CONSISTENCY AND TRANSACTIONS IN CAD DATABASE

Thomas Neumann, Christoph Hornung

Fachgebiet Graphisch-Interaktive Systeme
Fachbereich Informatik
Technische Hochschule Darmstadt
D-6100 Darmstadt, Germany

Abstract

A CAD database will reach a consistent state only at the end of the design process. This paper provides a framework for transforming the CAD database into the final consistent state. For this purpose we developed a model of a design object. Each design object can be composed of several representations. Some representations are independent of any other representations and some can be derived using other representations. Several versions of each representations can be stored simultaneously in the database. Using this model we derive a set of global consistency rules. The concept of transaction is refined to suit the CAD environment. A mechanism assisting in reaching the global consistency is described. A protocol controlling the concurrent access to the representation versions is given. Finally the consistency manager observing both the consistency and the concurrency protocol is outlined.

Introduction

Recently, research efforts have concentrated on possible applications of centralized database systems in CAD/CAM systems (Lorie81, Eastman80, Eberlein80, Lillehagen80). An integrated CAD/CAM system can be viewed as a chain of modules. Each module takes several files as input and produces several files as output. We can think of these files as being different representations of the same design. The original files are usually the product specifications, the budget for this product, the available parts, designs tools etc. The final output is usually a set of design and manufacturing documentations as well as a set of control files for numerical machines. Usually each module provides some data handling facilities. The format of file differs from module to module so that format translating facilities are needed if the modules should be connected together. Systems like these exhibit code redundancy, strong dependence of programs on the file organization and lack of facilities for consistency maintenance. With the design artifacts growing in complexity such as electronic circuits, VLSI, or aircrafts the amount of design data of all kinds grows rapidly. Therefore deficiency of systems which do not assist the designer in maintaining the consistency becomes apparent. In consequence research efforts have focused on the possibility of integrating database management systems into CAD/CAM systems. The existing database systems have been designed to handle business data. Business data can be characterized by small number of large files of formatted homogeneous records. A typical transaction against such database involves a small number of records and takes no more than a second. Sophisticated systems were designed which handle simultaneous execution of transactions and feature advanced recovery facilities. All these features were designed bearing business transactions in mind. Data and transactions in design applications differ significantly from those in commercial applications. First, the objects that are manipulated cannot be described conveniently using a small number of files of homogeneous records. The objects are more complex and have to be modelled by a collection of heterogeneous related records. There are only a few instances of such complex objects stored in the database. The database management system should provide functions for handling such complex objects (Lorie81). Second, we can distinguish between various representations of the same real world object. There can be a description of a logical circuit diagram. From that a layout description can be derived. The data management system should be aware of the connection between the logical description and layout. Traditional database systems do not assist the user in keeping different representations consistent. Third, the designer will want to experiment with different alternatives (versions) and will want to keep them in the database simultaneously until the design is complete. Again track of representations and versions should be kept by the system. Fourth the transactions i.e. the periods in which the database is inconsistent are in the order of days not seconds. Consequently, a different approach must be taken in designing the concurrency control scheme and in designing the recovery facilities. In traditional database systems the approach of aborting any incomplete transaction in case of system failure and resetting the system into consistent state has been adopted. This is clearly intolerable in a CAD system. The work done by the designer cannot be undone although he did not reach the consistent state yet. And fifth, the multiuser database systems supporting recovery features are usually too large to be integrated into one CAD/CAM system which should run on medium machines supporting a designer workstation. Instead a number of CAD workstations with simpler database management systems should be connected to a large machine hosting a large multiuser CAD data management system. Both the central and the workstation database management systems should be designed to support more complex data objects, storage of unformated data (think of a raster picture representing a background to a building being designed, or simply a textual description) should support multiple representations and their versions as well as the long lived CAD transaction.

We cannot of course address all of these issues in this paper. In the first section we shall make the notion of a design model more precise. Having done this, we shall state the consistency rules for such a model. Then we shall discuss some peculiarities of the transaction and suggest a locking protocol controlling the simultaneous access of design data. Finally we shall sketch a design of a CAD consistency Manager which combines the consistency rules and the locking protocols to maintain the database in a consistent state.

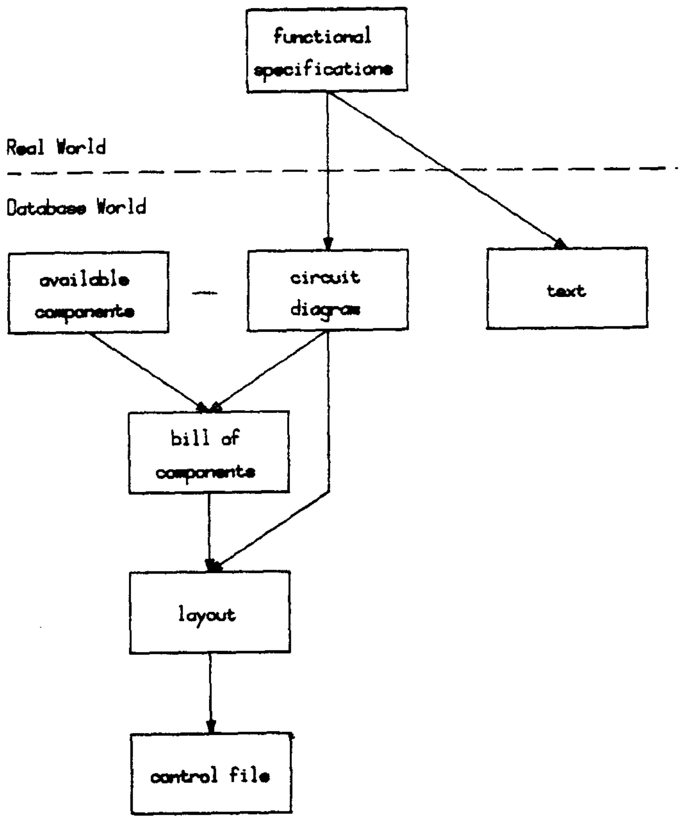


Fig. 1: Representations of an electronic circuit

The design object model

In this section we shall define what we understand under the design object. An object in the real world can be viewed from different angles and hence an object can have different database objects representing it. We shall call these database objects representations. Take for example a design of an electronic circuit. It will have several representations in the database, for example:

- circuit diagram
- description in form of text
- components list
- connections description
- layout diagram
- files for controlling the manufacturing tools

Some representations are independent of any other representations. We shall denote them as primary. In our example the circuit diagram is primary representation. Other representations are derived using the primary representations. They are called the secondary representations. In our example the layout may be derived using a connection list and a components list. Some representations may be structured as the circuit description, and some may be unstructured as a textual description describing the design (Fig. 1). The designer may want to experiment and design several circuit diagrams which meet functional specifications and store them in the database. For each of these diagrams he might develop several possible layouts using the same circuit diagram using the same component list. In this paper we shall call these alternatives versions. A group of versions derived from the same set of versions forms an equivalence class. So alternative layouts developed for one circuit diagram using the same component list forms an equivalent class of layouts with respect to this circuit diagram and to the corresponding component list. We shall call such a class a generation. In the following we shall formalize the notions of representation, generation and version.

Notation

Throughout the paper we shall use the following notation:

I will denote a set of non-negative integers
R_i will denote the representation i
R_I will denote a set of representations R_i where $i \in I$
G_{il} will denote the generation l of representation R_i

v_{ilt} will denote the version t of the generation l of the representation i

The relationships among the representations can be expressed in form of a directed acyclic graph (DAG) which we shall call the derivation graph. If there is an edge from R_I to R_j, then version of R_j can be derived using a version from each R_i where $i \in I$. The generation G_{jk} of R_j which was derived using the same set of versions of R_I forms an equivalence class. Versions are ordered with respect to time: $G = (v_{ikt1}; v_{ikt2}, \dots, v_{iktn})$. We further define a mechanism which maps a set of representations onto a single representation. To this end we define a set of functions $F=(f_{Ij}: \underset{i \in I}{X} R_i \rightarrow R_j)$. Each f_{Ij} takes as input versions from R_I and outputs a version from R_j. The set I is uniquely determined for each j by the derivation graph.

The derivation graph specifies the static dependencies among the representations of a design object. We further need a graph which captures the dynamic dependencies among the individual versions. For this purpose we define a dependency relation $dep(v_{ilt}, v_{jks})$. A tuple(v_{ilt}, v_{jks}) is in the dependency relation dep if and only if v_{jks} was derived by function f_{Ij} using v_{ilt}. The DAG defined by this relation will be referred to as the version graph.

A design scheme is a two tuple (R,DG) where R is a set of representations belonging to this design and DG is the corresponding derivation graph.

A design object is a two tuple (V,VG) where V is a set of versions and VG is the corresponding version graph. Such an object is an instance of the design scheme.

The consistency model

In the preceding section we have given a formal definition of a design object. In this section we state what constitutes a consistent design object.

We can distinguish between local and global consistency. Other researchers have observed the difference (Eastman 80). Each (design) version may be composed of several database entities. A set of predicates may exist on database entities. These predicates are called the consistency constraints (Gray 80). We shall call these consistency constraints the local consistency constraints. In our example such constraints may specify the least distance between two conductors within a layout version. Here we want to address the global consistency problem. We shall define the global consistency constraints to be a set of predicates associated with the set of representations. In our example global consistency constraints specify the dependencies between the circuit diagram and its corresponding layout. In the following we shall make the notion of the global consistency more precise.

Definition of Global consistency

A version v_{jkt} of representation R_j is globally consistent if and only if it can be derived from the corresponding set of representations R_I as defined by the derivation graph.

An object O=(V,VG) is globally consistent if and only if each version v_{jkt} of each representation R_j is globally consistent.

By supergraph of a version we will understand the set of all predecessor versions (direct and indirect) in the version graph.

Similarly, by subgraph of a version we will understand the subset of all successor versions (direct and indirect).

To state the consistency rules we define the notion of dependent and independent inconsistency:

A version is called independently inconsistent if the supergraph of this version is globally consistent and the version itself is globally inconsistent.

The subgraph of an independently inconsistent version is called dependently inconsistent.

To illustrate the meaning of these definitions consider the following example. Suppose the designer decides to modify the existing version of the circuit diagram. Then we can mark all layouts belonging to this diagram as dependently inconsistent. It will be pointless to do any work on these layouts until the circuit diagram becomes consistent again (consistent with the funcional specifications).

In the following we shall refer to the global consistency as consistency unless specified otherwise.

Assume that each time a version v_{jkt} is created from R_I using the function f_{Ij} and observing the derivation graph, a tuple is inserted into the relation dep. To add the consistency information we can extend the version graph to a labeled graph. The label can assume one of the following values 'consistent', 'independently inconsistent', 'dependently inconsistent'. When a version v is read with the intention to modify it, its subgraph is marked as dependently inconsistent. After it has been modified and is presumably consistent, its direct successors will be marked as independently inconsistent.

Then we can say that the object O(V,VG) is consistent if and only if for each version there is a tuple in the dependency relation and all labels of the version graph are marked as 'consistent'.

Following the consistency definition the following consistency rules can be inferred:

<u>Rule 1</u> A version v_{jkt} of a generation G_{jk} can be <u>updated</u> if and only if it is consistent or independently inconsistent.

<u>Rule 2</u> A version v_{jkt} of G_{jk} can be <u>created</u> using only consistent versions.

<u>Rule 3</u> Any version can be <u>deleted.</u> If a version is deleted then its subgraph must be deleted.

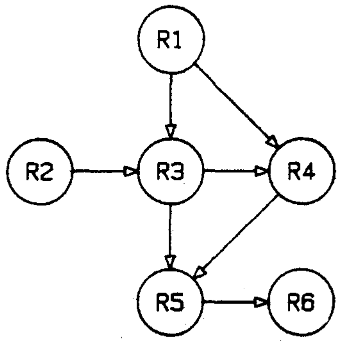
<u>Rule 4</u> Any version can be <u>read.</u>

An example of CAD object model

As an example, consider a design of a logical circuit (Fig 2). From the functional specifications several circuit diagrams may be developed. Here we consider the circuit diagram to be an independent representation. That is, the system will allow any modification on circuit diagram to be performed. The validation of the specifications is left up to the designer. From the logical circuit diagram other representations can be derived using the functions f_{Ij}. Let R_1 denote the circuit diagram and R_2 the list of available components. Further let R_3 be the list of components with which the circuit can be realized. R_4 should denote the connections description, R_5 the layout and R_6 the set of alternative control files for the numerically controlled tools. Then we can define the following mappings:

$(R_1,R_2) \rightarrow R_3$	circuit diagram and the available list, define the bill of components
$(R_1,R_3) \rightarrow R_4$	circuit diagram and the bill of components define the list of connections
$(R_3,R_4) \rightarrow R_5$	the bill of components and the connections list define the layout
$R_5 \rightarrow R_6$	layout defines the control file

The representation graph is shown below :
R_1 and R_2 are independent variables and can be changed at will. All other representations must be modified accordingly to keep the design con-



sistent.

The CAD transaction

The transaction concept as defined for classical systems is not well suited for CAD database systems. The transaction concept has been introduced to cope with the consistency issues in view of concurrently executing programs and in view of failures of software and hardware. We assume that the reader is familiar with these concepts and shall only briefly recapitulate the fundamentals (Gray80). The database can be seen as a collection of entities. Each entity can assume a value. The system provides operations, each of which manipulates one or more entities. Associated with the database are consistency constraints. The collection of all values in the database define a database state. A database state, which satisfies the consistency constraints is said to be consistent. A transaction can then be defined as a sequence of database actions which transfers the database from a consistent database state to a new consistent database state. In process of transforming the state, the database may be temporarily in an inconsistent state. The fundamental difference between the conventional and CAD database is the duration of inconsistent periods. A conventional database is inconsistent only during a run of a transaction, which usually is in the order of seconds. In contrast, the CAD database becomes consistent only at the <u>end of the design process.</u> Most of the time it is in inconsistent states. Therefore CAD transaction cannot be defined as a sequence of actions transforming the database into a new consistent state. We have distinguished between local and global consistency. The CAD database can be viewed as a collection of versions of object representations. With each representation there is associated a set of consistency constraints which define the local consistency. The relationships among representations define the global consistency. We say that, the CAD database is locally consistent, if each representation version is locally consistent. Conversely, the CAD database is said to be locally inconsistent, if any representation version is locally inconsistent. Then we can define the CAD transaction as the sequence of actions which transforms the CAD database from a locally consistent state to a new locally consistent state. The set of legal locally consistent states is however determined by the global consistency rules. The consistency constraits among representations are usually too complex to be observed automatically. Therefore, we propose a half-automatic approach to reach the final global consistency. The CAD transaction as defined in our paper is still much longer than a conventional transaction. In general, the time during which the CAD database is locally inconsistent is in order of days. Hence, it must span several design sessions and survive system restart. This kind of transaction has been suggested by other researchers (Lorie81, Gray81). The conventional transaction is both the concurrency and the recovery unit. Each transaction sees only consistent database state as its input. This is usually achieved by locking the entities which the transaction intends to manipulate. Being the recovery unit means, that all actions forming a transaction must occur or none may occur. The consequence of this is, that in case of a system failure only the results of commited (normally) ended transactions persist. The updates caused by uncommited transactions are undone. We want to keep the idea of the transaction being the concurrency unit. That is, we can have concurrent transactions transforming different representations or transactions transforming different versions of a representation. Each CAD transaction is allowed to see only locally consistent state. As we said earlier, we expect the system to assist us in reaching the globally consistent state. That implies, that not any two versions belonging to different representations may be modified concurrently. After all, there are global consistency constraints we hope to satisfy. The CAD transaction cannot serve as a recovery unit for an obvious reason: a great amount of work could be lost. In case of a failure some action consistent state must be restored. In this paper we shall not explore the issues of recovery any further. At the end of a conventional transaction the updates are validated against the consistency constraints. Only transactions, updates of which do not violate the consistency constraints are allowed to commit. In a CAD database two sets of consistency constraints must be checked:local and global. Again, the practice of aborting transactions violating the local or global consistency constraints is untenable. Global consistency validation might be impossible to perform automatically without designer´s intervention. Therefore, what we propose here, is that the system keep track of versions which may have become globally inconsistent due to current transaction´s updates and enforce the global consistency rules 1 to 4. The designer is given a chance to either force commit i.e. to declare the version as both locally and globally consistent or to perform additional update. We shall not discuss the consistency validation mechanisms here,which are application dependent. We assume that such a mechanism exists and that once the transaction has been allowed to commit the stored version is globally consistent. We shall examine how the database management system can assist the designer in maintaining the global consistency. As we mentioned in the introduction, we shall assume that there is a central depository of all design data. Several workstation databases are connected to the central database. Note that the workstation database need not reside on a satellite computer. It can represent a private portion of the central database. Versions can be extracted from the central database into the workstation database at the beginning of a CAD transaction. In conventional systems which handle short transactions, a transaction might be temporarily suspended if a lock cannot be granted immediately. CAD transactions cannot wait for locks as they could be waiting for a very long time. We see two solutions to this problem. One would be to force each transaction to preclaim all the objects it is going to use. If any of the objects cannot be retrieved the transaction will be aborted. This does not matter, as no work has been done yet anyway. The other possibility is to allow the designer to ask for new objects during the transaction execution. If the requested object cannot be granted the designer will be told so. He can then decide if he wants to adjourn the session or continue in spite of it.

The global consistency mechanism

The mechanism described here assists the user in reaching the ultimate design goal: the consistent database. In particular it prevents construction of new versions from inconsistent versions. We can view this mechanism as directing the design activities. The stored versions can be extracted in one of the following modes:

- consistent read mode
- read mode
- update mode
- derivation mode

In consistent read mode only consistent or independently inconsistent versions can be extracted. In read mode any version can be extracted. When a version is extracted in the update mode the old version will be replaced by a new updated version. The derivation mode tells the system that a new version will be constructed using this version as a source. The versions in the central database are globally consistent independently inconsistent or dependently inconsistent. They are assumed to be locally consistent. In order to obey the consistency rules transaction can extract a version v for update only if the version is consistent or independently inconsistent. The subgraph of so extracted version will be marked as dependently inconsistent. This implies, that a transaction cannot extract a version which may have become inconsistent due to its own updates on higher level in version graph. The rationale here is that it does not make sense trying to update a verson, ancestors of which are unstable (being modified). When the transaction commits, it will have returned a globally consistent version into the central database. The immediate successors will now be marked as independently inconsistent, so that they can be extracted for update and made globally consistent. In this way the whole subgraph and eventually the whole database can be made globally consistent. A transaction can extract a version in the derivation mode only if it is globally consistent. Again it does not make sense to create new versions using an inconsistent source version. Dependently inconsistent version can be extracted for examination only (in read mode).

The dependent or independent inconsistency can be checked using the version graph.

Synchronization of concurrent CAD transactions

In the real environment there will be several CAD workstations attached to the central host. Even with only one workstation multiple CAD transactions can be running simultaneously, since transactions at the workstation can be suspended and restarted at a later time. Hence a mechanism must be provided to synchronize concurrent CAD transactions.

The synchronizing mechanism should:

1 - prevent two transactions from updating the same version
2 - prevent a transaction from modifying a version when any of its descendants in the version graph are being modified
3 - prevent a transaction from modifying a version when any of its descendants in the version graph are being used by an application to create a version of a different representation
4 - prevent a transaction from modifying a version whose one or more predecessors are being modified
5 - prevent a transaction from modifying a version while it is being inspected by other transaction
6 - prevent deadlocks

We shall refer to these goals as concurrency rules. The above rules may be relaxed to allow reading of versions which have been extracted for update. This will be useful in the environment of long CAD transactions.

All the above points are obvious. The deadlocks must be prevented, since there is no way how they can be resolved. Transactions cannot be rolled back as it is practice in conventional database systems because of their long duration. The deadlocks are automatically prevented if the above mentioned practice of not suspending transactions or by forcing each transaction to preclaim all versions it is going to use is adopted.

In this paper we shall ignore the problems of mixing the short (traditional) and CAD transactions. For simplicity we shall assume that all transactions are CAD transactions and that the only lockable granule is a representation version.

We introduce two lock modes: update or <u>u-mode</u> and derivation or <u>d-mode.</u> The u-lock is an exclusive lock. The <u>d-lock</u> will serve three purposes:

- it will be used to lock a version extracted as a source for derivation
- it will signal that a locking is being done at a lower level on the version graph
- it will be used to lock a version extracted for read if desired

The d- and u-lock modes are <u>incompatible.</u>

The CAD transaction is well behaved if it observes the following locking protocol.

- if a transaction wishes to extract a version for update it must lock <u>all</u> the paths leading to it in d-mode and and the version itself in u-mode

- if a transaction wants to create a new version of a representation it must lock for each source version <u>all</u> path leading to it in d-mode

- if a transaction wants to prevent other transactions from modifying a version or its supergraph while it is inspecting this version it must lock <u>all</u> path leading to it in d-mode and the the version itself in d-mode

- locks can be released only at the end of the transaction and so they can released in any order

The reasons for the releasing the locks at the end of the transactions only can be read in (Gray75) and will not be discussed here.

It can be easily shown, that if a transaction is well behaved, it preserves the concurrency rules:

- Suppose transaction T acquired a u-lock for a node n in the version graph. Then according to the locking protocol all path leading to this node must have been locked in d-mode by T. Hence no node lying on the path to the node n could have been locked in u-mode by any other transaction since u and d modes are incompatible. Hence goals 1, 2 and 4 of concurrency rules are satisfied.

- Suppose transaction T extracted a version v in derivation mode. Then according to the locking protocol all path leading to this version and the version itself must have been locked in d-mode. Hence no version in v´s supergraph can be locked in u-mode since u and d modes are incompatible. Hence item 3 of concurrency rules is met.

- Similarly, if a version v has been extracted for inspection and locked then all path leading to it and the version itself must have been locked in d-mode. Hence no version in v´s supergraph or the version itself can be locked in u-mode. However, in view of longevity of CAD transactions, locking a version which is being extracted for read will seldom be done. More often, a version will be extracted without acquiring a lock. Such transaction could see an old version while new version is being prepared.

The Consistency Manager

The Consistency Manager assists the designer in maintaining the global consistency of the design object. It does that by observing both the locking and the consistency protocols. The locks pertain to the CAD transactionsand do not persist beyond the transaction termination. In contrast the version graph exists independently of transactions and must persist beyond the transaction completion. Transactions submit lock requests to the Consistency Manager. The Consistency Manager can grant a lock request only if there is no conflict due to concurrency protocol and due to consistency rules. If a transaction needs a version in update mode the consistency manager grants permission if

- the version is consistent or independently inconsistent
- its supergraph can be locked in d-mode and the version itself in u-mode

After the version has been extracted the corresponding subgraph is labeled as ´dependently inconsistent ´. This ensures that no other transaction can extract a subgraph version for update. When the transaction commits its updates, the immediate successors of the updated version are labeled ´independently inconsistent´ and the lock is released. Now the immediate successors may be updated and made globally consistent. In this manner the entire database can be step by step transferred to a globally consistent state. The database will normally become consistent only when the design is completed.

If a version is needed in derivation mode

- it must be consistent
- a d-lock must be acquired for its supergraph and for the version itself

If a version is needed in consistent read mode with lock the consistency manager grants permission if:

- the version is consistent or independently inconsistent
- the supergraph and the version itself can be locked in d-mode

Consistency Manager does not suspend transactions, but rather returns a message if the requested lock cannot be granted. The rationale is that the transaction could be suspended for a long periods of time (days).

The locks managed by the Consistency Manager differ from conventional locks in that they are recorded in nonvolatile storage as they must span sessions and system failures. Such lock locks are called permanent locks (permanent for the duration of a transaction, of course). If the system is to handle both the conventional (short) and the CAD (long) transactions the conventional Lock Manager and the Consistency Manager must be aware of each others locks. An efficient solution to this problem remains to be found.

Conclusions

We believe that the above discussion helps to clarify the issues of consistency in CAD databases. We realize, that it might be difficult if not impossible to build a system which will check global consistency of different versions automatically. The primary reason is the dependency of such mechanisms on the particular application, and their high complexity. Further reasearch is needed to clarify these issues. However, we strongly believe that a CAD transaction manager based on ideas presented above would be of great assistence in keeping the CAD database consistent and will make the database management systems more attractive in this application area.

References

Eastman81:
Charles M. Eastman, Gilles M.E. Lafue
Semantic Integrity Transactions in Design Databases
Proceedings of the IFIP Working Conference on CAD Databases,
to be published by North Holland Publishing Company, 1982

Gray75:
J.N. Gray, R.A. Lorie, G.R. Putzolu, I.L. Traiger
Granularity of Locks and Degrees of Consistency
IBM Research Report RJ1654, 1975

Gray80: Jim Gray
A Transaction Model,
IBM Research Report RJ2895, 1980

Gray81: Jim Gray
The Transaction Concept: Virtues and Limitations
Proceeding of the Conference on Very Large Data Bases, 1981

Lillehagen81: Frank M. Lillehagen
Towards a Methodology for Constructing Product Modelling Databases in CAD
Proceedings of the IFIP Working Conference on CAD Databases,
to be published by North Holland Publishing Company, 1982

Lorie81: Raymond A. Lorie:
Issues in Databases for Design Applications
Proceedings of the IFIP Working Conference on CAD Databases,
to be published by North Holland Publishing Company, 1982

Eberlein81:
Werner Eberlein, Hartmut Wedekind
A Methodology for Embedding Design Databases into Integrated Engineering Systems
Proceedings of the IFIP Working Conference on CAD Databases,
to be published by North Holland Publishing Company, 1982