elements and composition are varied until the criteria are met Often the criteria change during this process. In well-structured sub-problems, optimization can be applied. The management of these criteria is a major task of engineering and occupies a significant portion of an engineering team's efforts.

Many feasibility and performance criteria can be expressed as semantic integrity constraints, that apply to a design database in the same manner that semantic integrity constraints have in databases in general.

This paper situates the needs of design databases regarding semantic integrity management with respect to the current state of research, and proposes directions to investigate. In particular, it refines and puts to use for design databases a notion which is central to integrity management, namely, the notion of integrity transaction. An integrity transaction is traditionally defined as an operational unit such that some guarantee about semantic integrity can be made before and after this unit, but not during.

Next section introduces some major aspects of semantic integrity management, particularly integrity transactions. Section 3 presents the notion upon which our revision of integrity transactions is partly based, that integrity dependencies between data values in design are often directed. In section 4, we observe that in general, the integrity which is guaranteed between transactions cannot be that of the whole database, but only of parts of the database. This forces a re-definition of integrity transactions. Section 5 addresses relationships between integrity transactions such as nesting and time ordering. Finally in section 6, semantic integrity management is placed within the larger context of confidence in database values, and identifies other factors of this confidence for design databases.

2t. Components of Integrity Management and Their Organization into Integrity Transactions

The management of the semantic integrity of a database has three components:

- *Expression* of integrity constraints, e. g., as *assertions* that apply to the database. Of course, many design constraints cannot be expressed with assertions in any formal language. The present concern is therefore initially limited to those constraints that can be expressed formally, although this issue will be revisited later.

- *Checking* integrity constraints. This is to be considered with regard to the fact that the database integrity is potentially threatened by every update, and therefore, every update is associated with some checking.

- *Maintenance* (i.e., enforcement) of integrity constraints. This usually consists of rejecting or undoing updates which cause violations, or of performing further updates to compensate for the faulty ones.

The expression of an integrity constraint includes the *qualification* of the units of data, or *variables*, i.e., records (or tuples) or record attributes (or tuple components), to which the constraint applies. Variables are qualified by the conjunction of their type and values. The qualification of variables by their values can be arbitrarily complex, e.g., heavy equipement may be qualified for its inclusion in structural considerations according to its weight, location, criticalness of its function, etc... If a qualified value is that of a database key for a given type, then at most one instance of this type is concerned by the constraint. On the other hand, a constraint may apply to all the variables of the specified type regardless of their values, as in the constraint that no 3-D object can spatially intersect with any other. Thus, the association of variables and constraints can be primary or be side effects of variable and constraint creations, deletions and updates.

Database updates, whether aimed at integrity management or not, primarily consist of creation, update, read and deletion of variables. In addition, database updates sometimes consist of creating, deleting, or updating an integrity constraint. Updating a constraint consists of altering the qualification of the sets of variables to which the constraint applies, or the relationship between these variables. For example, a piping system may be initially laid out and sized for static loading. Later, it may be checked for dynamic loading, which may result in updates of previously defined values. For convenience, we shall say that a variable or a constraint is manipulated when it is updated, created or deleted.

In current database practice, the database administrator (DBA) instructs users (i.e., application programmers and users) how to update the database in a logically consistent manner. Users then take responsibility for verifying and maintaining integrity constraints. When constraints are application independent, this can lead to redundancies and inconsistencies. An alternative consists of incorporating application independent constraints into the database or into the operations for manipulating data, so that they can be automatically checked.

There are essentially two major approaches for the selection of the integrity constraints to check when a variable is manipulated. In the first approach, the checking of a particular constraint is incorporated into the operators for manipulating the data types to which the constraint applies. This incorporation is part of data type definition. It corresponds to the notion of abstract data type developed in programming languages, in which the behavior of a data type allowed by its manipulation operators is part of the type's semantics. While it binds the definition of integrity assertions to the definition of data type operators, it does not necessarily imply that constraint definition are burried into the code for these operators, as opposed to being declaratively defined. Such assertions also serve to specify the effects of the operators, and thus, contribute to program verification. In this approach, the constraints to check when a variable is manipulated are those which are associated with the variable's type, and whose qualification matches the variable's current value. The authors have been part of a group that has designed and implemented a database language incorporating both complex schema definitions and procedural capabilities for managing integrity constraints along the lines supported by data abstraction [10].

In the second approach, assumed by many database researchers, the Data Base Management System (DBMS) performs the selection of the constraints to check when variables are manipulated. This selection can be performed at the time the update is executed, e.g., [20], or for limited classes of constraints, it can be sketched at compile time, e.g., [15]. In particular, the checking of some constraints on sets of data can be automatically optimized [5], [13], [17]. This approach allows the separation of explicit data and constraint definitions.

Two main features provided to the users by the more advanced DBMS's are the ability to define database conditions upon the satisfaction of which some checking (or anything else) is done (e.g., triggers in System R [2], or alerter conditions [6], or demons elsewhere,...), and *integrity transactions,* e.g., as in System R and Query-By-Example [24].

An integrity transaction is the execution of a collection of operations so that integrity can be guaranteed before and after this execution, but not necessarily during. For instance, in an architectural plan, four walls may intersect to form a room. The relocation of one wall may eliminate the bounding condition of the four walls, and require that two others have their sizes altered. Also, the room shape will have to be re-computed. Thus one wall re-computation may violate integrity constraints regarding the bounding conditions of other walls and spaces. Multiple updates are required to regain the integrity of these conditions. All these actions together constitute an integrity transaction. The satisfaction of these integrity constraints is not guaranteed before the end of the transaction.

When writing his/her application programs or queries, the user indicates the collections of operations whose excution will constitute integrity transactions. To the extent that it is generaly intended, among other things, to assign meaningful values, the execution of an entire application, or of a short query, can be seen as an arbitrarily large or small integrity transaction. The notion of transaction as a collection of operations semantically tied together originated in other aspects of data (or resource) management, e.g., concurrency control or error recovery.

Given these abilities and responsibilities, there is a wide choice as to when checking can be done with respect to maintenance, and maintenance with respect to checking. Ah update can be checked either before or after it is committed to the database. In the latter case, it can be checked either immediately or some time after the update was committed. Delays in checking can extend until the end of a user defined transaction, or until the satisfaction of a user defined condition, or until an explicit request by the user; The maintenance following the detection of some violation can take place immediately after checking, and in case of update rejection, it consists of deleting temporary updates if checking was done before updates are committed, or of undoing already committed updates. Maintenance can also be performed some time after checking, for example, by the end of a user defined transaction, or even as prompted by the DBMS rather than indicated by the user (as suggested below). A major consideration regarding these choices concerns the propagation of faulty updates, and its control.

3. Directed Integrity Dependencies

We now introduce a notion which is useful for our revision of integrity transactions. When the update of a database variable is found to violate a value integrity constraint, there is essentially a choice of two maintenance actions: (0 the update is rejected, or (ii) the update is accepted, but the database variables related to the updated variable in the violated integrity constraint, must also be updated in order to re-establish the truth of the violated constraint The user who defines integrity transactions chooses the latter alternative.

A major difference between these two alternatives concerns the dependencies between database variables. In the first alternative, the value to assign to a variable is restricted by the values of the variables related to it by integrity constraints. In the second alternative, the values of the related variables are updated according to the (new) value of the updated variable.

These semantic dependencies between record values can be directed. For instance, if the new value is rejected, it means that it depends on the related ones, and has no influence on them. If the related values are updated, it means that they depend on the new one, and may or may not have some influence on it Records are linked by undirected dependencies when they depend pn each other. Directed dependencies are quite common in design. For example, structural loads determine the proportions of structural elements; electrical or fluid flows determine wire and pipe sizes respectively. But none of these relations are normally defined in the reverse direction..

In a given value integrity constraint, a variable whose value determines another variable value, is called *a parent* of this other variable in this constraint, and the determined variable is called a *child.* A child value can be partly, or completely bound, in which case it is redundant. (Redundant variables are the only ones which can, and should only be automatically updated). Two variables can be both parent and child of each other, in which case they are linked by an undirected dependency. Then, they have the same integrity status in that constraint. We shall say that a variable is a *strict* parent (child) of another variable if it is a parent (child) and not a child (parent) of that other variable. The values of the variables which are strict parents in a given constraint are not restricted in this constraint, i.e., the integrity status of these variables in this constraint is always true. In other words, the responsibility for the satisfaction of a constraint is impressed upon its children (which may or may not be strict children).

The notion of directed integrity dependency was mentioned in [14], and was used by one of the authors as a way to reduce and re-distribute in time, the cost of integrity checking by delaying the checking of updated variables'strict children [18], [19].

4. Global and Local Integrity

It should be observed that the integrity of the whole database does not necessarily have to be guaranteed before and after every transaction. As a matter of fact, this complete guarantee between transactions is generally impractical in design databases for several reasons.

The semantic integrity of a database for a design project is not complete until the end of the project. This is due to the very nature of design, where the coherent composition of many tightly related elements is the final goal. Most design decisions are made tentatively, without knowing whether all possible constraints are satisfied. Alternatives for parts of the projects are not fully evaluated, conflicts between constraints are not yet detected, or resolved.

This is amplified by the fact that parts of the database are protected so that only some users can access them. In other words, users have partial *views* of the database. This affects integrity management since integrity transactions are initially defined by users. Integrity constraints can cross views, and be hidden in some views. Several authors have developed ways of controlling updates through views so that other views, and the integrity of these views, are not affected by these updates [3], [8], [12]. Generally, this concern has no practical application in design databases. Designers' decisions often affect and override each other.. For example, it is quite common that a design element first designed or supplied for a given purpose, be also utilized later for other purposes. The later utilizations of this element are constrained by various aspects of the element set for its initial purpose. In terms of directed integrity dependencies, they are children of this element, and to the extent that they cannot affect it, they are its strict children. Now, it is conceivable that the initial designer be able to ignore these later utilizations. More generally, the users of a given record should not be forced to know the children of this record. For instance, the users of some design specification should be able to ignore, temporarily or permanently, the implementation of this specification. In other cases, data and integrity dependencies may not only be allowed, but forced to be ignored by protection policies. Therefore, a user cannot define integrity transactions that take care of the integrity of records related to the records of his/her view but unknown to him/her, especially if these records are children of his.

Thus, for most of the time it is. used, a design database not only lacks data, but the integrity of its existing data is not complete, because existing constraints are not evaluated or satisfied, or because constraints have not been stated. In the largest sense, the process of building a design database is a single integrity transaction by itself.

The lack of global integrity throughout most of design forces the introduction of the concept of *local integrity,* in terms of (sets of) individual variables and (sets of) individual constraints. Local integrity directly translates into integrity transactions. A transaction is re-defined as the execution of a collection of operations on a set of variables, during which the integrity of these variables, with respect to a set of constraints, is not guaranteed, but before and after which such integrity is either true or not applicable.

The locality of integrity is "specified by variables and constraints. We shall call these variables and constraints, the variables and constraints which *specify* the transaction. At one extreme, a transaction can be specified by a single variable and a single integrity constraint, and be called an *atomic* transaction. At the other extreme is the transaction which covers the whole database building process. Variables and constraints may be useful specifications of integrity transactions, not only to specify locality of integrity, but also because in general, the integrity of variables with respect to constraints is the goal of integrity management, and database operations are only means to achieve this goal. Several integrity transactions can be specified by the same variables and constraints, and consist of the execution of the same or of different operations at different times-

Locality of integrity can add flexibility to the scheduling of checking with respect to maintenance, and reciprocally. Different consequences of a given update can be checked at different times after the update. For example, Lafue proposed to check immediately a record update against the record's parents, but to delay the checking of the record's strict children until these children become the focus of attention *of* some application [18], [19]. Similarly, different maintenance updates following some checking could be performed at different times after the checking.

5. Nesting and Ordering of Transactions

Since an integrity transaction can be specified with as few as one variable and one constraint, it is to be expected that many transactions will occur before the final state of a database is reached. On the other hand, we saw that in design, the final database state marks the end of an overall integrity transaction. Both points are only possible if the integrity transaction over the whole design has nested within it other transactions. Nestings of integrity transactions can be several levels deep, with atomic transactions at the bottom. An integrity transaction incorporates the operations to check and maintain the integrity of its specifying variables with respect to its specifying constraints. These maintenance (and possibly checking) operations may have other transactions nested within them, since Ihe maintenance of one constraint may lead to the violation of others.

The transactions nested within another transaction may be logically connected in various ways:

- AND nesting. The most common case, its interpretation is that aH the nested transactions must be successfully completed before the outer transactions completes.

- XOR nesting. This form requires that exactly one transaction from a set must complete prior to the outer one. This logical connection is relevant when alternative mutually exclusive methods exist for design development, orwhen alternatives for a unique part of the database need to coexist temporarily before being thoroughly evaluated.

• OR nesting, when at least one of a set of transactions must succesfully complete.

The number of nested transactions which must successfully execute before an outer transaction completes, can be stated by an integrity constraint. This constraint specifies the outer transaction. Thus, integrity transaction nesting corresponds to integrity constraints nesting.

The extent to which nested integrity transactions can execute in parallel is another important aspect of integrity management It is often the case that a database update cannot (or should not) be executed before some pre-condition on the database is met. For example, in IMS, such pre-conditions regulate the manipulation of segments depending on whether they have a logical parent [16], or in the CODASYL model, they regulate the creation and deletion of records according to their roles in CODASYL sets and their type of set membership [7]. Such pre-conditions are based on existence integrity constraints typical of traditional data models. It is suggested here that pre-conditions based on more general existence integrity constraints than allowed in traditional data models, and on value integrity constraints involving directed dependencies, would be desirable supplements. For example, it could be that if bounds to the number of parents of a given variable are expressed in one or more integrity constraints, then the number of parents currently existing in the database should be within these bounds. Or it could be that the integrity of all or some of the currently existing parents be satisfied before the given record be allowed to be updated.

More generally, pre-conditions are a necessity for guaranteeing local integrity after transactions. If global integrity is assumed after each transaction, it becomes an implicit universal pre-condition for all transactions. Without global integrity, it is necessary to define the pre-conditions required for the checking and maintenance of the integrity specified in a transaction. Note that pre-conditions in terms of integrity dependencies are feasible only if integrity dependencies are directed. To the extent that they say that parents must exist and satisfy integrity before children, such pre-conditions fit intuition.

Since it restricts the order in which database operations can execute, the enforcement of pre-conditions may impose partial or total seriality between integrity transactions. It could also conceivably be amended with tolerance of temporary violations, and thus, add some sophistication and complexity to integrity management The enforcement of pre-conditions can be requested are a matter of policy which may be inherited from the DBMS, or at a higher conceptual level, fixed by the DBA (e.g., the project manager) in order, for example, to promote a particular design method. The enforcement itself can be the DBMS's responsibility if integrity constraints are known to the DBMS. In addition to interrupting a transaction because the pre-condition for an operation of this transaction is not currently met, it is conceivable that the DBMS suggest some way to resume the transaction. This could consist of notifying the applications authorized to satisfy the pre-condition (e.g., by creating, deleting or updating parent records) that their intervention is needed, and perhaps by interrupting these applications if they currently are active, or by activating them if they are not. This suggests a line for future research.

6. Confidence in Design Database Values

One further elaboration can be introduced. The integrity of a database variable with respect to some semantic integrity constraints is part of the confidence the designer/user can place in the value of this variable. However, semantic integrity measured at some point in time, is not the only factor in this confidence. It was observed earlier that design integrity constraints are not always explicitly part of a design database, nor can they always be expressed in a formal language. Here are two related observations.

The integrity of a database variable is defined with respect to the constraints which currently apply to this variable. We saw that creation and deletion of integrity constraints, as well as association and disassociation of variables and integrity constraints, were counted among the database operations which affect integrity. So, just because the integrity of a variable is currently true doesn't imply that the confidence in the value of this variable is total. This confidence depends also on how many integrity, constraints will eventually apply to that variable but have not been applied yet

It may also happen that a parent constrains a child, or more generally, an ancestor constrains a descendant, in such a way that no value is possible for the descendant. For example, it may happen that there is no possible (or known) implementation of a given specification, in which case, the specification must be changed. This happens often in design, where specifications come from normative knowledge and past experience, but the proof of their feasibility in a particular project is in their implementation.

If a variable is found to be infeasible, in that it imposes an empty set of values for one of its descendants, and consequently must be updated, then this variable can be seen as partly determined by its descendant, or rather, by the infeasibility of its descendant. But this descendant infeasibility may be difficult to express with an integrity assertion constraining this variable[4] . Convergence to consistent values for.the variable and its descendant may then require judgment and art from the designer. However, while the dependency of variables upon their descendants often cannot be expressed as explicit constraints, the confidence in the value of a variable can be increased as the confidence in the feasibility of its children increases.

These two observations are related to each other in that they both suggest that the confidence in design database values increases as the design approaches completion. The first observation is that the confidence in a variable increases with the number of constraints imposed on, and satisfied by, this variable. The limit of this increase (i.e., the number of constraints In the final state of the database) may be hard to estimate, and the currently missing constraints hard to predict Inversely, the second observation is that the confidence in a variable increases with the number-of variables it constrains in a feasible way. This number ceases to grow when no more descendants are created for some variable. Whether or not a given variable will be used by another one before the end of the design, may also be difficult to predict. Taking all this into consideration, the confidence in the contents of a design database is generally less than, and at most equal to the degree to which its current integrity is guaranteed.

7. Conclusion

Design databases have particular needs and problems regarding their semantic integrity management This is due to their high degree of dynamism, and to the nature of design. In general, designers working on a common database affect each other's design, want to tolerate temporary violations of their own or of others' integrity constraints, and add .to the definition of the database integrity applicable to their own or to others' views. Their integrity transactions reflect their limited views of the overall design. These transactions for local integrity must then be combined and scheduled. This can be (partly) automated, perhaps by some specialized DBMS component, based on the knowledge of all the integrity constraints defined on the database. This would require the integration of protection and integrity, so as to specify for whom particular integrity constraints must be checked, maintained, and/or visible. Such features of computer-aided integrity management are urgently needed if designing with a computer is to become as

[4]The infeasibility of a variable may also be due to the conjunction of two ancestor values, in which case, each ancestor may be seen as bound by the other. This kind of dependency may also be hard to express.

flexible as designing with paper and pencil.

8- References

1. AbrialJ.R. Data Semantics. In *Data Base Management,* Klimbie & Koffeman, Ed.,North Holland, 1974.

2. Astrahan M., Blasgen M., Chamberlin D., Eswaran K.,Gray J., Griffiths P., King W., Lorie R., McJones P., Mehl. J., Putzoiu G., Traiger I., Wade B., Watson V. "System R: A Relational Approach to Database Mangement" *ACM Transactions on Database Systems 1,*2 (June 1976).

3. Bancilhon F., Spyratos N. Update Semantics of Relational Views. INRIA, LeChesnay, France, 1978.

4. Bandurski A., Jefferson D. Enhancements to the DBTG Model for Computer-Aided Ship Design. Workshop on Databases for Interactive Design, ACM, 1975. Waterloo, Canada

5. Bernstein P., Blaustein B., Clarke E. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. Sixth Conference on Very Large Data Bases,, 1980.

6. Buneman O.P., demons E.K. -Efficiently Monitoring Relational Databases." *ACM Transactions on Database Systems* 4,3 (Sept 1979). "

7. *CODASYL Data Description Language Committee. DDL Journal of Development* 1976.

8. Dayal F., Bernstein P. On the Updability of relational views. Fourth Conference on Very Large Data Bases, 197a

9. Eastman. "General Purpose Building Description Systems." *CAD Journal 8,*1 (Jan 1976).

10. Eastman C. Systems Facilities for an Integrated Design Data Base. 17th Design Automation Conference, ACM/SIGDA, IEEE, 1980.

11. Eastman C. Recent Advances in Representation in the Science of Design. 18th Design Automation Conference, ACM/SIGDA, IEEE, 1981.

12. Furtado A., Sevcik K., Dos Santos C. "Permitting Updates through Views of Data Bases." *information Systems 4* (1979).

13. Furtado A., Dos Santos C, de Castilho J. "Dynamic Modelling of a Simple Existence Constraint" *information Systems 6* (1981).

14. Hammer M., McLeod D. A Framework for Data Base Semantic Integrity. Conference on Software Engineering, 1976.

15. Hammer M., Sarin S. Efficient Monitoring of Database Assertions. International Conference On Management of Data, ACM/SIGMOD, 1978.

16- *IBM Corporation. Information Management System/ Virtual Storage Utilities Reference Manual.* 1978.

17. Koenig S., Paige R. A Transformational Framework for the Automatic Control of Virtual Data. Seventh Conference on Very Large Data Bases,, 1981.

18. Lafue G. An Approach to Automatic Maintenance of Semantic Integrity in Design Databases. National Computer Conference, 1979.

19. Lafue G. *An Approach to Automatic Checking of Semantic Integrity in Design Databases.* Ph.D. Th. SUPA, Carnegie-Mellon University, 1979.

20. Stonebraker M. Implementation of Integrity Constraints and Views by Query Modification. International Conference On Management of Data, ACM/SIGMOD, 1975.

21. Tsubaki M. Multi-level Data Model in DPLS-database, dynamic program control and open-ended POL support. First Conference on Very Large Data Bases,, 1975.

22. WiederholdG. "Research in Knowledge Base Management Systems." *Sigmod Record 1*7,3 (April 1981).

23. Wisnowsky D. "ICAM: The Air Force's Integrated Computer Aided Manufacturing Program." *Astronautics and Aeronautics* (1977).

24. Zloof M. Security and Integrity within the Query-By-Example Data Base Mangement Language. IBM Thomas J. Watson Research Center, 1978.