Data structures and DBMS for CAD Systems- A review

Mehak Sharma*
Department of Computer and IT
C.T Institute of Management & Technology, Jalandhar, India

Manikant Sharma
Department of Computer Science
Lovely Professional University, Phagwara, India

Abstract **- The structures for the storage of data in CAD systems influence to a large extent the effectiveness of the system. This paper reviews the wide range of data structures and database management systems (DBMS) available for structuring CAD data. The relationship between these basic data types, their composite structures and the classical data models (on which many DBMS are based) is discussed, and the limitations of existing DBMS in modeling CAD data highlighted. This paper also outlines the historical development of data management systems in order to identify the key issues for successful systems. It identifies the need for data independence and the embedding of structural and behavioural semantics in the database as key issues in the development of modern systems. Hierarchical, Network, Relational, Object-oriented and Object-relational data management systems are reviewed. A short summary of related research is given. The paper concludes with some speculation on the future directions that database technology might take.**

Keywords**: Data structures; Database management systems; Computer aided design**

I. INTRODUCTION

We all know that there is a discipline which we call software engineering, it has to be the case for there are a sufficient number of textbooks available with the phrase appearing prominently in the title. Many worthy academic institutions have chairs of software engineering and there are numerous international conferences, workshops, symposiums and the like dedicated to the exploration of sub-areas of software engineering. It is not as clear that there exists a similar discipline called data engineering. The disciplines of software engineering and data engineering are similar but have different emphases and historical roots. The starting point for a software engineer is a task that must be carried out on a computer. A data engineer most often begins with a task that exists already either as a paper-based system or in some computerised form and seeks to engineer a better solution. A software engineer says that a program is made up of data and algorithms and generally means transient (main memory) data. A data engineer says that applications are constructed to run on top of data and means permanent (secondary storage) data. A software engineer designs systems, a data engineer constructs a basis upon which systems may be built. This said, the similarity between data engineering and software engineering is probably greater than the difference. Both disciplines attempt to encourage principles and practices that enable developers to speedily construct systems which match their specifications and can be demonstrated to function correctly. The two disciplines are about engineering solutions to similar problems. Both disciplines attempt to extract essential semantics from a real world situation and preserve them in an application. Software engineering tends to seek for ways of encoding these semantics in code whilst data engineering embeds them in metadata.

Data structures refer to the method of organization of data within a database or a computer program. A more formal definition is provided by Stubbs & Webre' who define a data structure as 'a data type whose values are composed of component elements that are related by some structure'. Data structures can also be regarded as the combination of brick, mortar and glue that hold databases together. The component data elements of a data structure could be either atomic (i.e. non-decomposable) or data structures themselves. The relationships between these component data elements constitute the structure and have implications for the functioning of the data structure. They condition the set of operations which act upon the data structure and its component elements, as well as the efficiency with which they perform. It is also the nature of these relationships that differentiates one type of data structure from another.

II. ORIGINATION OF DATA STORAGE SYSTEMS

The invention of magnetic storage media such as magnetic tape and magnetic disks enabled the permanent storage of large quantities of data in a manner that made them amenable to computer processing. The term 'large' is not used in absolute sense it is simply an indication that storing punched card or paper tape representations of data was never a realistic option for many potential data processing applications. A number of business-related uses of computers came into being as a direct result of this development. Typically these relied on 'batch' operations. Stored records were kept on master files. Over a period of time a set of transactions or operations were collected and at

an appropriate time run against a master file. The master file and the transaction file were sorted in the same order on some key. At regular intervals the transactions were applied to the master file and a new updated master file was produced. At the same time a report indicating the success or the failure of each transaction was generated.

This scenario contained a number of inadequacies. Firstly this type of system made no attempt to describe the data it held. The only assistance a programmer could hope to receive from underlying software was that the operating system could find the file. Once the file had been located on the disk it was the programmer's task to handle the file as a contiguous piece of permanent storage. There was no indication given as to whether the bytes read were represent-ing single characters, character strings or numbers. It was the programmer's task to add the semantics of the application to the stream of data retrieved from the disk. Eventually programming language support was provided to make this task easier, however, such support was limited to aiding an individual programmer and not everyone (including non-programmers) who needed to access the data. It was possible for two programmers to describe the same data in two different ways and hence apply different semantics to it. What semantics were made available in a programming language were limited to a simple description of the way in which the data might be displayed and did not describe the operations and constraints that were appropriate to it.

Secondly, it was quickly recognized that this pattern of processing was repeated time and time again. The central logic of each program was identical, all that altered was the details of the input and output operations. Despite this, each program was handcrafted each time. This did not improve productivity nor did it ensure that a solution known to be correct was applied consistently. Thirdly, the idea of a file of data in isolation did not correspond with the way data was known to behave in application areas. A computer file corresponded to what one might expect from a manual filing cabinet. It was a bringing together of a number of fixed format pieces of information called records. Records consisted of a number of fields that held individual pieces of information. The records in a file were normally sorted in some order to allow speedy processing and retrieval. It was known, however, that many applications relied on an ability to retrieve records based on their relationships to records in other files. More than this, the validity of entries in some records depended on entries found in other files. Implementing systems that embodied these semantics was possible but involved the construction of quite complex programs that were difficult to maintain.

III. TYPES OF DATA STRUCTURES

Data structures are classified according to the relationships between their component data elements, as well as their definition. Most data structures can be broadly classified as static or dynamic structures as shown in Figure 1.

3.1 Static data structures

Static data structures are by definition fixed in size and structure throughout their lifetime. They are thus restricted and unsuitable for modeling dynamic situations. Typical examples of static data structures are arrays, records and sets.

3.1.l Arrays

An array consists of a set of data elements which is denoted by a single identifier or variable name. The component data elements are all of the same type and can each be accessed using an index or subscript which refers to the position of the particular data element within the array. The number of elements in an array is often predefined; an overestimation of the array size leads to memory wastage whilst an underestimation means that not all data elements can be accommodated.

3.1.2 Records

Records differ from arrays in the heterogeneity of their components. This means that a record structure can contain elements of different types. For example, a book record can be declared as follows:

```
VAR Book: RECORD
Title: ARRAY [0 . . 100] OF CHAR
pages: CARDINAL
Year: INTEGER
END
```

The components of a record are called fields and are named. The variable, Book, can have a value assigned to any of its fields as follows:

Book. Title := 'The Psychological Profile of Goalkeepers'

Records are regarded as static structures even though their components (fields) could sometimes have a dynamic structure (e.g. a list of the book's chapter headings). This is because the basic structure remains static and it is not possible, for example, to introduce a new record field.

3.1.3 Sets

Sets are collections of data elements which are similar to arrays in being homogeneous. However, the elements of the structure are related only by their membership in the set. MODULA-2 sets have two restrictions - they must contain constants only and the base type of a set must be an enumeration or sub-range type.

3.2 Dynamic data structures

Unlike static data structures, dynamic data structures do not have a fixed size; they can grow and shrink as required and can represent dynamic real world situations (such as a queue). Dynamic data structures provide for more efficient memory management as memory is allocated only when needed and freed memory locations can be reused. Linked lists and trees are examples of dynamic data structures. An important tool in the construction of these data structures is the pointer - a data type whose values are the locations of the values of other data types.' The component data elements of dynamic data structures (also called nodes)

store pointers to one or more other data elements in addition to their own values; these pointers establish the relationships between these nodes.

3.2.1 Linked lists

Linked lists consist of nodes each of which contains a pointer to the next node in the list. A list in which the nodes store pointers to both their successor and predecessor nodes is called a doubly linked list and has the advantage that traversal of the structure can be bi-directional, as shown in Figure 2. Stacks and queues can be represented by linked lists. A stack exhibits a last-in-first-out (LIFO) protocol; all insertions and deletions are made at the same end of the data structure. In contrast, a queue has insertions and deletions occurring at opposite ends of the data structure - a fist-in-first-out (FIFO) protocol. Linked lists may be ordered with respect to a given key; insertions and deletions are controlled by that key and can take place at any point within the structure such that the order of the list is maintained.

3.2.2 Trees

Trees, unlike lists, are nonlinear data structures. However, like lists, they are capable of recursive definition (i.e. being defined in terms of themselves). A tree is either empty or consists of a unique node called the root node together with any number of subtrees. Each node (except the root) has a unique 'parent' and each non-terminal node has one or more 'children'. A tree is a natural structure for keeping track of information that has a one-to-many (or hierarchical, or nested) relationship among its elements. Figure 3 illustrates a general tree structure. Variants of this general structure include binary trees (in which each node has at most two children) and B-trees (balanced trees in which all terminal nodes are the same distance from the root).

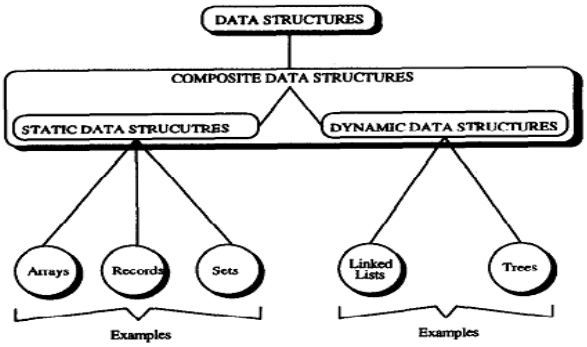
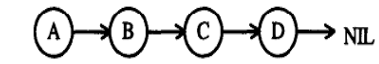


Figure 1 Types of data structures.



(a) Singly linked list

(b) Doubly linked list

Figure 2 Linked lists.

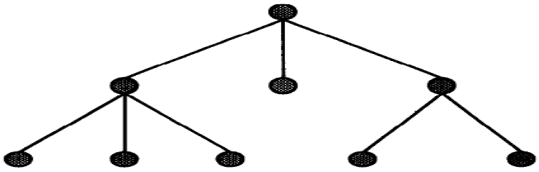


Figure 3 General tree structure.

3.3 Composite data structures

A composite data structure results when a data structure is made up of a combination of static and/or dynamic data structures. The use of composite data structures offers the software developer a great deal of flexibility in data structuring. Useful composite data structures include arrays of records (e.g. a table) and arrays of pointers. Many computer programs and DBMS (presented in the next section) make use of composite data structures of one form or another.

IV. DATABASE MANAGEMENT SYSTEMS (DBMS)

A DBMS is a computer software system consisting of a database structure and programs to manage the database. It usually provides facilities for the organization, access and control of the database. Database management systems are widely used commercially and are classified according to the type of data model employed as shown in Figure 4). The three most popular data models are the hierarchical data model, the network data model and the relational data model. A relatively new data model, the object-oriented data model (OODM), has also been discussed.

4.1 Hierarchical systems

In the late 1960s, magnetic tape was still a major medium for data storage. Tape does not have the addressing flexibility of the magnetic disk and therefore a data model that supported sequential access was necessary for this type of storage. This requirement led to the development of the hierarchical model of data implemented in IBM's database product: Information Management System (IMS). Any hierarchy of records can be represented as a sequence and such a sequence can be stored on magnetic tape. The first major data model came into being purely out of consideration for the underlying physical storage it had to work on. The original use intended for IMS was "bill of materials processing" and the data model chosen was ideal for this purpose.

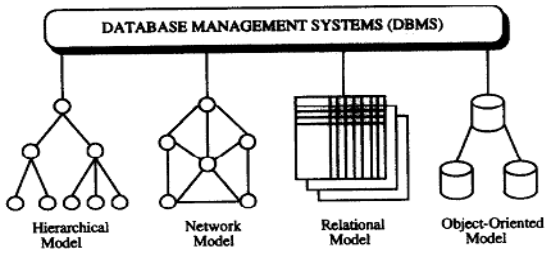


Figure 4 DBMS data models.

This type of application deals with facts such as "Part A is constructed from Parts B and C, Part B is constructed from Parts D, E and F". This is a natural hierarchy (tree) and is easily mapped to the IMS data model. More complex scenarios required extensions to the original model so that data whose relationships could not be represented by a single tree could efficiently stored as a collection of trees. IMS did not capture the semantics of the data it stored beyond being able to represent relationships between records. Individual fields were not identified by the database management system; a record was defined simply as a number of bytes into which data could be placed. As a consequence it was unable to support ad hoc queries. The processing semantics were entirely embedded within the programs written for applications and it was necessary to write programs in order to access the database.

4.2 The network model

The network data model is a generalization of the hierarchical data model. It represents data as a set of record types and pair-wise relationships between record types. It does not observe the single-parent rule of the hierarchical data model; a child record can have any number of parents. This enables the network model to represent arbitrary relationships (including non-hierarchical ones) amongst entities. The complexity of the network data model is a major disadvantage. Any increase in the number of records means an increase in the number of pointers used to establish the relationships between the records. This has adverse maintenance implications and also makes network models difficult to use. Another disadvantage is that the basic structure of network of associations has to be pre-established so that the pointers can be set up and maintained as new records are added; this is unsuitable for design.

Several examples of network data models are available; these are usually referred to as CODASYL (Conference on Data Systems Languages) systems. Notable amongst them is IDMS (Integrated Database Management System). Figure 5 shows logical diagrams of relationships in both the hierarchical and the network models. In the hierarchy, a record of type A may be related to many records of type B and many records of type C. This is all that is permitted. In the network diagram a record of type D may be related to many records of type F and also to many records of type G. This could be represented in the hierarchical model. However, the diagram also indicates that a record of type E may be related to many records of type G. These additional relationships would not be permitted in a pure hierarchical model. Given these two one-to-many relationships it is possible to construct many-to-many relationships between records of type D and records of type E (i.e. a record of type D may be related to many records of type E and vice versa).

4.3 The relational model

In the relational data model, tables (or relations) are used to represent data. Each relation is a two-dimensional table consisting of a fixed set of attributes (or columns) and a time-varying set of tuples (or rows).

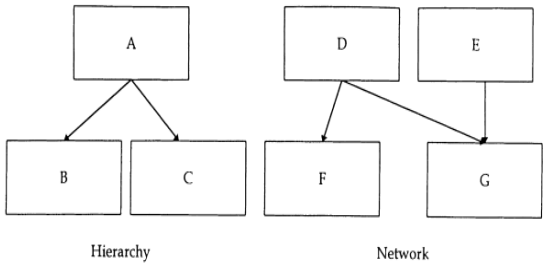


Figure 5 Comparison of hierarchy and network.

Tuples are equivalent to record instances and the associations between them are determined not by pointers or computer memory addresses, but on the basis of identical contents in the attribute fields of different tuples. Columns are named and contain values of the same type whilst rows are distinct, have a unique identification key or primary key (i.e. no duplicate tuples allowed) and are not ordered. The relational data model is rich in its ability to represent directly a wide variety of relationship types' and does not require any pre-definition of physical access paths to represent associations between different records. On the other hand, it has limited capabilities for data abstraction and its inherent redundancy makes normalization operations necessary.

4.4 The object-oriented data model (OODM)

This is a relatively new data model for DBMS. Object-oriented systems are believed to have their roots in programming languages such as SIMULA and SMALLTALK" and have the following characteristics: abstraction of data, inheritance of properties and encapsulation of data and operations." In OODM, all conceptual entities (and relationships) are modeled as objects; an integer or string is as much an object as is a complex assembly of parts. Objects are grouped into classes with lower level objects (sub-classes) inheriting the properties of their parent classes (superclasses). Thus, object classes are organised into a simple hierarchy. In many systems, however, a class can have more than one superclass, thereby generalising the hierarchy into a lattice (or network) structure.

An object consists of a private memory (which holds its state and is made up of the values for a collection of instance variables) and a public interface (which is made up of messages and their associated arguments, through which objects communicate with one another). OODM have the potential to represent complicated relationships and support object versions and transaction management. They are seen as a potentially useful model for many engineering applications. Many experimental models have been developed and it will not be too long before OODM achieve the same level of success as the other data models discussed earlier.

4.5 Semantic models

The separation of storage concerns from mechanisms for representing real world information found in the

ANSI/SPARC architecture and the relational model allowed a number of researchers to concentrate on so called "semantic" models. The purpose of these models was not necessarily to produce something that could be immediately implemented but rather to provide a mechanism through which the structural aspects of a real world situation could be captured. The simplest of these models was the entity relationship model [1]. This offered little more in the way of semantics than the relational model, however, through its diagramming technique it provided a means by which a database designer could present an overview of the essential aspects of a database schema. The entity relationship model was later enlarged to allow the expression of data semantics, which cannot be directly, represented using a relational database [2]. Hammer and Mcleod's semantic database model (SDM) [3] was a particularly rich example of this type of model. Whilst these models were not incorporated into widely available database products, they helped to demonstrate the limitations of the relational model. Kent [4, 5] also addressed these limitations directly in a paper and an influential book. The shortcomings that were identified did not single out classes of applications which could not be constructed using relational technology instead they highlighted a case where important data semantics would reside in the application programs and not in the database. The semantic models clearly demonstrated that classification was an essential mechanism for capturing the full structural semantics of an application and this fact increased the interest generated by object-oriented database systems.

V. OBJECT-RELATIONAL SYSTEMS

The publication of the object-oriented database manifesto [6] led to an almost immediate response from those interested in the further development of relational systems [7]. Proponents of the extension of relational technology argue that object databases have been evolved primarily to support programming language interaction with data and that this requires very different techniques than those required to support a query language. Consequently whilst object databases are efficient for supporting complex data they cannot efficiently support query languages, especially query languages which allow updates. The object-relational camp identifies the major weakness of conventional relational systems as an inability to support complex data. The solution proposed is the addition of facilities for handling such data on top of existing SQL facilities.

Stonebraker [8] argues that this requires the addition of the following four features:

(i) Support for base type extensions in an SQL context
(ii) Support for complex objects in an SQL context
(iii) Support for inheritance in an SQL context
(iv) Support for a production rule system.

Base types in traditional relational systems usually include character, string, integer, fixed point, floating point, date and time. A base type is one that cannot be decomposed into any further fields. Object-relational systems allow the database designer to define new base types. Such a definition will involve the construction of code that defines basic operations on these new types. Once this has been written the new types can be incorporated into SQL queries in exactly the same manner as the built-in types. To be able to make full use of extensible base types the system must also allow the construction of user defined functions and operators. In the relational model attributes are traditionally atomic. That is they cannot be decomposed by the database.

Object relational systems support complex objects which consist of aggregations of values of other types. In an object-relational system, mechanism exist for the definition of complex objects, complex objects may be manipulated by SQL queries, complex objects may be used to introduce new types into the system and it is possible to introduce user defined functions which operate on complex objects. Inheritance is one of the key concepts of object-orientation. In object-oriented programming it allows a programmer to re-use already written code when defining a new type. Inheritance has been introduced into object-relational systems in order to re-use complex object definitions and the user-defined functions that operate on them. It is possible to define a subtype of an existing type. The new type will inherit the data and the functions of its super-type. The increased complexity of the applications it is possible to build through the use of object-relational systems require additional integrity constraints to those provided in conventional relational systems. One mechanism exploited for this is the use of rules. Each rule is associated with an event. When that event occurs, the operation associated with the rule is carried out. Rules are used to ensure that the database is maintained in consistent state and returns consistent answers to queries. These four types of extensions appear to be logical devel-opments of relational technology. They do, however require relational vendors to completely re-engineer their offerings. One key area of concern is query optimization. The ability of a relational database management system to transform submitted queries into optimized equivalents was cited above as one of the major benefits of the relational approach.

VI. SIMULTANEOUS DEVELOPMENTS

The previous discussion has dealt with the general trends of commercial database management systems. In parallel with these developments there has been much research into other aspects of database technology. These research strands have looked at applications that have been largely ignored by the commercial vendors. This section presents a summary of these areas.

6.1 Deductive databases

Deductive databases provide mechanisms whereby new facts can be inferred via rules from data stored in the database. They provide a mechanism for capturing behavioural semantics within the database in a declarative manner [9-11].

6.2 Active databases

Active database research looks at the way database management systems can be built so that the system is able

to react to events occurring in the database [12-14]. Again the key issue is to develop mechanisms which can be stored within the database rather than in an external program.

6.3 Temporal databases

Temporal database systems deal with situations where facts are associated with a time [15, 16]. Time can be handled successfully using a relational database but only with considerable effort on the part of the implementor. Temporal database systems add time-related semantics to conventional systems.

6.4 Distributed databases

Initially database systems were regarded as a centralized information resource located on a single central computer. This no longer matches the structure of today's multinational companies or indeed the way many smaller companies organize themselves. Distributed database research seeks to devise solutions to the issues that arise where global data is found in a number of geographically distinct locations [17-20].

6.5. Multimedia databases

The proponents of both object-oriented and object-relational database management systems cite multimedia as an application area in which their systems will be effective. This is because multimedia is an area where the use of complex objects will be essential. Much research has been undertaken to determine what the requirements of this type of system will be [21, 22].

6.6 Spatial databases

In conventional data processing databases relationships are relatively simple and involve a limited number of entities. For example, a part appears on an order. In three-dimensional space every entity relates to every other entity and the relationships are much more complex. Spatial data-bases attempt to capture these semantics [23].

6.7 Component database systems

Component database systems are database management systems, which can be extended through the addition of software components [24, 25]. This is an extremely promising field of research and may well form the basis for the next major development in commercial data management systems.

VII. LIMITATIONS OF CONVENTIONAL DBMS IN CAD APPLICATIONS

DBMS employing some of the data models discussed above are widely used commercially. Some have been applied to the storage of CAD data with varying degrees of success but it is generally accepted that they do not provide an ideal solution. CAD applications often impose additional requirements which are not adequately provided for by conventional DBMS. This is mainly due to the special characteristics of CAD data and the nature of the desired processing functions; these differ significantly from those of business data and have led to the development of purpose-written DBMS for CAD. In contrast to commercial applications which deal with well-structured and fairly homogeneous data, CAD applications handle a wider range of data types including graphical, textual, procedural and other data. The relationships between the various CAD data elements are much more complex than the relatively simple relationships between business data elements. Many cyclic, recursive and other object-specific relationship types exist amongst CAD data elements and need to be suitable represented.

The effective handling of geometrical and non-geometrical data required by the engineer is often not available in commercial DBMS. Strong links are necessary between them to reinforce the bond between the graphical design representation and its non-graphical attributes. Baron [26] sees the separate handling of these two data types as not contributing to the ease of use of existing DBMS. It also makes it harder to maintain consistency between them and incurs a time penalty in responding to interactive queries originating from the CAD system.

The static schema definition of most DBMS is incompatible with the evolutionary nature of engineering design. CAD data structures grow with the design of the artifact and cannot be constrained to a pre-defined structure. Other limitations of the business-oriented DBMS include their limited tools for integrity maintenance and inability to support multiple representations. Because of their static structure, commercial DBMS have difficulty in coping with changes and modifications in the evolving design; inconsistencies could be introduced into the database which are expensive to eliminate. Also, the need for the designer and/or other design team members to view CAD data from different perspectives is not usually supported. The limited speed of the access mechanisms of commercial DBMS makes them unsuitable for interactive engineering design which has strict requirements for response times. This is becoming less of a problem than hitherto as current DBMS become more interactive. Much of CAD information is interpreted and not explicit; there is therefore the need for the semantics of CAD data to be conveyed by DBMS. Conventional DBMS do not adequately provide for this and are unable to maintain the semantic integrity of the data.

The preceding catalogue of CAD data characteristics which are not optimally handled by commercial DBMS is not exhaustive but will suffice to illustrate the problem.

VIII. REQUIREMENTS FOR A CAD DBMS

Attempts have been made by Kimura et a1.,[27] and Managaki [28] to identify the peculiar requirements of a CAD DBMS. Liu's criticism of conventional DBMS centres on the relational database but he also points out that similar limitations can be found in the hierarchical and network models, and goes on to deduce some 'special requirements' for CAD database systems.

Foisseau & Valette [29] also address the issue of CAD database requirements but, perhaps, the most comprehensive set of requirements can be found in the functional specification put forward by Staley [30].

A CAD DBMS requires similar facilities as a conventional DBMS but needs to be much more flexible to cope with the special characteristics of CAD data. The additional facilities required include the following:

(i) A support for dynamic model definition reflecting the evolving nature of the design which cannot be pre-defined. The designer should be able to define new classes of data, refine them or redefine them, and operate on the stored data all along the design process.

(ii) The wider range of data types which are contained in the definition of a CAD model should be supported. Closely related to this is the ability to adequately cope with both graphical and non-graphical design information.

(iii) Relational DBMS have been faulted for their limited capabilities for data abstraction; it is believed that a support for abstract data types can enhance their use for storing and managing CAD data.

(iv) A CAD database should have the ability to handle the large volume of data associated with CAD. The complex inter-relationships between these items of data also need to be adequately represented; this will ensure efficient retrieval of information and limit response times.

(v) Maintenance of data integrity is of utmost importance. Unlike in business-oriented DBMS where relationships are relatively simple and constant over time, CAD data is in a state of evolution and there is the requirement for a mechanism for maintaining integrity and consistency of data.

(vi) Somewhat related to the above is the requirement for version control. Version control must allow for the development of alternative engineering designs that can be globally evaluated prior to permanent updating of the database.

(vii) A facility for long transaction processing in contrast to the shorter transaction durations in business applications is necessary in a CAD database. An engineer generally works for long periods on a set of data in an interactive mode; updating operations need to take account of this.

(viii) There are usually several parties involved in an engineering project and the CAD database has to provide for concurrent access (and distributed processing where access from remote stations is required). This multi-user access to the CAD database whilst necessary, induces a need for some form of security or access control facility to prevent data being corrupted or interfered with.

(ix) In engineering design, it is sometimes necessary to see data from several perspectives or at various levels of detail. Thus, a CAD database should have the capability to support multi-representations/multi-views of objects and detail level control. The data in view remains essentially the same but is able to communicate different aspects of design information to the end-user/designer depending on his point of view.

(x) A CAD database should satisfy the need for information to be derived from stored data. This implies a capacity for some level of embedding of semantic information or knowledge encapsulation within the database. A knowledge-based approach is expected to facilitate information deduction and decision-making.

Additional requirements could be added to those highlighted above - some more specific to different CAD applications and others applying to databases in general. The ideal CAD database is not yet a physical reality; this is buttressed by the conclusion of a SERC-sponsored assessment of databases for engineering to the effect that ‘no database management system (DBMS) currently available can provide all of the facilities required in engineering applications. However, this is an area of active research and it may not be too long before a suitable CAD DBMS emerges; such a system will need to incorporate the preceding requirements.

IX. ROLE OF PRODUCT MODELS

Product models are seen in many circles as a means of addressing the above requirements. A product model is an abstract definition of a product with, ideally, all relevant product data stored in the model and able to be abstracted to documents in various formats. It contains product data which is defined by the International Organisation for Standardisation (ISO) as ‘a representation of facts, concepts, or instructions about one or more products in a formal manner suitable for communication, interpretation, or processing by human beings or by automatic means’. The product model thus contains both geometric and non-geometric information and uses the concepts of the discipline involved. It is usually intended that the model defines the various data generated through the product life-cycle - from specification through design to manufacture. The driving force behind the continuing evolution of product models has been the STEP standard which seeks to facilitate the transfer (between CAD systems) of both graphical information and the underlying non-graphical engineering information. While they offer significant potential for addressing the requirements for an engineering DBMS, there are still difficulties to overcome before product modeling-based systems are commercially available. The differences across engineering disciplines in the nature of the product being modeled serves only to further complicate issues. However, appropriate frameworks for the development and utilization of product models in many disciplines (including construction) are being formulated.

X. CONCLUSIONS AND FUTURE WORK

This paper has attempted to show by considering the history of the development of database management systems, what such systems are about and what makes certain systems successful in the world of business and commerce. The key issues identified are:

(i) Database management systems must hide the complexity of permanent storage mechanisms from programmers.

(ii) Database management systems must act as a tool to speed up application development.

(iii) Database management systems must provide facilities suited to the requirements of the environment they exist in, over time these requirements will become more complex.

The development of database technology has largely been driven by the emergence of new application areas. The key drivers of the future are likely to be multimedia and Internet applications. We should expect to see developments in the database world that provides facilities for dealing with these challenges. In the past we have seen database technology applied almost exclusively to data processing requirements. This concentration on a single area has led to a situation where one flavour of technology has dominated. In the last decade other areas of computing have become interested in the use of databases. This will probably mean that a number of database models will coexist. Data processing systems will probably uses object-relational systems and SQL 3. This technology is, however, unsuited to many other application areas and these will use object databases. Other types of database solution may emerge to meet new requirements.

It is clear from the above review of data structures and database management systems (DBMS) for CAD that careful design of these facilities is essential for the effectiveness of a CAD system. The structures adopted have implications for the accuracy and robustness of the model, the speed of operation, the efficiency of data retrieval mechanisms, and the exchange of data with other applications to mention but a few. Existing storage mechanisms have been shown to be inadequate for CAD data and a set of requirements for a CAD DBMS drawn up to highlight several desirable features. The emergence of product models is seen as an important opportunity to address many of the requirements. However, there are still many practical issues to resolve before an appropriate framework for effective modeling of CAD data becomes a practical reality.

References

[1] P.P.S. Chen, The entity-relationship model-toward a unified view of data, ACM Transactions on Database Systems 1(1) (1976) 9-36.

[2] T.J. Teorey, D. Yang,, J.P. Fry, A logical design methodology for relational databases using the extended entity-relationship model, ACM Computer Surveys 18 (2) (1986) 197–222.

[3] M. Hammer, D. McLeod, Database description with SDM: a semantic database model, ACM Transactions on Database Systems 6 (3) (1981), 351–386.

[4] W. Kent, Data and Reality, North Holland, Amsterdam, 1978.

[5] W. Kent, Limitations of record-based information models, ACM Transactions on Database Management Systems March (1979).

[6] M. Atkinson, F. Banchilon, D. De Witt, K. Dittrich, D. Maier, S. Zdonik, The object-oriented database system manifesto, Proceedings of the First International Conference on Distributed and Object-Oriented Design, 1989.

[7] M. Stonebraker, L.A. Rowe, B. Lindsay, J. Gray, M. Carey, P. Bern-stein, D. Beech, Third generation database systems manifesto, ACM SIGMOD Record 19 (3) (1990) 31–44.

[8] M. Stonebraker, D. Moore, Object-Relational DBMSs The Next Great Wave, Morgan Kaufmann, Los Altos, CA, 1996.

[9] S. Ceri, G. Gottlog, L. Tanca, Logic Programming and Databases, Springer, Berlin, 1990.

[10] S.K. Das, Deductive Databases and Logic Programming, Addison-Wesley, New York, 1992.

[11] G. Wagner, Foundations of Knowledge Systems, with Applications to Databases and Agents, Kluwer Academics, Dordrecht, 1998.

[12] Active Rules in Database Systems, in: N.W. Paton (Ed.), Springer, Berlin, 1999.

[13] S. Ceri, P. Fraternalli, Designing database applications with objects and rules. Addison-Wesley, New York, 1997.

[14] K.T. Owens, Building intelligent databases with Oracle PL/SQL. Triggers and stored procedures, Prentice Hall, Englewood Cliffs, NJ, 1998.

[15] Temporal Databases: Research and Practice, in: O. Etzion, S. Jajodia, S. Sripada (Eds.), Springer, Berlin, 1998.

[16] R.T. Snodgrass, The Temporal Query Language TSQL2, Kluwer Academics, Dordrecht, 1995.

[17] A. Sheth, J. Larson, Federated database systems, ACM Computing Surveys 23 (3) (1990) 183–236.

[18] D. Bell, J. Grimson, Distributed Database Systems, Addison-Wesley, New York, 1992.

[19] S. Ceri, G. Pelagatti, Distributed Databases: Principles and systems, McGraw-Hill, New York, 1994.

[20] M.T. O ¨zsu, P. Valduriez, Principles of Distributed Database Systems, 2, Prentice Hall, Englewood Cliffs, NJ, 1999.

[21] H. Samet, General research issues in multimedia database systems, ACM Computing Surveys 27 (4) (1995) 630–632.

[22] V.S. Subrahmanian, Principles of Multimedia Database Management Systems, Morgan Kaufmann, Los Altos, CA, 1997.

[23] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, New York, 1990.

[24] A. Geppert, K.R. Dittrich, Constructing the next 100 database management systems: like the handyman or like the engineer?, ACM SIGMOD Record 23 (1) (1994) 27–33.

[25] J.A. Blakeley, OLD DB: a Component DBMS architecture, Proceedings of the 12th International Conference on Data Engineering (ICDE), New Orleans, February 1996.

[26] Baron N. et al., An approach to the integration of geometrical capabilities into a data base for CAD applications. In File Structures and Databases for CAD, eds Encarnacao, J. & Krause, F.L., North-Holland, Amsterdam, 1982, pp. 231-240.

[27] Kimura F. et al., Construction and uses of an engineering data base in design and manufacturing environments. In File Structures and Databases for CAD, eds Encarnacao, J. & Krause, F. L., North-Holland, Amsterdam, 1982, pp. 95-111.

[28] Managaki M., Multi-layered database architecture for CAD/CAM systems. In File Structures and Databases for CAD, eds Encarnacao, J. & Krause, F. L., North-Holland, Amsterdam, 1982, pp. 315-330.

[29] Foisseau, J. & Valette, F.R., A computer aided design data model: FLOREAL. In File Structures and Databases for CAD, eds Encamacao, J. & Krause, F. L., North-Holland, Amsterdam, 1982, pp. 315-330.

[30] Staley S. M., Knowledge representation and database management for CAD. Ph.D. thesis, Purdue University, December 1985.