NISTIR 4987

Database Management Systems In Engineering

Katherine C. Morris
Mary Mitchell
Manufacturing Engineering Laboratory

Christopher Dabrowski
Elizabeth Fong
Computer Systems Laboratory

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

NIST

NISTIR 4987

Database Management Systems In Engineering

Katherine C. Morris
Mary Mitchell
Manufacturing Engineering Laboratory

Christopher Dabrowski
Elizabeth Fong
Computer Systems Laboratory

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

December 1992

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin, Secretary

TECHNOLOGY ADMINISTRATION
Robert M. White, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

DATABASE MANAGEMENT SYSTEMS IN ENGINEERING

Katherine C. Morris
Mary Mitchell
Manufacturing Engineering Laboratory

Christopher Dabrowski
Elizabeth Fong
Computer Systems Laboratory

National Institute of Standards and Technology
Gaithersburg, Maryland 20899

ABSTRACT

Until recently the applicability of database technology to engineering systems has been limited. Early database systems addressed large-scale data processing needs of easily automatable applications. These applications were characterized by very uniform data and well understood processing methods. Engineering applications, on the other hand, are characterized by highly complex data with very variable structure. The need to represent engineering data has driven advances in database technology.

Engineering domains also impose unique, new requirements on other aspects of database technology. In particular, to support the evolutionary nature of the engineering environment, recent developments in database technology have focused on the temporal dimensions of data management. In addition, the present trend in manufacturing towards concurrent engineering raises new considerations for the cooperative use of data in a distributed engineering environment. All of these factors are reflected in the new generation of database systems and are described in the article.

This manuscript will appear in
The Encyclopedia of Software Engineering,
John Wiley & Sons, Inc. Publishers.

Funding for the preparation of the manuscript was
provided in part by DARPA/SISTO — the
Defense Advanced Research Projects Agency/
Software & Intelligent Systems Technology Office.

TABLE OF CONTENTS

INTRODUCTION

Most engineering-related software addresses very specific problems. These problems are typically computation intensive and very limited in scope. Until recently this approach has been an effective use of computer and human resources. However, in the future, engineering and manufacturing processes will need more integrated product development environments. Both cultural and procedural changes are needed to support the engineering environments of the future, and these changes will require integrated software systems. Databases are essential for integrating software and for reliably sharing data among diverse groups of people and applications. Database technology will be an integral part of the emerging software environments.

In this article the application of database technology to engineering problems is examined for different levels of complexity within the computing environment. This introduction provides some background on the topic and includes the description of an example that is used throughout the article. In the first section on the NATURE OF ENGINEERING DATA, the use of database technology for stand-alone applications is considered. Mechanisms for data representation to support engineering applications are particularly important for implementing engineering software. The following section titled MANAGING CHANGES discusses database techniques for managing changes within the software environment. The next section on the COOPERATIVE ENGINEERING ENVIRONMENT discusses considerations for supporting multiple engineers working cooperatively. The state of database technology is discussed in the concluding section.

The Engineering Problem

The primary focus of engineering is on the creation of a product. The types of products that engineers produce vary widely. But whether the product is a building, an airplane, an integrated circuit, or a computer device, many facets of its creation are similar with respect to the computing technology needed to support the engineering process.

Automated control of the computing environment is needed to achieve a high-level of engineering and manufacturing productivity. The amount of information available at a person's finger tips through a computer is growing beyond the ability of people to usefully absorb it. At the same time the reliance of engineering and manufacturing processes on electronic information is also growing. The need to manage that environment is greater than ever, as is the potential to exploit access to information. The technology which controls this environment is essential to improving engineering and manufacturing processes.

The engineering and manufacturing process involves many different people and systems which typically are distributed across a business enterprise and even between enterprises. For example, in the engineering of a human-computer interface device, such as a *mouse*, one engineering group establishes the user interface requirements, such as number of buttons, dimensions, size, and other physical features of the mouse, while another group decides what type of material is suitable for the product, and a different group is responsible for packaging the product. A wide variety of software applications supports the design of a final product which incorporates the diverse perspectives of different engineering teams. Consequently, the computing environment is very complex.

Throughout this article a casing for a mouse is used as an example of an engineered product. The casing is part of the larger, consumer product — the mouse; however, the casing itself is also a product, which is supplied to the maker of the mouse. Many different aspects of the mouse casing are important to engineers developing the product:

- wire frame model representation: shape, dimensions, tolerances
- solid model representation: shape, mass, volume
- material: strength, durability, molding quality, thermal properties, elasticity
- product aspects: roles in assembly, identification, version, model number, serial number, lot number, color, packaging type

All these different aspects of the casing can be related through a conceptual model or schema. **Figure 1** illustrates the schema for a product such as a mouse casing. The schema describes the types of information and the structure of the information needed to engineer a product. The example is very general and simplifies the actual information which would be needed to represent the complete information for a product. For instance, solid models and wire frame models each require extensive and different data structures for their representations. These portions of the schemas are omitted here to simplify the example.

The Need for Database Solutions for the Problems of Engineering

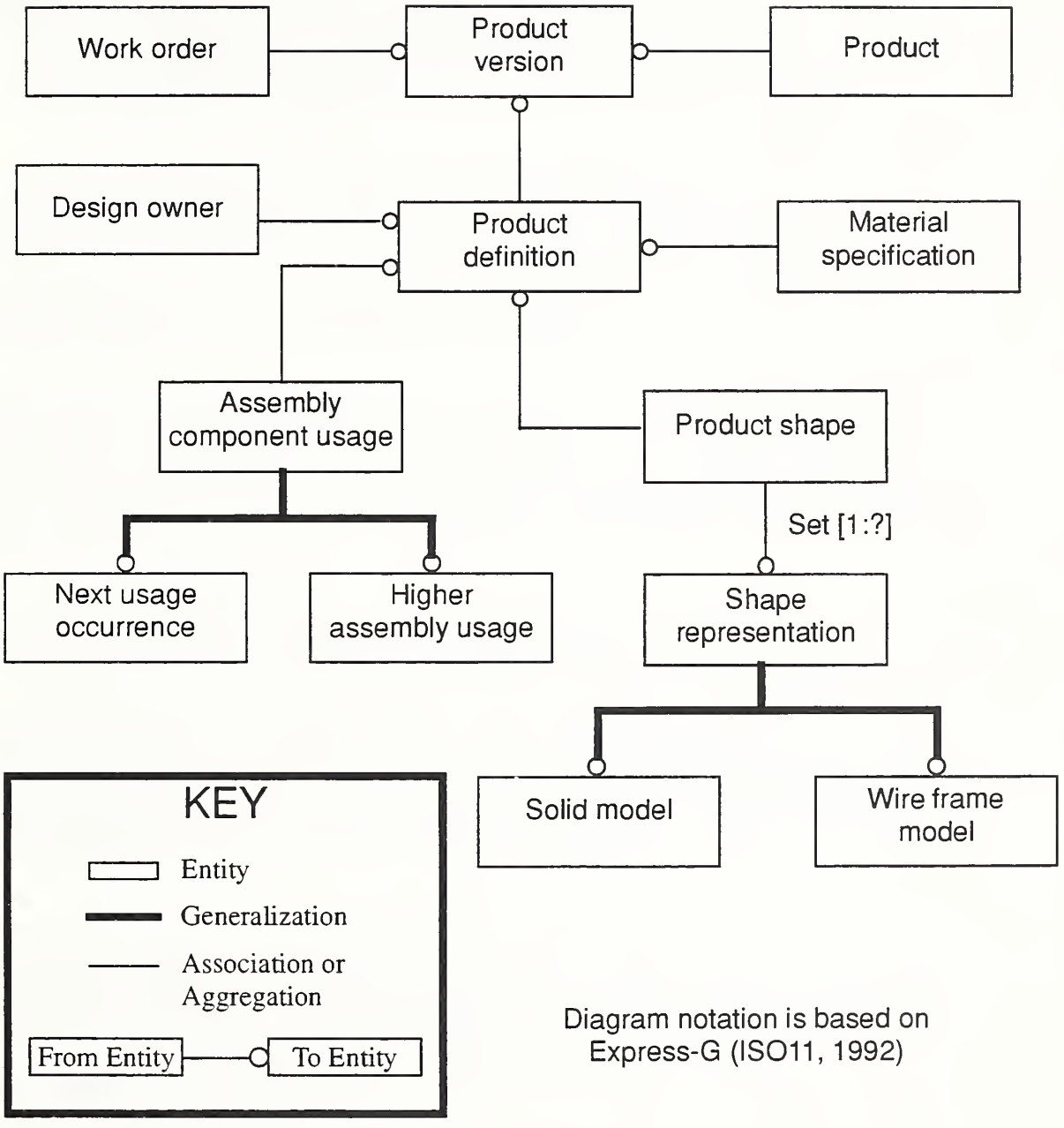
Many computationally intensive problems from a variety of engineering domains benefit from the use of computer technology. The pocket calculator antiquated the slide rule, which only a generation ago was the primary tool of trade in many engineering domains. Similarly, problems of larger scope have been addressed by software development teams. These groups of programmers typically implement a sub-system of a particular engineering discipline, such as the analysis of the structural properties of a product or numerical-control optimization for manufacturing.

The multiplicity of computer-based support systems[1] for engineering is indicative of the utility of this approach for automating the engineering process. The limited scope of such sub-systems is well suited for implementation in software because the problems are manageable. From the perspective of human understanding, these problems are a suitable size to automate since the processes are well understood. From the perspective of managing software development, the solutions to these problems are reasonable since the solutions can be implemented by a small software development project.

The historical evolution of computing for engineering has resulted in *islands of automation*. Since the specialized sub-systems are isolated from each other, each sub-system can be thought of as an island in the process of building a final product. The challenge for the next generation of software

1. Computer-based support systems for engineering support any Computer-Aided operation or process and are sometimes generically referred to as CAx systems. They include MCAD (Mechanical Computer-Aided Design) e.g. drawing/drafting; ECAD (Electrical Computer-Aided Design), e.g. PCB (Printed Circuit Board) layout; MCAE (Mechanical Computer-Aided Engineering), e.g. solids modeling; ECAE (Electrical Computer-Aided Engineering), e.g. logic design; CAM (Computer-Aided Manufacturing), CIM (Computer-Integrated Manufacturing), e.g. NC (Numerical Control) processing and photo-plotting.

Figure 1: Example Data Model



engineers is to integrate these systems in meaningful ways. Their integration will lead to a smoother automation of the entire engineering process.

One significant factor for improving the current product development process is the availability and consistency of data throughout the process. Database technology provides mechanisms to manage the consistency of data and to manage the availability of data as it progresses through the product life-cycle. Therefore, database technology is a key component of the future software environment for engineering.

Database technology has evolved in parallel to the evolution of software to support engineering. Target applications for database technology have traditionally been data intensive business applications. In these applications relatively simple operations are performed on large volumes of data with uniform structure. The term *data processing* refers to these types of applications. The engineering world, on the other hand, is full of computationally intensive, logically complex applications requiring sophisticated representations.

Recent developments in database technology emphasize the need to provide general purpose support for the type of functions involved in the engineering process. Modern database systems are evolving to support rich information models that address both data representation and data semantics. In addition, more sophisticated operating environments for shared usage of information are also emerging.

The typical software environment for engineering applications consists of independent programs which share data using an operating system's facilities for file storage. This mode of data sharing has many shortcomings that have been addressed by database systems. Using the file system to share data causes programs to be dependent on a fixed format for the data. In addition, operating system environments only manage disk access to prevent the concurrent reading and writing of files. Concurrency control at the logical level is necessary for control over extensive sharing of data. Database systems extend operating systems' mechanisms by providing more sophisticated methods for managing concurrency at a logical level so that shared data does not become corrupted.

Database systems improve on file systems in the following ways:

- Data access is based on logical structure rather than the physical structure of a data file; therefore, applications do not need to parse files to access data and access can be controlled at a logical level rather than by file.
- Database management systems provide mechanisms to optimize memory management and disk storage.
- Rules for managing data consistency can be defined along with the structure of the data so that data consistency is not the responsibility of every application program but is uniformly maintained whenever data is accessed.

Despite the advantages of traditional database management systems over file systems, database management systems have disadvantages that have made them unsuitable for engineering applications until recently (Encarnação, 1990). Traditionally the advantages of file systems over the database management systems have been the speed of access to the data and user control over the

data structures. These advantages are quickly disappearing as database access is organized around more robust structures and as physical storage techniques evolve to better handle the dynamic storage requirements of engineering systems (Bancilhon, 1990) (Cattell, 1992) (Kim, 1989). Modern database systems offer these advantages over their predecessors:

- Integration with the programming languages used in engineering has made disk access transparent to the application program and easier for the application programmer to use.
- More flexible memory management techniques, designed to support complex data, have improved the performance of database systems for engineering applications.
- New mechanisms to support flexible system integration and cooperative work have been developed to address the particular needs of engineering.

Using Database Technology in Engineering

The use of database technology in the engineering computing environment differs from traditional data processing. Many database features developed for the more traditional applications can be transferred to the engineering environment. Some of these features take on new meaning, but many are only viewed from a different perspective in engineering applications. Engineering applications impose unique requirements on, but also can particularly benefit from, the following aspects of database technology:

- data representation
- change management
- cooperative processing

The initial application of database technology to the engineering computing environment solves some of the simpler needs of engineering applications. Database management systems provide data representation capabilities that decouple the logical format for data representation from an inflexible file format. In this sense the database becomes a *persistent* data store which alleviates the need for each application to parse a file into the program's internal data structures.

The sophisticated data representation capabilities needed to support engineering systems have only recently emerged in database systems. Engineering data contains complex interrelations and data types for which general purpose support has not been available until recently. The typical engineering application involves highly structured data and navigation of these structures is a more common operation than repeated processing of a single data structure. The more recent approaches to data representation provide a platform for direct expression and encoding of the rich semantics of data with respect to representation and constraints. Techniques for data representation which are particularly suitable for engineering are described in the section on the NATURE OF ENGINEERING DATA.

Strategies for applying database technology to the broader category of needs for the entire engineering computing environment are also emerging. These strategies address the coordination of the evolution of a product's development throughout its life-cycle. The introduction of database technology into engineering processes can provide the opportunity to improve the entire way of doing business for many industries. The technology is a key enabling factor for future directions

in engineering and manufacturing automation and it is essential for concurrent engineering, flexible manufacturing, and enterprise integration.

Changes within the engineering environment are an inherent part of the engineering process. As a product is being developed, information needs are evolving along with it. Managing those changes is a crucial service that can be supported by a database management system. A reliable change management system is essential for reducing the time needed to engineer and manufacture a quality product. Database support for managing the changes within the environment are described in the section on MANAGING CHANGES.

Another factor in accelerating the product development process is the availability of data throughout the process. The typical hardware working environment for engineering applications consists of computing workstations and file systems that are connected by an electronic network. This hardware topology does not inherently support shared access to data; therefore, additional methods are needed to support data sharing in this environment. Database management systems provide mechanisms for concurrency control and data distribution which make data quickly and reliably available to a variety of groups cooperating in the development process. These techniques are discussed in the section titled COOPERATIVE ENGINEERING ENVIRONMENT.

The remainder of this article discusses the relevance of database technology in engineering applications as outlined above. Database technology is useful for all types of engineering applications. As the engineering computing environment increasingly grows more complex, the use of database technology will play a more significant role in the engineering process than is the practice today.

NATURE OF ENGINEERING DATA

The nature of the data in engineering applications is quite different from business applications. In engineering applications, data is characterized by highly complex interconnections between structures. The volume of the data, while potentially large, is a secondary consideration to the complexity of the data. The following characteristics of engineering data illustrate its complexity:

- **The structure of engineering data is non-uniform and unpredictable.** Instances of a single conceptual structure can vary in size and the data set representing an entire product may consist of a large number of different data structures with relatively few instances of each structure. For example, a complete description of a product includes all aspects from physical design to material requirements to cost estimates. Many complex structures are needed to represent such a diverse set of information. In addition, for many of the structures there will be only one instance for each product and that instance may be shared across products.
- **Many of the commonly manipulated concepts require representations which are networks of data structures and relationships.** For example, the geometry needed to represent a wire frame model is typically a collection of points with specific relationships defined between them in meaningful ways for visualizing the connectivity of elements of the design. The wire frame representation of the shape is only one aspect of

the product, which must be connected to other representations and associated information.

- **The interconnections between data structures are numerous and the same data structure may participate in many roles.** In the example, the mouse casing is a product; however, from another perspective the casing is a component in the assembly of a different product, the mouse.
- **A large percentage of data is dependent on the existence of other data.** For example, a version of a product is dependent on an initial product specification.
- **Completeness of a data set is relative to the stage in a product's life-cycle.** For example, when a product is initially designed, it will not have manufacturing data associated with it.
- **The level of accuracy needed for numeric values can vary depending on the semantics of the data and on the application using the data.** For example, the application used to verify that the mouse buttons can be assembled into the mouse casing requires more precision than the application used to design the packaging for the mouse casing.
- **Complex rules exist for data instantiation.** For example, when a product is defined to be an assembly, it must have more than one component and each component will have at least one mating condition which correlates to a mating condition in another component.
- **Algorithms may be required to ensure data integrity.** For example, when a component of an assembly is deleted, the structure of the assembly needs to be resolved. When the design of the mouse casing is deleted, the mouse buttons are also affected. In some circumstances the designs for the buttons could be removed from the database, but in other cases, such as when the designs are used in other assemblies, they are still needed.
- **A single abstract concept may sometimes be represented at a detailed level in more than one way.** Different representations are required to support the different functions for established engineering practices. For example, the shape representation of a product can be a wire frame model, a simple raster image, or solid model. The choice of representation is dependent on established practices for the type of product and the task which uses the representation. To represent a product designed as an assembly, a wire frame model of the components or a raster image of the assembly is typically used to illustrate the shapes and spatial relationships of the components. A solid model representation of the product's shape might not be as useful in illustrating the assembly.
- **The evolution of a product or design is an important historical record.** The history provides a basis for reconstructing the rationale for design changes and is important for improving the design process. For example, product liability can exist for decades; therefore, an accurate design change history is an essential record.

In the mid 1980's several authors (Katz, 1985) (Powell, 1988) (Rumble, 1984) (Staley, 1986) (Su, 1986) identified the lack of a suitable data modeling technology for engineering data. Database design methods at that time focused on the data processing needs of business applications. The

data processing field concentrated on increasing the speed of processing large amounts of uniform data with simple structures. The term *impedance mismatch* has come to refer to the mismatch between the organization of data supported by traditional database management systems and the organizational needs of engineering applications.[1]

Data modeling techniques emerging today specifically address the types of problems common to engineering applications. The techniques combine features of traditional database design (Date, 1990) with knowledge representation techniques from the field of artificial intelligence (Sowa, 1991). The technology known as *semantic* or *conceptual information modeling* resulted from the application of semantic networks to database design (Batini, 1992) (Brodie, 1984) (Mylopoulos, 1989).

Conceptual modeling produces a *conceptual design* of the entire contents of a database based on the semantics of the data. From a conceptual design a *database schema,* suitable for a particular database management system, is derived. The database schema is then evaluated with respect to the usage patterns of the applications and refined to support their performance needs. Thus several database schemas can support the same conceptual design. Briefly, the process of database design can be viewed as going through three stages:

- Conceptual modeling includes requirements analysis and results in a conceptual design.
- Database implementation results in a database schema for a database management system.
- Physical design optimizes the way in which data is stored on physical media.

In the first stage, conceptual modeling, the complexity of the data is captured without the details needed for computer implementation. This stage results in an understanding of requirements which are then represented in a conceptual design. The conceptual design specifies, as fully as possible, the semantics of the information involved in the process or processes being modeled. A conceptual design is used to:

- develop agreements on requirements for specific engineering tasks,
- specify the design of information systems in terms familiar to an engineer, and
- develop integrated information requirements which are common to multiple engineering tasks.

In the next stage a database schema suitable for a computer implementation is developed. A conceptual design is translated into a schema compatible with the targeted database management system. The schema for a particular database management system is represented in a data definition language. The application designer translates the conceptual design into the data definition language for the database management system chosen for implementation.

Both data models—the conceptual design and the database schema — provide an abstraction of the information needed for one or more application uses in the real-world. Certain details such as the actual physical organization on computer hardware are deliberately omitted. Details on the

1. More generally the term *impedance mismatch* refers to the situation where the structures used to define a database schema drastically differ from the data structures needed by the application.

physical organization of the data are addressed in the physical database design. This separation of concerns into different types of designs improves control over and planning for the information system.

Characteristics of these data models as used for engineering databases are presented in this section. A discussion of conceptual modeling is followed by a discussion of particular database constructs which are significant to engineering applications. Physical design is primarily controlled by the database management system; however, some relevant aspects are discussed in the remainder of the section.

The Conceptual Design

An engineer's perception of the information needed to perform a job differs from a computer's need to organize data into structures for efficient access and storage in a database. A conceptual design reflects the engineer's use of information. It presents information needs in understandable terms while capturing details about the concepts and things relevant to an engineering application. Just as the design of a product is important in communicating the functional aspects of the product, a design for the engineering data is important in communicating the functional aspects of information. Conceptual designs are particularly useful for developing engineering databases because they provide:

- **a method for managing the data model**: models remain intellectually manageable even as the application complexity increases,
- **data independence**: models are not tied to the physical organization of data,
- **data stability**: as an application is refined, models evolve without changes at the detailed level affecting higher level abstractions,
- **precision of expression**: models accurately capture all the data relationships which are important from perspective of a particular application,
- **support for integration**: the flexibility provided by abstraction mechanisms increases the visibility of data structures and constraints and thereby facilitates integration of different engineers' perspectives on the data.

Several different approaches to conceptual modeling have been developed. The approaches include techniques for developing and documenting a conceptual design. The techniques can be divided into three categories: *entity relationship* or binary relationship models (Chen, 1976) (Nijsen, 1989), *semantic* or extended semantic data models (Sowa, 1984) (Hull, 1987) (Peckham, 1988), and *predicate logic* models. (ISO-TC97, 1987) (Ullman, 1988).

Engineering organizations and software vendors commonly select one category of conceptual modeling techniques and provide additional guidance on how to apply the techniques consistently within their organization. Commercial computer-aided software engineering (CASE) systems are available for some of these techniques. Additionally, these systems often generate database schemas for one or more database management systems.

Conceptual modeling techniques (Brodie, 1984) (Borgida, 1985) provide mechanisms for representing both

- structural properties of data and
- constraints on data.

Techniques for representing structural properties include mechanisms for describing relationships between data. Mechanisms for expressing constraints on data can capture information such as acceptable domains for data values or required relationships between structures. The following paragraphs describe some of the important mechanisms for supporting engineering applications. These mechanisms are common to many of the conceptual modeling techniques.

Support for structural properties

Conceptual modeling techniques often provide four mechanisms for describing and managing the complex structure of information (Brodie, 1984):

- classification
- association
- aggregation
- generalization

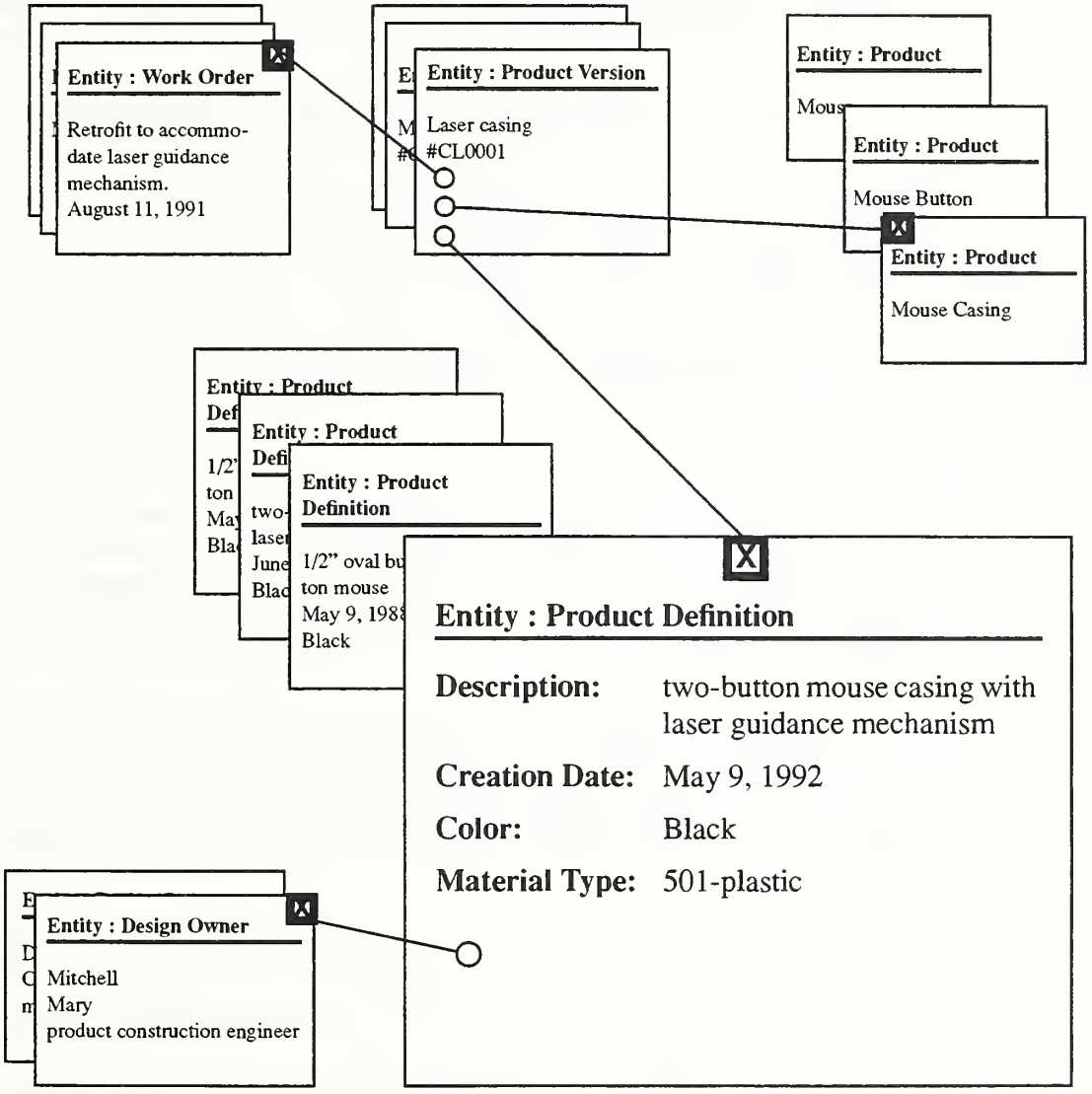
All conceptual modeling techniques provide *classification* mechanisms. Classification is used to describe entities and the associations between entities. An entity is an abstraction of a real-world concept which is significant to the application. An entity can have many instances where each instance has different values but still has the same data structure. In the example a mouse casing is one instance of a product; the mouse is another type of product. Casings may come in different colors; those casings are also separate instances of the abstract concept of product. Thus an entity represents an abstract class of things that share similar structure and function.

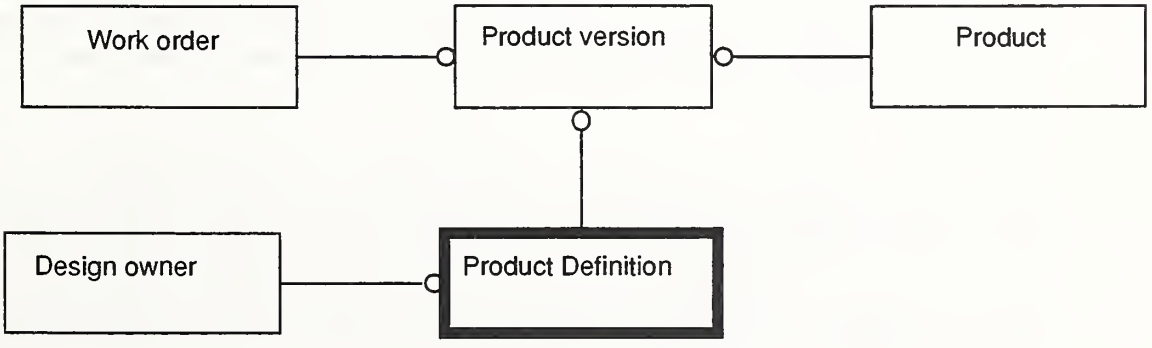
An entity may have attributes which represent the characteristics or properties of the entity. For example, a mouse casing has a color; in **Figure 2** color is shown to be an attribute of the *product definition* entity. Instantiation of an entity refers to the declaration of a set of values for describing a specific real-world occurrence of an entity. A particular mouse casing is black, whereas mouse casings in general have a color.

Relationships between entities are represented in a number of different ways. Sometimes the connection is loose when the entities exist independently of each other. These relationships are sometimes called *associations* or *hasa* relationships. For example, a product definition has a design owner; however, both the product definition and the design owner exist as meaningful items without the other. Networks of relationships between entities are formed by using association.

More complex relationships are represented using the techniques of aggregation and generalization (Smith, 1977). *Aggregation* uses a single entity to represent the composition of several entities. Conversely, an entity may be decomposed into several entities. A reference to the aggregate entity implies a reference to all of its component entities. This mechanism represents the relationship where one entity is a part of another and is sometimes called the *partof* relationship. The entity *wire frame model* can be thought of as an aggregate in **Figure 1**, since all the information

Figure 2: Product Instance





needed to represent a wire frame model is not included in this figure. Thus the entity is an abstraction representing all the entity types needed to compose a wire frame model. *Generalization* represents a relationship between two entities where one entity is a specialized kind of the other entity; this relationship is called an *isa* relationship. The specialized entity inherits the characteristics of the more general entity. Hierarchies of entities are formed using the generalization mechanism.

In **Figure 1** *shape representation* is shown to be a generalization of both *solid model representation* and *wire frame model representation*. Both *solid model* and *wire frame model* are representations for shapes. As shapes they share some common characteristics; however, they are very different from each other in the details needed for their representations and in the functions they support. For instance both representations describe a shape and can be used to estimate a product's dimensions, but in each case the method for calculating the dimensions is very different. In this example the specialized entities are mutually exclusive (i.e. a solid model is not also a wire frame model); however, specialized entities need not always be exclusive.

Support for constraints

Conceptual modeling techniques also support the specification of *constraints* on data. Constraints aid in defining the semantics of data by imposing restrictions on the acceptable data values for instances of entities and their attributes. Different conceptual modeling techniques support the representation of different types of constraints. The following categories of constraints are common among several techniques:

- **uniqueness**: An attribute or group of attributes may be designated as a unique identifier for an entity. This provides a way of identifying an instance of an entity by name. For example, a mouse casing may be identifiable by a model number.
- **existence dependence**: To be meaningful some entities may be dependent on other entities for their existence. For example, it would not be meaningful to have a product version without a product!
- **domains**: The domain of attributes may be restricted. For example, the mouse casing must be constructed out of a suitable material. Lead would not be an acceptable material for the casing due to weight restrictions and molding quality, although lead may be used for other parts of the mouse.
- **optionality**: In some cases optional information may be associated with an entity without affecting the validity of the instances. Whether or not certain information is optional may be captured in the conceptual design. For example, when a mouse casing is initially designed, it may not have a release date assigned but the design is still valid. However, when the design is released for production, then a release date is required for the design to be valid.
- **cardinality**: Cardinality constrains how one entity relates to another by restricting the number of instances of each entity that may participate in the relationship. The schema in **Figure 1** illustrates that a product's shape may have several shape representations; the label *SET [1:?]* indicates that at least one shape representation is needed for a product. Such constraints contribute to the meaning of the relationships. Each shape representation for the mouse casing represents the same product shape but in different ways each with distinct purposes.

Application Integration

Conceptual modeling is especially useful for integrating different applications. The semantic integration of data is necessary for different applications to share the same data. An integrated conceptual design is developed based on analysis of the semantics of the data from the different applications. The integrated conceptual design is the first step in enabling system interoperability. The integration process provides understanding of the data used by the applications, identifies areas in which data overlaps, and formulates consistent definitions to be used by the applications. Techniques for semantic integration are described in the section on MANAGING CHANGES.

To illustrate the significance of application integration, consider the example of the mouse casing. Suppose that changes need to be made to the casing due to a design flaw: the left button has a tendency to break since people apply more force with the index finger. Several different applications will be involved in resolving this problem. A design application captures usage requirements for the casing. During the design process usage requirements, such as pressure and temperature requirements, are initially recorded without decisions as to the specific material for the product. Another application analyzes the materials that could be used to create the casing. This application uses the requirements to select a suitable material for the casing. A third application is used to analyze the entire design, including material, to determine whether the flaw has been corrected. Such an analysis could use data from the design database, such as the shape of the mounting mechanism and button, and from the materials database, such as strength and temperature tolerances for the selected material. Much of the data representing the design of the complete product is not used at all in this analysis.

In order for all three applications to interoperate, common terminology is needed. The most basic requirement for managing the data in this case is for a mechanism which relates the design and the material used in the final product. However, a more thorough analysis will require more complex interconnections. Ultimately all three applications should be related through a common database schema.

The Database Schema

Typically a database schema is designed using one of the conceptual modeling techniques and then translated for implementation. Translation to a database schema is a difficult process to efficiently automate. Efficient implementation requires an understanding of the application area, the usage patterns of the data, and the underlying data structures of the database management system. These variables are unknown to the computer system; therefore, the results of an automated translation process are often not optimal. Some success at translation has been achieved, particularly in the area of conceptual modeling for relational database schemas (Batini, 1992) (Loomis, 1986) (Rumbaugh, 1991). But these techniques are most suitable for simple information systems and not for the complex applications found in engineering.

A current area of research is for mechanisms to support the translation process for engineering (Nixon, 1990). Translation mechanisms can be divided into two complementary approaches. The first approach is to expand conceptual modeling techniques to capture a broader range of information including more details about the anticipated use of the data. This approach requires identifying what information is needed to optimize an implementation. However, the addition of such

information to a conceptual design complicates the already difficult design process and thereby reduces manageability. The second approach is for the database management system to support the structures used for conceptual modeling. While this does not make the translation process automatic, it does simplify manual translation. An ultimate solution should combine the two approaches.

The database schema consists of the structures and operations necessary to define the way data is logically organized and accessed within a database management system. The technology used to represent the database schema has evolved significantly in the last decade, resulting in the *object-oriented* approach for database systems (Atkinson, 1989) (Bertino, 1991) (Cattell, 1992). Some of the major categories of database technology are *relational*, *network*, and *object-oriented* (Date, 1990). Each of these approaches provides a different logical framework for describing and accessing data in a computer system. Commercial database management systems use one of these approaches but may also combine features from more than one approach. Hence, an *extended relational* database management system may include some features of the object-oriented as well as relational approaches.

The object-oriented approach combines the traditional database approaches with concepts from the fields of programming languages and artificial intelligence; hence many of the object-oriented techniques more closely resemble approaches to conceptual modeling which share similar roots (Zdonik, 1990). While application of the object-oriented paradigm is not limited to engineering, it is the first generation of database design methodologies that is particularly well suited for engineering.

The schema for a particular database management system is represented in a database language. A database language consists of a data definition language for defining the database schema and a data manipulation language, or *query language,* for accessing and manipulating data defined by the schema. To effectively represent data in engineering applications, the database language used by the system needs methods to represent the major mechanisms used in the conceptual design. One method for representing these mechanisms is the use of classes in the database language.

Class and Association

A *class* is an implementation vehicle that corresponds to an entity in the conceptual design[1]. A class is used to define the data structures and operations needed to support the conceptual entity in a database implementation. The data structure used within a class may or may not directly reflect that described within a conceptual entity. The implementation of a class may be optimized with respect to data consistency or considerations about the usage of the class within a program or query. A class defines a logical structure for storing instances of the conceptual entities within a database management system. However, in deriving a database schema from a conceptual design a one-to-one mapping between entities and classes does not necessarily produce the best representation (Ullman, 1988). For instance, groups of entities in the conceptual model can often be collapsed into a single class definition for more efficient implementation or to enforce a tight coupling between closely related concepts. The entities *product version* and *change request* in the

1. Different approaches to data representation use different terms to refer to the concept described here as class. For example, in a relational database the term *table* is used.

example in **Figure 1** may be implemented as a single class since they are closely related ideas and instances of either entity should not exist without an instance of the other entity. A new product version should not be created without a *change request*, and a change request should generate a new product version. Furthermore, an application often needs both sets of data at the same time; in this case the usage of the data is reflected in the database schema by combining the entities into a single class.

The concept of class is of central importance to engineering database models. A class is a general template that provides structure for representing information. A class groups an entity's attributes in the database schema. The class represents attributes using simple data types such as integer, floating point, and character types and identifies *associations* to other classes. An instance of a class, often referred to as an *object*, is the set of data that represents one occurrence of the thing described by the class. In **Figure 2** the object "two-button mouse casing with laser guidance mechanism" is an instance of the class *product definition*. In addition to providing a storage structure, a class may provide operations for accessing and manipulating the instance data.

Aggregation

In engineering applications aggregate relationships are very common. A product definition, as shown in **Figure 1** for example, is an aggregation of a product's shape, material specification, and components of an assembly. The relationship between the members of an aggregation and the aggregate entity itself is an important consideration for managing data. In some cases, a member of an aggregation may depend on the existence of the aggregate entity for its own existence. For example, the existence of a product's shape depends on the existence of the product's definition. However, in other cases, the aggregation implements the reuse of shared data. For example, multiple products can use the same material specification. The implementation and semantics of these relationships may be supported in the database schema.

The dependence of a member of an aggregation on the aggregate entity, such as the product's shape on the product definition, is known as an *existence dependency*. The implementation of this type of dependency may be encapsulated in a *containment* class definition. A containment class contains the members of the aggregation. This concept is important for maintaining consistency and for efficient implementation of a database system. For example, information about existence dependencies can be used by the database management system to automatically remove dependent instances when their containers are deleted. Similarly, the relationship indicates potential access patterns for the data. For example, the wire frame representation of a product shape may contain voluminous geometric data describing that shape. Thus when the wire frame representation is deleted or otherwise accessed, the same operation can be applied to all the associated geometric data.

When aggregation is used to implement the reuse of components, the members of the aggregation are not dependent on the aggregate entity. In contrast, the independence of the members may be explicitly designed to provide for their reuse. In particular, multiple versions of a product may share many of the same features. When the differences between versions are small, the most efficient implementation is to keep relationships to unchanged aspects of the product and create new objects to store the changes. In the mouse casing example, when a new *product version* is created a new *product definition* is created. The new *product definition* maintains references to the

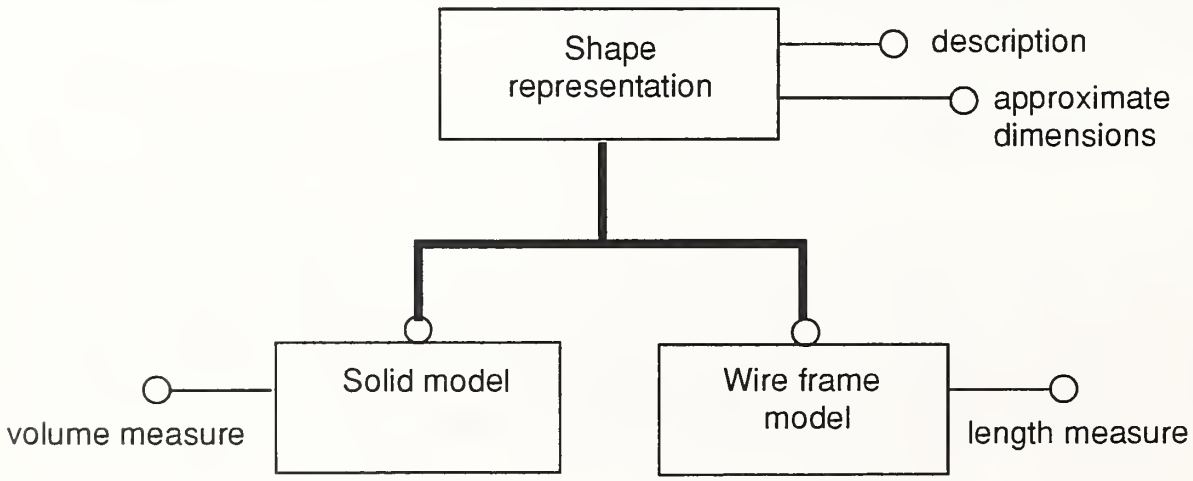
unchanged items in a previous version and creates items for changed aspects of the product. In this case the aggregation is often implemented in the database schema as a simple association.

The existence dependency between an aggregate entity and its members can not always be generalized for all instances. In other words, the relationship may differ between instances of a class. For example, a product's shape may not be meaningful without the existence of a product (as described above); however, the product's shape may be reused by multiple versions of a product. When the product's shape is being reused, then its existence is dependent on all the versions in which it is used. In a data definition language this type of relationship is more complicated to represent. Therefore, many database management systems do not explicitly support such relationships. An engineering database application will require additional mechanisms to support these relationships.

Generalization

At the conceptual level, the mechanism of generalization and its complement, specialization, is used to describe naturally occurring relationships between entities and provides a mechanism to prevent redundant definitions. In the database schema specialization through inheritance is captured in the data definition language. The definition of the more specific class designates the more general class from which attributes and operations are inherited. The more specific class definitions may add specialized attributes and operations. **Figure 3** shows more specific subtypes of shape representations inheriting attributes from the general concept of *shape representation.*

Figure 3: Generalization / Specialization



Functions for presenting an image from a wire frame model or a solid model representation may be associated with both these classes; however, the actual operations that present the images and the images themselves distinctly differ based on the type of shape representation. In object-oriented systems the specialized functions which implement the presentations may be attached to the different class definitions.

In engineering applications, complex *class hierarchies* are common. Different types of data models support different types of inheritance structures. In general, conceptual models often support more sophisticated inheritance structures than do database models. The field of conceptual modeling is trying to optimize representational capability with respect to information semantics (Sowa, 1991), while the database field is optimizing performance in a computing environment (Bancilhon, 1990). These different goals are manifested in the variety of different data models. With respect to inheritance, for example, conceptual modeling techniques commonly provide inheritance mechanisms which allow overlapping subtypes. In database systems support for this type of inheritance is much less common since it cannot be implemented as efficiently.

Support for Constraints

In general, database management systems provide only limited support for constraints. However, considerable research is focussed on this topic and commercial systems are likely to extend the support that they provide in the not too distant future. Current systems support some of the types of constraints described for conceptual designs, such as optionality and uniqueness; however, most constraints must be maintained by each application program. Support within the database management system will ensure that the constraints are rigorously and consistently applied.

The more complex types of constraints often require algorithms for their definition. Furthermore, the interpretation of the constraints may differ based on the application in which they are used. These problems make general purpose support for constraints difficult, since traditional database languages are designed for simple data manipulations and do not have the advanced features that programming languages support.

Two approaches to implementing complex constraints are developing. The first approach is seen in many of the recently emerging object-oriented database management systems. These database management systems are integrated with a programming language. The database management system uses the data structure definition capabilities of host programming language as the data definition language. In this approach constraints are implemented in and controlled by the same host language. Another approach extends the database management system interface to include the capability to initiate an external procedure from the database system. The external procedures are written in a programming language.

While these approaches provide better support for complex constraints, they are not optimal since they are not fully developed. Researchers in this area are looking for solutions to problems associated with inter-language sharing and control over the configuration of software in the environment. Ultimately, an integrated software system is desired.

Database Query Capability

In addition to providing representation mechanisms for storing data, database management systems also provide the mechanisms to access the data in the database. This capability is manifest in the form of a query language or interactive database browser. Query languages permit users to specify the retrieval of and iterate through variable-sized collections of data in the database. Engineering databases require the ability to specify retrieval of complex structures; however, traditional database management system query languages do not support this need in a manner which

is easy to use. Therefore, many database management systems provide an interactive data browser which allows the user to navigate through the data using a visual interface.

While an interactive interface is sufficient for many purposes, an easy-to-use, programmable interface is also desirable. In the realm of object-oriented database management systems, considerable research has been done to develop query languages having additional expressive power and more friendly user interfaces. Many of these approaches are integrated with an interactive browsing capability. Commercial systems incorporating the results of this research are emerging.

A database management system provides a logical interface to data which is physically stored on a disk. Advances in data representation and access support allow applications to be developed without regard for or consideration of how the data is stored on disk. However, the management of physical storage is the primary concern with respect to performance. Therefore, database management systems improve performance with mechanisms to designate advantageous physical organizations for data. Some of these mechanisms are discussed below.

The Physical Organization

The logical organization of data, as described above, enables database management systems to provide general purpose support for complex problems. However, a sound logical organization does not by itself guarantee fast response times for data access. Good performance is dependent upon the physical organization of the data on disk and the expeditious use of memory. These factors are controlled by the database management system software, but most database management systems provide mechanisms for external direction of the physical organization. Understanding which strategies for physical organization are employed by a database management system and what external interfaces are provided is an important aspect to consider when implementing an engineering database.

The physical organization of data should reflect the intended use of the database. Usage information is captured in a physical database design which guides the database management system in

- determining a physical organization of data on disk, and
- selecting complementary strategies for management of memory.

The effective use of disk and memory minimizes the computing overhead associated with disk access and can dramatically improve the performance of database applications.

For a database system supporting many different applications, achieving good overall performance can be an enormously complex problem (Jefferson 1980) (Teorey 1982). The problem of developing an effective physical organization is especially difficult for engineering information systems. In engineering applications data is not as uniform or predictable in structure as traditional applications. In addition, the amount of data needed at any given time can vary from a single value, such as the name of a product, to a massive structure, such as the representation of a circuit design for a computer.

The design of efficient physical organizations has been the subject of extended research and has led to the development and refinement of highly sophisticated techniques. Future generations of

database management systems may include the dynamic reorganization of data based on usage patterns which should further improve performance. Advances in hardware technology such as random access memory have led to the development of sophisticated techniques for managing substantial quantities of data in memory. These techniques have produced dramatic gains in performance for logically complex data.

Physical Database Design

Just as the database schema is arrived at through conceptual design, the physical organization is determined through the process of physical database design. The goal of physical design is to improve the overall performance of the database system by reducing the time needed to access data and the cost of storage.

From a user's perspective, physical design is often viewed as a collection of techniques designed to enhance database performance by optimizing the way data is stored on physical media. This view, however, does not always reflect the true complexity of the design task. Selecting an efficient overall design requires thorough analysis of the different ways that data is used by different applications. Additional factors, such as the hardware environment and limitations in the physical design methods supported by the particular database management system, further complicate the problem.

Sophisticated techniques have been developed for different aspects of physical design (March, 1983), (Carlis, 1983), (Fedorowicz, 1987), but a comprehensive methodology guaranteed to produce optimal overall designs does not exist. Tools for particular aspects of physical database design, although limited, are becoming increasingly available, especially in CASE environments. The effectiveness of these tools is limited by their inability to consider all the factors involved in complex applications. At present, physical database design is done largely by people.

The implementation of a physical database design is accomplished through facilities provided in the database language. Database management systems use many techniques to optimize their performance, but most of these are hidden from the user. Only techniques which can be controlled externally through the database language are described here. These techniques support the location and retrieval of data by structuring the physical organization on disk and also include complementary techniques for managing memory.

One of the primary goals of physical database design is to minimize the need to access data stored on disk. Disk access is much slower than memory access. Therefore, minimizing disk access often results in better performance. However, many of the techniques which optimize the retrieval of data from disk often are not optimal for updating the data. Trade-offs based on the expected use of the data, such as whether it will be updated often, need to be considered in the physical design.

File Organization

On disk, data is stored in files. File organizations are integrated physical structures that represent decisions about how to access and store data in files. File organizations may specify methods for:

- the location and retrieval of data from files, and
- the placement of data within files.

Database systems manage space utilization for file organizations as the amount of data grows (or shrinks) over time. Effective management of disk space impacts file organization performance.

A complex database system has many file organizations, each of which may store data corresponding to one or several interconnected classes in the database schema. The design of file organizations can be particularly difficult. The complexity of usage of data by different applications creates trade-offs and dependencies to consider in designing file organizations (March, 1987). Small differences in file organizations can radically effect the performance of a database system (Wiederhold, 1987). The remainder of this section discusses file organization techniques in greater detail.

File organizations utilize access methods to retrieve data. Access methods that facilitate a particular type of access are often less effective for other types of access. Three access methods are commonly supported by commercial database management systems: sequential, index, and hashing. Sequential file access is an exhaustive item by item scan of a file. This method is efficient if a substantial portion of the file must be retrieved but is costly when locating individual items in a larger file. Adding an index to the file allows individual items to be located and retrieved efficiently but requires the additional overhead of maintaining a separate index structure. Hashing is a method by which data items are located using algorithms to transform key values into physical addresses. Hashing is effective for locating individual items but is inefficient for retrieving many items in a single operation and may exhibit poor performance if file size increases drastically.

A good file organization can reduce the number of disk accesses needed to retrieve complex data by the method of placement of data within physical files. Two techniques are particularly important: clustering and segmentation. Clustering refers to the placement of related data items in close proximity to each other. Usually, this means placing related items on the same page, since the page is the smallest unit of physical storage read from disk (Cattell, 1992). Clustering permits efficient retrieval of objects whose classes have such a strong association that are normally accessed together. For instance, in the schema shown in **Figure 1**, suppose that product versions and their corresponding work orders were frequently retrieved together but seldom accessed in other ways. Performance could be improved by clustering instances of the entity *product version* with corresponding instances of *work order* on the same page. In general, aggregate classes in the database schema are prime candidates for clustering.

Segmentation techniques divide a class's attributes into groups on the basis of common access. The data for the attributes in these groups is then stored in separate segments (March, 1983). A file organization may specify division of a physical file into several segments each of which is a smaller file. By storing data in smaller files, good segmentation can reduce the time necessary to access data and transfer it to memory. For instance for the entity *product definition* in **Figure 2**,

suppose data for the attribute *description* (a potentially very long data string) is accessed separately from (and less frequently than) other data for the other attributes. Segmenting *product definition* by placing the data for *description* in one segment and data for the remaining attributes in another effectively creates two smaller files. Scanning the product definitions without their descriptions requires fewer disk accesses and prevents the unnecessary transfer of their descriptions from disk to memory. The application of clustering and segmentation on interrelated class definitions can result in a physical data organization that does not resemble the conceptual data model or the database schema.

Memory Management

Techniques for managing memory have undergone significant advances in recent years. The three commonly used techniques discussed here are buffering, pre-fetching, and pointer swizzling. By synchronizing these techniques with an effective physical organization, dramatic improvements in performance can be achieved for working with large, complex engineering data. The advent of client-server architectures has further increased overall capacity of memory. This architecture allows an application to process data on local workstations, while the database management system uses a different machine. This separation has resulted in greater flexibility in memory management, relieved contention for processing resources, and improved overall performance.

Buffering, or caching, of data is a technique commonly used by commercial database management systems. A buffer, or cache, is an area in memory into which data items retrieved from disk are placed. Overall system performance can be improved by retaining frequently retrieved data in a buffer, thus eliminating the need for repeated disk accesses. Increases in the size of computer memories have resulted in larger caches which in turn can handle larger numbers of complex objects.

Another technique for enhancing the use of memory is pre-fetching. Pre-fetching is a technique which brings data into memory before it is explicitly requested. Data to be pre-fetched can be selected based on established usage patterns or other predetermined guidelines. For instance, an application may request a portion of a complex object. If additional members of the aggregation are located on the same page, a pre-fetching strategy would retrieve all these parts at the time the initial request is made. This saves the cost of a subsequent access to the same page on disk.

Object-oriented database systems have been credited with the introduction of pointer swizzling, a technique for improving the speed with which associations between objects can be followed in memory. Pointer swizzling involves replacing symbolic references used on disk to maintain relationships between objects with the actual addresses at which the objects are stored in memory. This eliminates the overhead of using address tables to follow relationships between objects. Pointer swizzling also facilitates the transfer of swizzled objects from buffers into memory. Pointer swizzling is particularly effective if objects are retained in memory for extended periods of time and frequently referenced (Cattell, 1992).

The policy decisions as to when and where to use pre-fetching and pointer swizzled techniques are made by the database management system; however, many systems provide mechanisms which allow applications to guide these decisions.

By indexing files, clustering components of complex objects, utilizing large caches, and employing advanced techniques such as pre-fetching and pointer swizzling, the number of disk accesses necessary to retrieve engineering can be dramatically reduced. In addition, by using these techniques, data can be more efficiently managed in memory.

Summary

The design of a database system is a complex process. The techniques described above facilitate the definition and organization of an efficient database implementation for sharing data among engineering systems. However, the development of a database implementation is rarely as simple as defining and organizing information in a database schema. Many other factors are involved in an actual implementation and its maintenance.

For instance, the results from the three steps in designing a database system — a conceptual design, a database schema, and a physical design — may themselves become data within the database system. The mappings between each layer in the database design process become important pieces of information in maintaining the database implementation.

The setting in which the database system must operate may be very volatile. The maintenance of the schema in a database implementation is one of the considerations. The impact of multiple applications in the software environment leads to other considerations for the database system. More considerations emerge from the engineering process itself. Some of the mechanisms provided by database management systems to manage the influences of these factors in the changing software environment are discussed below.

MANAGING CHANGES

Engineering is a creative process resulting in numerous, intermediate by-products — sometimes called *engineering artifacts*. The activities involved in the creation of products build on each other and ultimately lead to a final end-product. This evolutionary nature of engineering can be greatly aided by database technology. Database technology can support the growth of a product with mechanisms to:

- store different versions of products or other artifacts,
- manage the evolution of the data model,
- monitor and control access to engineering data and systems, and
- integrate the information needs of multiple applications.

This section expands on these mechanisms and other considerations related to these aspects of change.

Support for Versioning

During the course of an engineering effort, different aspects of the final product change as it evolves. Alternatives are defined and analyzed to determine the best choices for the final product. These variations may also produce designs for different versions of the product. Changes usually

occur incrementally as designers explore alternatives. Therefore, recording the changes and the history of changes is an important database management service. The versions serve as a historical record of a product's design and may be useful at a later time for considering product and process improvements. (Dittrich, 1988) (Katz, 1990)

Some database management systems provide mechanisms to help manage versions of data[1] in the development process. These mechanisms typically include support for

- establishing and maintaining relationships between versions,
- controlling access to different versions, and
- notifying collaborators of changes to versions.

A versioning system tracks the evolution of a product by allowing engineers to create a sequence of distinct versions of a design or other aspects of a product. A database management system maintains an ordering between different versions. When a version is modified in two different ways, these alternatives can be saved as different versions creating a branch in the version hierarchy. Thus alternatives may be tracked using a version hierarchy. (Landis, 1986) (Chou, 1989) (Ideally an engineer's notebook would accompany the version hierarchy and provide the rationale for the final choices.)

Data management operations provided by a versioning system include the ability to create new versions and to retrieve, modify, compare, and delete versions. In addition to facilitate control of the engineering process, some versioning systems may allow different categories of versions to be created. For instance, a version may be classified as work in progress. These intermediate versions may be maintained privately by individual engineers and not shared with other engineers. When the work has progressed to a point where few changes are expected, a stable version may be created for access by others (see "Cooperative Engineering Environment: Concurrency Control"). Once no further changes are expected, a permanent or final version of the data can be made available for use in other activities. The database management system may control access to the different version categories (Chou, 1989).

Another important aspect of versioning concerns the association between two objects. When one object references another object, the reference may be to a specific, predetermined version of the second object or it may reference a default version of that object. Often the default version is the most recent stable version; however, sometimes, such as during preliminary design when many different alternatives are being explored, the most recent version of the object may not be the most suitable default. A versioning system may permit the engineer to determine or change the mode of reference in the class definition for the object.

In a cooperative engineering environment where data is shared by many engineers, the creation of a new version may require the engineers to be notified of the change. Approaches to change notification are divided into *message-based* and *flag-based.* Message-based notification requires that a

1. *Version* in the database management terminology is different from *product version* as defined in a conceptual schema. The concept of a version as managed by the database management system is independent of the business decision of when to create a new version of a product. A version in the database management system may be used to represent incremental changes on work in progress, regardless of whether a new version of a product is created for the purposes of manufacturing and marketing.

message be sent in the event of an update. The message may be sent immediately or deferred until a convenient time in order to minimize processing overhead. Flag-based notification sets a flag and notification occurs when the data is accessed. Change notification options may also be specified in class definitions. (Chou, 1989)

The maintenance of versions impacts the performance of a database management system. Versioning operations increase processing overhead and managing versions requires more storage space. Efficiently implemented versioning systems do not maintain a separate copy of data for each version — instead the system stores only the changed portion of data with appropriate pointers to elements common between versions.

As the product progresses through each stage in the engineering process, separate copies of the data may be stored and released as new versions. (For instance, a new version could be created when a design is sent from engineering to manufacturing.) Business practices and policies determine when product versions are created and how they are maintained. However, the versioning mechanisms provided by database management systems support these established policies. For further information on version modeling in engineering databases see Katz (Katz, 1990).

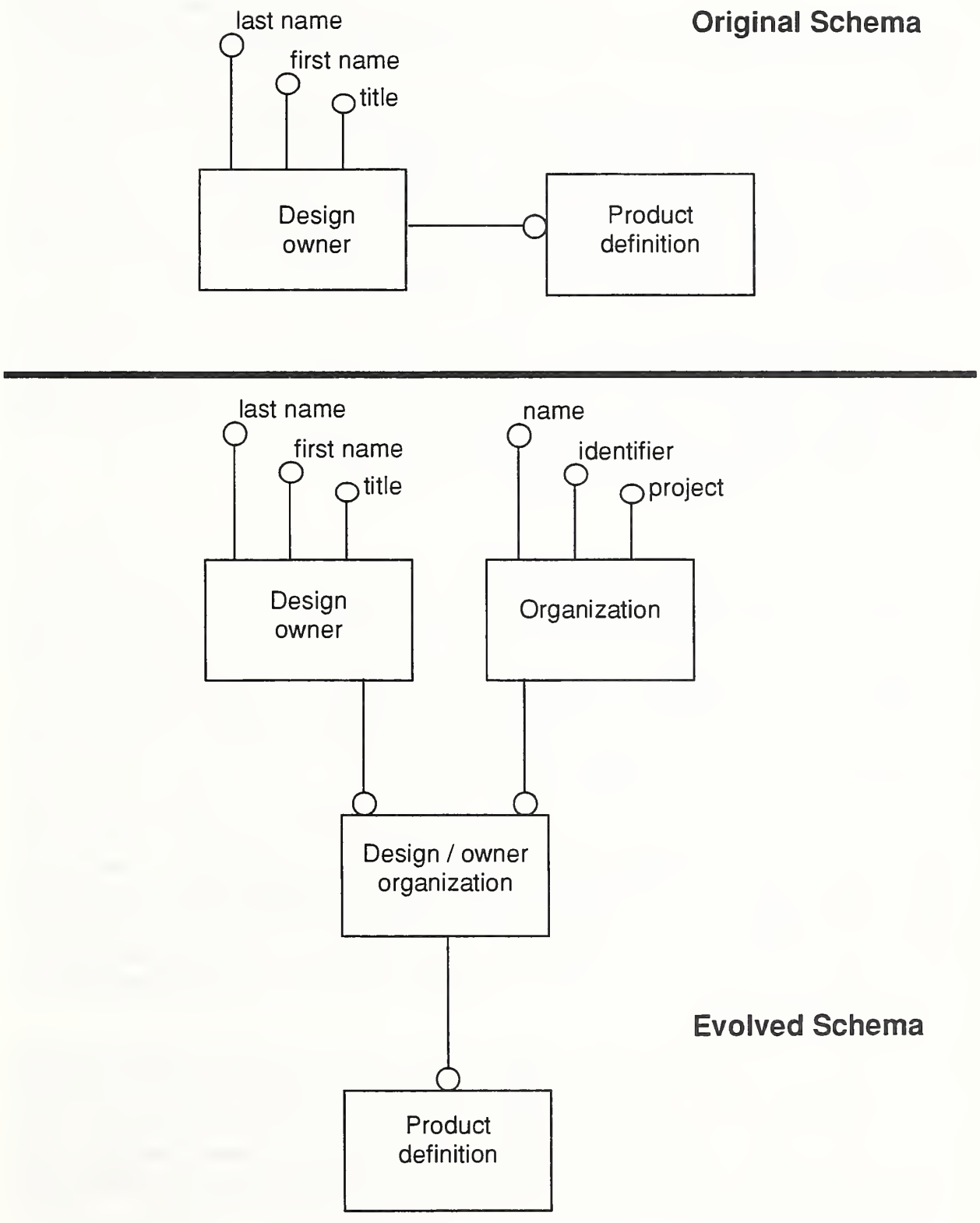
Schema Evolution

Engineering is an inventive process that requires flexibility from database management systems. Improvements to the engineering process or the introduction of new processes provide a competitive advantage in industry. However, these changes may require additional types of information or they may alter the way in which existing information is accessed. Thus the schema for an engineering application needs to evolve with engineering processes. The iterative nature of the engineering process involves specifying information requirements (building a schema), developing product data to support those requirements (populating the schema), and then further refining the requirements (evolving the schema). This last step is referred to as *schema evolution.*

The schema for the mouse casing product of the earlier example (**Figure 1**) assumes that a product definition has a single owner who belongs to a particular organization. The organization associated with the owner does not need to be explicitly stated in the schema since the organization is be the same for all owners. However, **Figure 4** illustrates what happens to the schema when, at some later time, other organizations become involved in product development. Under the new business practices, the owner of the product definition may belong to one of several organizations. In this case the entity *design owner* is no longer sufficient information and a new entity is needed to identify the organization. The *product definition* entity is also effected by the changes. The new schema incorporates the new aspect of the product.

Schema evolution is a significant challenge in the field of software engineering in general and raises additional considerations with respect to database management systems. A database management system relies on the definition of a schema both for managing data and for providing a logical view of the data to a user. Changes to the schema impact both the organization of the data within the system and the ability of the system to present the data to the user. In particular, database systems need mechanisms and policies for addressing the issues of both *data migration* and *application migration.* Data migration refers to the transformation of a data set to reflect changes

Figure 4: Schema Evolution



to the schema describing the data. Application migration refers to the management of the impact that schema evolution may have on associated applications.

Schema evolution and the related issues of data migration and application migration are supported to different degrees within a particular database management system. Generally the complete migration to a different schema requires human intervention. A challenge to database management systems is to provide facilities to support schema evolution by minimizing the amount of human intervention necessary without jeopardizing the correctness of the data. For instance, in the previous example, where an organization was added to the database, a database management system utility could be used to associate a single organization with all the design owners in the original database. However, if the design owners did not actually all belong to the same organization, not only would this strategy result in incorrect data, but the results of any queries involving the organization and the design owners would also be unreliable. While this approach would minimize the work needed to migrate the data to the new schema, it may not be desirable.

Analysis of the types of changes a schema may undergo is provided in several sources (Banerjee, 1987) (Spooner, 1989) (Kim, 1990) (Ahmed, 1991) (Kohout, 1992) (Zicari, 1992). Mechanisms for implementing schema evolution are dependent on the implementation of a particular database management system and are, therefore, beyond the scope of this article. However, the related issues for data translation and application migration are discussed below.

Data Migration

The primary ingredient needed for data migration is a specification of a transformation which to be performed on the data to make it compatible with the new schema. The transformations can be encoded in an algorithm provided by the database management system, or the database management system can provide a utility for the user to specify new data or an appropriate transformation algorithm (Kohout, 1992) (Clark, 1992). In the example in **Figure 4**, a suitable transformation would be to assign a default value to the organization associated with a design owner, since all owners already in the database belong to the same organization. A mechanism for implementing this transformation within the database is needed.

A secondary, but nevertheless important, concern is the timing of the transformations. There are two approaches to this aspect of data migration. The data set can be migrated all at one time; this approach is called *immediate*. In this case the transformations are applied in batch to the entire data set, or the transformations can be performed dynamically on an as needed basis (Zdonik, 1990) (Björnerstedt, 1989). The latter approach is called *deferred* and may be preferable if the data set is large, if only a portion of the data requires transformation, or if a user must provide input to the transformation.

In the schema evolution example, an immediate migration would associate the appropriate organization with all the owners in the database using a default value as described above. However, if the database contains designs for several years, which are no longer actively used and may never again be needed, then this effort would be unwarranted. A deferred approach is better but the use of a default value is not suitable using this approach, since a default value may result in incorrect associations for new owners.

Application Migration

Approaches to managing the impact of schema evolution on existing applications can be divided into two categories: those which identify the areas of impact, and those which hide the impact. Each of these approaches has advantages and disadvantages, and a robust system should support a combination of these solutions.

The first approach is characterized by CASE systems. This strategy uses a dictionary to track and identify which applications or parts of applications are affected by a change to the schema. The dictionary stores a mapping between applications, the conceptual design, and the individual classes in the database schema. Based on these mappings the dictionary can be used to determine which applications are affected by a particular change. For instance, the schema change described in the previous example should only affect those applications which use the entities *design owner* and *product definition*. In addition, the dictionary can also indicate the degree of the impact of the changes by tracking which particular attributes of an entity are used in which applications. In this example not all applications which use the changed entities are necessarily affected by the change; only those which involve the connection between design owner and product definition are impacted. The dictionary can also be used to evaluate the impact of a change in the conceptual design on the implemented database schema and thus to gauge the magnitude of the change.

The second approach of hiding the changes is characteristic of object-oriented systems. Using this software design principle applications do not directly rely on the data in the format that it is stored in the database. Access to the database is buffered by functions attached to class definitions called *access functions*. As a schema evolves, if the class hierarchy remains unaffected and the access functions do not change, then the applications are functionally unaffected by changes to the schema. In the example in **Figure 4**, if the class which represents product definition maintains access functions for the owner's name, then applications using the product definition class should not be affected by the change. However, major changes to the underlying data set may significantly affect performance; therefore, it may be desirable to change the applications. Note that this approach is only effective if the changes to the schema do not affect the existing class structure (Osborn, 1989) (Narayanaswamy, 1988). In relational database systems this approach is supported by mechanisms for defining views on the data.

An additional consideration for application migration is the extent of the impact of a change on application programs. Often changes to the schema will require one or more applications to be modified. In addition, the data may need to be reorganized on disk. In both these situations applications are unavailable until the migration is complete. Typically, the reorganization of data on disk affects all applications using the database.

The impact of schema evolution is buffered when the changes are hidden from applications through the use of access functions. However, changes to the underlying physical organization of data may even impact applications whose source code is not affected. From the application perspective, applications whose source code is not impacted by changes to the schema can be classified into three categories based on the degree to which they are affected by the changes. A *run-time* compatible application is not affected by changes to the schema. An application which is *link-time* compatible will need to be re-linked with the database system; however, the application's object code is not affected by the change. A *compile-time* compatible application must be

re-compiled before the application can reliably be used. The product development environment will be impacted for applications which fall into one of the latter two categories. In addition, even run-time compatible applications may be affected, if the database is shutdown during the migration.

Tools for Managing the Environment

Database technology facilitates control over the computing environment. Through the use of a database management system a certain amount of control over the environment is maintained through the mechanisms for data organization. Another application of database technology, *repository* technology, is evolving to further support control over the environment. Repository technology integrates several other technologies for information management, such as database management, CASE, object-oriented programming, and configuration control.

Repository technology is used to control a computing environment which may span many systems including databases and other engineering systems (Appleton, 1985) (Devlin, 1988) (Dywer, 1987). Repository systems may provide functionality, such as

- monitoring of the environment,
- impact analysis,
- process improvement, and
- workflow automation.

This technology will be essential to support interoperability between diverse software systems, including database management systems.

Repository technology has roots in database technology through the concept of a data dictionary. In 1977 an ANSI/SPARC database study group (Tsichritzis, 1977) described a framework for database systems. This framework described the role of a data dictionary as being essential to operating and maintaining a database system. In the ANSI report, a system is described in which two people are the focal points of the information management process: an enterprise administrator and a database administrator. The enterprise administrator manages a system's conceptual design, and the database administrator manages the database schema. In this framework these two people keeping track of all the information about the system's configuration using a data dictionary.

Since then, database systems and computing environments have grown much more complex. A sophisticated data dictionary is becoming an essential component to an effective computing environment. The data dictionary contains descriptive information about conceptual designs, as well as the correspondence between these designs and their implementations in individual database systems (see previous section "Schema Evolution: Application Migration"). In addition, the dictionary describes the software components of the computing environment, the relationships among the components, and the relationships between the components and their users. This information is used to control and regulate system configuration, versioning, security, protection, storage, access, schema management, and data traceability for all components of the environment.

In most existing application environments, the data dictionary is not automatically maintained, but software which supports automated maintenance has recently been made available. The early use of a data dictionary was to track the use of entities from a conceptual design within a database schema. Statistics derived from the dictionary could be used to manage changes to the schema, to optimize physical design, and to monitor usage requirements. However, the role of the dictionary has broadened. Data dictionaries are now used to control distributed computing environments consisting of a diverse set of software and hardware systems.

In the engineering environment a data dictionary is not only necessary, but it is also important that the dictionary be *active*. In other words, the dictionary should contain the information needed to control the environment that is supported by software which is ultimately responsible for controlling the environment. An active dictionary guides the work flow through an organization's islands of automation. The dictionary maintains the location of software and data within the distributed environment. In addition, it maintains the process information needed to structure the work flow through the environment.

Techniques for Schema Integration

Tools which manage the complexity of the computing environment provide mechanisms to support system interoperability; however, for systems to be fully interoperable, common terminology must be available for sharing data. Techniques for schema integration are used to manage and integrate diverse applications. The techniques provide a mechanism for understanding the information needs of multiple applications, such as the applications involved in the re-engineering of the mouse casing as describe earlier (see "Nature of Engineering Data: Conceptual Design: Application Integration.") In that example, in which the left mouse button was prone to breaking, three applications needed to share data in order to correct the design flaw.

Schema integration techniques are used to produce an integrated conceptual design based on an analysis of the semantics of the data used in different applications (Batini, 1986). The integrated conceptual design contains the common terminology needed for interoperability between applications. The design typically serves as the basis for the schema of a shared database. However, researchers are currently investigating mechanisms to use such a design as a basis for interfacing multiple database systems (Breitbart, 1990) (Krishnamurthy, 1987) (Litwin, 1990).

Semantic analysis, which leads to a common terminology captured in a conceptual design, may be divided into five steps (Batini, 1992):

- application schema definition,
- conflict identification,
- discovery of inter-schema relationships,
- schema restructuring, and
- functional evaluation.

Application schema definition results in the conceptual design corresponding to a particular application. In order to understand the information requirements of each application, a separate conceptual design may be defined for the applications that are to be integrated. These designs, or

application schemas, provide a means of communicating and understanding the information needs for the different applications. The designs also define the scope of product data needed by a particular application.

The next two steps, conflict identification and discovery of inter-schema relationships, identify the areas of overlap in the data needed by the different applications. The data within these areas must be clearly and consistently defined in order to be shared by the different applications.

Conflict identification uncovers discrepancies in the way the same data is defined by different applications. Conflicts can be divided into two categories: name conflicts and structural conflicts.

Name conflicts are the result of either synonyms or homonyms between the applications. The names used to refer to the same real-life concepts are often different between applications. In the example in **Figure 5** the design application uses the term *material type* to designate requirements for a material (e.g. heat-resistant, fire-retardant,....) based on the product's intended usage. Whereas the material selection application uses the same term *material type* to distinguish whether the material is available locally or must be imported. For the two applications to share data common terminology is needed.

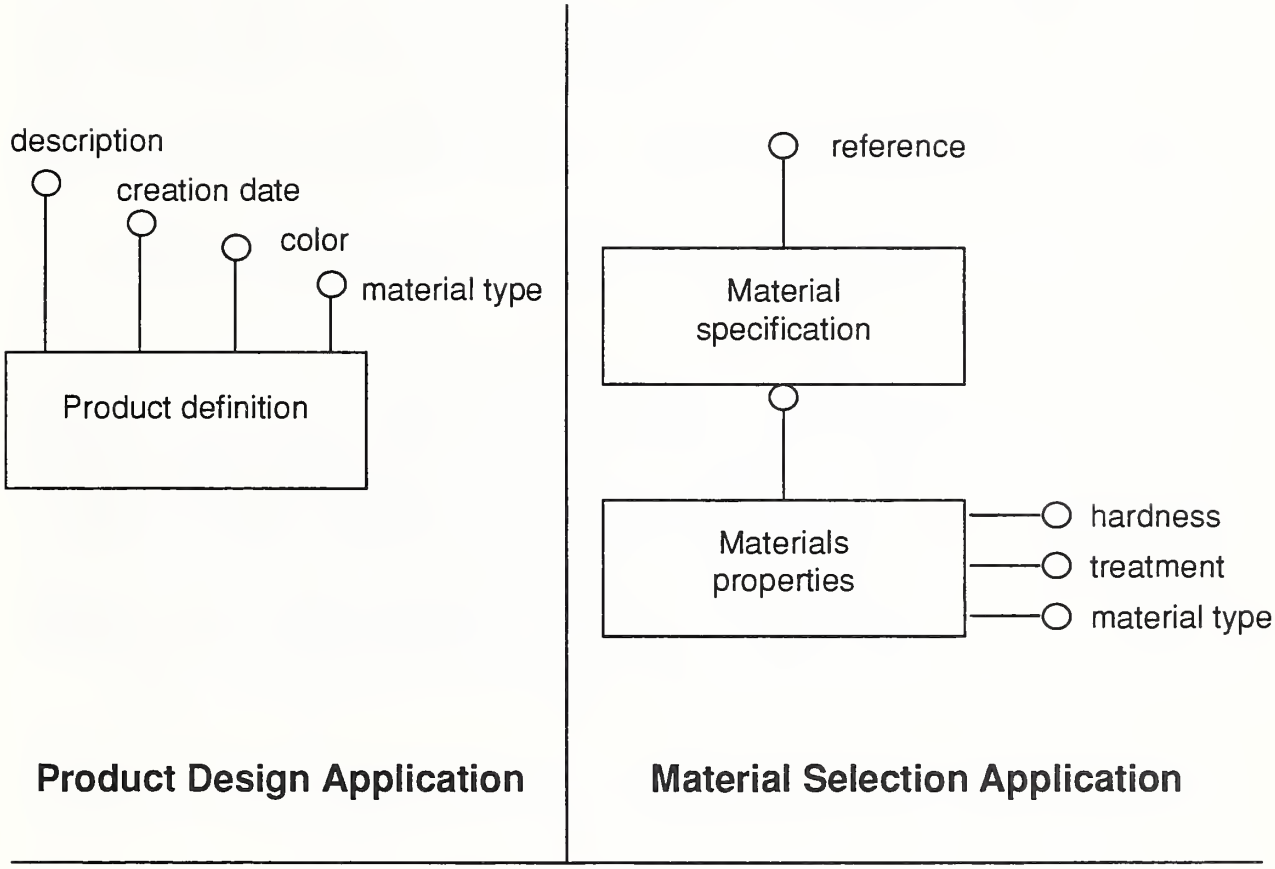
Structural conflicts result from the same concepts being used in different ways. For example, in the design application an enumeration of values may be associated with the valid values for *material type*. Based on the intended usage of the product, these values provide guidelines for the material to be used in the product's construction. The analogous concept in the material selection application will require more complex data involving several values. The material selection application calculates a material's thermal thresholds and pressure tolerances to determine the suitability for its intended usage given certain environmental conditions.

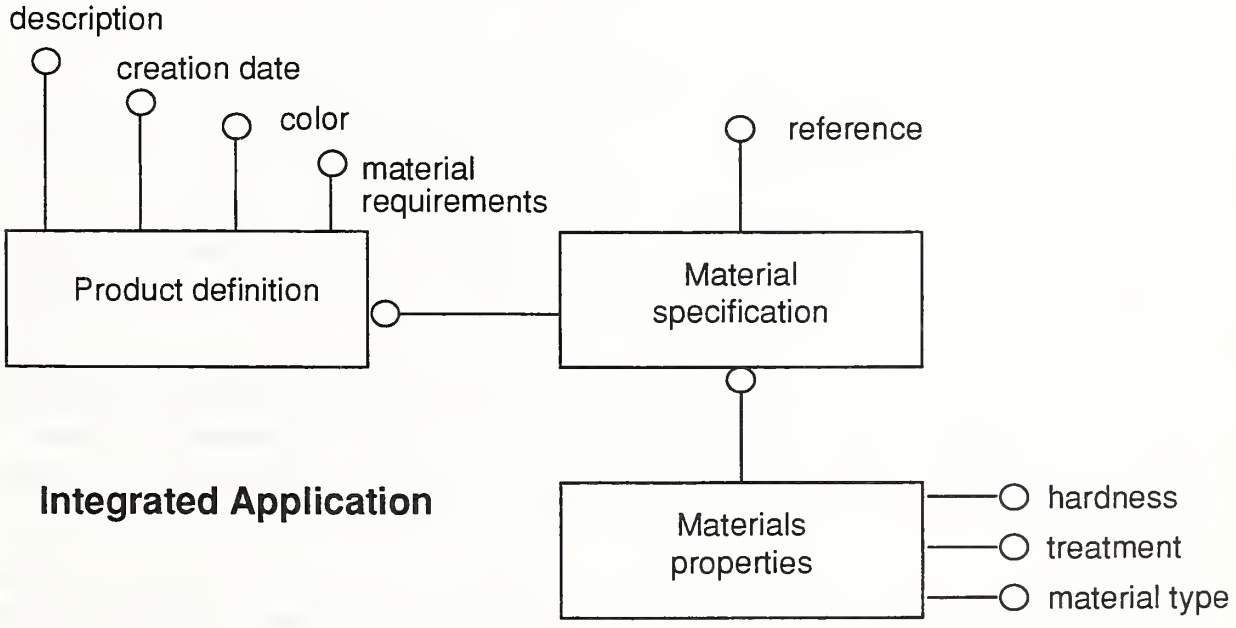
The discovery of inter-schema relationships often begins with conflict identification. Often conflicting elements are strong clues of inter-schema relationships. Sometimes inter-schema relationships are appropriate but are not pre-defined by either application. For example, an enterprise-wide identifier for a product may not be required by the applications under consideration; however, such an identifier is extremely useful for coordinating different applications.

Schema restructuring combines the conceptual designs from different applications into a single design. Conflicting aspects of the individual designs are resolved by defining a common terminology and structure for the data to be used by the applications. Inter-schema relationships are explicitly represented as concepts in the new design. The constructs described above for conceptual design are very useful in restructuring. Specifically, generalization allows shared concepts to be represented abstractly but also allows them to be specialized for the different applications.

Finally, a functional evaluation of the integrated schema assures that the information needs of the individual applications are supported by the integrated model (Mitchell, 1991). A reliable method for evaluating the integrated schema is to demonstrate that the information needs of the individual applications are fulfilled by the new schema. The new schema should be able to provide all the data that was available in the independent schemas.

Figure 5: Schema Integration





Schema integration techniques assist in the management and integration of diverse software systems. Combined with the mechanisms described earlier for schema evolution these techniques may be used to introduce new applications smoothly into a software environment that uses a shared database. The consolidation of many applications in a single database implementation reduces data redundancy and inconsistency by allowing the applications to work from the same set of data. However, the use of a centralized database also requires more cooperation to manage the sharing of data. Database management facilities to support effective cooperation in this environment are evolving. Several effective techniques which already exist are discussed in the next section.

COOPERATIVE ENGINEERING ENVIRONMENT

Engineering is a cooperative process in which work is coordinated over an extended period of time. In the course of a development effort, individual aspects of the product may be worked on concurrently by several engineers or engineering teams. This situation is known as *concurrent engineering*. Even when designers develop separate components, shared access to parts of the design is required due to the interdependencies within a complex product. A database management system allows multiple engineers to share information in a manner that preserves the consistency and correctness of the underlying data. Traditional database management systems rely on established techniques for controlling data sharing. For the engineering environment, these techniques may be modified and supplemented with additional methods. In particular, concurrency control and data distribution need special consideration in the engineering environment.

Three types of situations arise in an engineering environment in which data needs to be shared:

- within a work group of engineers,
- across the full product life cycle, and
- between enterprises involved in joint development.

An engineering work group is a small, highly specialized team whose members work together on a specific area of product design such as solid modeling, finite element analysis, or printed circuit board layout. These engineers are typically equipped with personal workstations. The team needs an environment where data is stored and managed as a single unit and is accessible at any time by any application. The data may be partitioned, replicated, and physically distributed across several different systems. However, design changes must be maintained centrally so that new versions of the design are easily and quickly available to all team members. In addition, the most current set of data should be available to the team members responsible for releasing a new version of a product design to other teams who may be working concurrently on different aspects of the product. Team members need to work independently without concern for whether others might duplicate or do work in conflict with their work. These engineers may need to manipulate and navigate through entire designs or portions of designs. The techniques for concurrency control, described below, are particularly useful for engineering teams.

Across the full product life cycle various releases of the data associated with a product are needed during different stages of the development process. The typical process stages where data sharing is crucial and distribution is inevitable include, but are not limited to: 1) moving from conceptual

product design to detailed product design, 2) moving from design to manufacturing, and 3) moving from first prototype manufacturing to volume production. In addition, improvements to a product's design maybe introduced as the design is refined in the later stages of development. These changes should be available for the next iteration of the product's design. Thus, data also needs to flow backwards through the product development cycle.

This environment poses special requirements for management of data security, application work flow, work planning, configuration and version control, product release, and other issues related to product development and life cycle. Other considerations for sharing data across the product life cycle are the number of users and variety of applications sharing the data. To support the great diversity of development environments involved in creating a product, techniques for managing distributed data are being develop.

Enterprises involved in joint development projects are characterized by vendor/supplier and contractor/sub-contractor relationships or collaborative development between different companies or organizations. Within these projects data needs to be integrated both vertically (up and down a single business organizational chain) and horizontally (between different organizations). Database management systems provide facilities for making this possible.

Some of the ways that database management systems support cooperation in these environments are described in the following sub-sections.

Concurrency Control

One of the primary functions of any database management system is to regulate the concurrent use of data in a meaningful way. The concept of *transaction* is central to this management. A transaction refers to a logical unit of work consisting of a series of operations, or *updates,* to be performed on the data. The set of updates in a transaction must be completed as a unit for the data to remain logically consistent. Updates are *committed* — saved permanently in the database — or *aborted* — the data remains unchanged within the database. To prevent the introduction of inconsistency to the database, the updates within a transaction are made indivisible or *atomic*--either all updates are committed or all are aborted.

In a multi-user database environment, many transactions may execute concurrently. If access to data is unregulated, the actions of two transactions may interfere with each other and result in inconsistent changes to interrelated parts of the database. To allow data to be shared and prevent such errors from occurring, database systems manage the parallel execution of transactions through concurrency control protocols. Concurrency control protocols prevent these errors by causing coinciding transactions to perform their actions in a serial, or consecutive, order. (A group of transactions executing concurrently is said to be *serializable* if their order is equivalent to a serial order of execution.) These protocols may vary considerably and involve various techniques that differ in terms of the level of data consistency they enforce, the extent of data sharing they permit, and the speed at which they execute.

Perhaps the most widely used method of concurrency control is based on locking of data items. A lock is placed on data to restrict access so that multiple transactions do not interfere with each other. Two basic kinds of locks exist: *read* and *write* locks. Read locks are shared. They can be

held by several transactions and allow data to be read but not changed. Write locks are exclusive. They are held by only one transaction and allow data to be inserted, deleted, or modified. An *unlock* operation releases the locked data item for use in other transactions. Locking promotes consistency of the database, but it limits the extent of sharing since a transaction may need to wait to access an item which is already locked by another transaction.

Another consideration for concurrency control is the scope of data which is locked. Traditionally, locks are implemented by database management systems either at the logical level by locking all the instances of a class as a whole or by locking a single instance within a class, or at the operating system level by locking the pages on which the objects are written. The breadth of the data covered by a transaction in traditional systems is confined to those units. However, these units for locking are not as useful in the engineering environment. Consequently, alternative mechanisms are being explored for locking sets of inter-related instances which may span multiple classes.

One well-known concurrency protocol is the *two-phase locking* protocol (Date, 1985) (Elmasri, 1989). In this protocol transactions lock data items to restrict access by other transactions. Serializability is achieved by requiring that a transaction lock all its desired items prior to unlocking any items. While this leaves the database in a consistent state, transactions must often wait to lock all required data items. Because the two-phase protocol is based on the assumption that conflicts are likely to occur, it is sometimes referred to as *pessimistic* concurrency control. In heavily used systems, delays can be frequent and lengthy. For the purposes of cooperative engineering, the two-phase locking protocol is too limited because of the complex and extensive interrelationships between the classes in the database schema.

To increase the extent of data sharing, other protocols must be employed. *Optimistic* concurrency control (Kung, 1981) assumes conflicts are unlikely and permits transactions to access and modify data in a temporary work space without placing locks. When the transaction attempts to commit its changes, a check is made if other transactions have simultaneously modified the same data items. If a conflict is discovered, one or more transactions may have to be delayed and rescheduled later. While the optimistic protocol can improve performance because it eliminates the overhead associated with locking, high volumes of transactions can result in frequent delays. In an attempt to overcome the shortcomings of both optimistic and pessimistic approaches, other protocols such as the semi-optimistic and mixed mode have been proposed. These protocols are currently being developed.

The extent of sharing also can be increased by using versioned objects (see "Support for Versioning"). Two or more transactions may access different versions of the same object. The database management system may determine which version of an object should be used by a transaction. When the selection of a version is the responsibility of the database management system, the selection imposes additional processing overhead.

Long duration transactions (Korth, 1988) together with *private databases* are used for engineering systems to support prolonged activities by teams of engineers. These techniques constitute significant advances in database technology, but also present new problems (Ranft, 1990). In contrast to the short transactions in traditional database management systems which are intended to last for seconds or minutes, long duration transactions can last for hours, days, or longer.

The use of private databases permits portions of a product's data to be isolated in a smaller, more exclusive environment, such as a local workstation, where a design team can work in isolation using either short or long-duration transactions. Portions of a larger, central database are *checked out* and place in a private database for access by a limited number of users. After work is complete, the data is returned or *checked in* to the larger database. A lock may be placed in the central database on data that has been checked out. A write lock ensures that no changes are made to the data which is checked out, but does not prevent others from seeing the central data. Combined with versioning, checkout procedures allow parallel work on a single aspect of a product's design.

Concurrency control methods for cooperative engineering are still evolving. While the techniques described in this section support cooperative processing to a significant extent, other concurrency control protocols are expected to emerge that will more completely meet the needs of the engineering community.

Data Distribution

The technology for managing data in a distributed working environment is extremely important for engineering (Krishnamurthy, 1987). In engineering and manufacturing applications data needs to be available at many physically distributed sites throughout a product's life cycle. Engineering involves a multitude of information sources and requires the coordination of databases from different disciplines, often comprising a vast number of manual or automated information processing activities. Two aspects of data distribution which are particularly significant in engineering environment are

- the availability of data throughout the engineering process and
- control over data during the process.

In a distributed working environment data or other components of the software environment often exist on more than one computer systems. Distributed computing systems with databases at multiple sites are connected in many different ways (Ceri, 1984). Data and other resources may or may not be duplicated at many different sites, and the individual database management systems may be maintained and controlled by different mechanisms and administrative policies. *Heterogeneity* — the dissimilarity between the separate sites of a distributed computing system — is determined by the combination of hardware, operating system software, and database management systems. In a totally homogeneous environment, all of the system components are the same at all sites.

The wide range of variation between computing systems in a distributed environment presents a significant challenge for database technology (Gupta, 1989). The environment variables, which create difficulties in sharing data, range from different machine architectures to inconsistent terminology used by different applications. Database management systems interface to complex networks. Databases integrate different applications' views of data. These capabilities make data available in a controlled manner to the many sites involved in a product's development. Current systems support these needs to a limited extent; however, the computing standards that will enable a broader support for distribution are emerging (see the next section on "Standard Interfaces").

Availability of Data

In an engineering environment it is important that data is available when and where it is needed. As a product's development progresses, data is needed in different physical locations at different times. Many of the data sources are files, older database management systems, or stand-alone application systems such as Computer-Aided Design packages. Such systems need to be integrated into the distributed computing environment. Often a large amount of data is transferred between engineering groups. In order to make efficient use of the resources, engineers often need to request data in advance. Advance preparation assures that all the necessary data will be available at their work station when they are ready to work on a task.

Database management systems are developing ways of shortening this preparation time. Mechanisms for reducing the effect of two factors that significantly influence the availability of data — the location and the replication of data within the software environment — are being developed.

An important characteristic of a distributed environment is the level of visibility of the location of the data in the system. As the visibility level decreases, the engineer needs to know less about where data resides. *Location transparency* refers to the situation in which the engineers are unaware of the physical location of the data, and a single command can access the data from multiple databases at different sites. For tightly coupled work, such as within a single engineering work group, location transparency is highly desirable. As the need to share data expands to a broader community, location transparency becomes more difficult to achieve and, perhaps, is even less desirable because of security considerations. However, even within the broader community, the database management system can be used to track the location of data within the system and to migrate data between locations.

Another important characteristic of a distributed environment is the level of visibility of replicated data. *Replication transparency* refers to the fact that an engineer is unaware of the data being replicated in separate databases. The engineer may treat a replicated data item as if it were stored as a single data item in a single database. The replication, however, needs to be coordinated and controlled by software and database administrators to ensure consistency between the sites. The database management system may use replication to increase the availability and reliability of data within the system. Data replication of this type that is used exclusively by the database management system should be completely transparent to the engineer and controlled by the database management system and system administrator.

Database techniques for versioning, long duration transactions, and private databases allow the user to control data distribution and replication within the software environment; however, additional policies are needed to determine when and where they are appropriate.

Control over Data

Another important characteristic of a distributed environment is the establishment of policies for accessing and updating data (Fong, 1988). In an engineering environment data often needs to be distributed among different projects, and shared data needs to be made available to several projects. For example, an engineer performing structural analysis of a product would need access to the complete set of material properties of the product. An engineer working on the assembly of

the same product would need access to a few, but not most, of the material properties. In order to manage this environment, a control architecture needs to be established. Coordination and control over data may be either centralized, where all decisions for managing the distributed data are made by a system-wide data administrator, decentralized, where each local site exercises its own policy for managing its own database, or a combination.

Distributed database management systems support control over data through schema integration strategies which reflect and support the control architecture. A common classification of schema integration strategies is based on the autonomy of the components (Breitbart, 1990) (Heiler, 1989). Different strategies address different levels of integration of the system components and different levels of global services. The approaches to integration may be divided into two categories based on the source of control over the data. The first category uses a *global schema* to coordinate data management between systems; the second category uses a more loosely defined *federated schema*. A global schema approach to system integration creates one logical schema to which all the participating systems must comply. Using the federated approach the distributed systems maintain more autonomy, yet participate in a federation to allow partial and controlled sharing of their data. (Heimbinger, 1985)

Whether the distribution strategy is federated or not, the distributed architecture can have different variations (Sheth, 1990) (Fong, 1991). The most common distributed architecture is the client-server computing model. The node that makes the database access request is referred to as a client node, and the node that responds to the request and provides database services is referred to as the server node. Using the client-server architecture the server node maintains control of shared data while much of the processing and private data are distributed to the client machines. Client-server architectures can be configured in a variety of ways ranging from single client and server to multiple servers supporting many clients. The various configurations and associated component autonomy affect all aspects of the availability of and control over data.

The client-server architecture creates special considerations for cooperative engineering. In particular for managing shared data, locks need to be consistently maintained across client and server nodes. Since much of the data actively being used is distributed to client machines, the database server needs to notify the clients of lock requests and updates to the data. An additional consideration involves the reliability of data if the server crashes or if access between the client and server is disrupted. Some database management systems store locks as persistent objects in the database so that they can be restored if the server crashes; however, this strategy incurs a significant processing overhead that is often unnecessary.

CONCLUSION

Generalized database management systems have been in commercial use for two decades, but they are only just beginning to be widely applied in engineering and manufacturing environments. Research and development efforts in the data management arena continue to produce new capabilities and products that are useful for engineering applications. Standards needed for systems to interoperate in a cooperative engineering environment are also emerging. Future database management systems will provide mechanisms for sharing data about a product throughout its lifecycle.

State of the Art of Commercial Database Products

In the 1980s relational database technology emerged. Relational systems decoupled the logical format of data from its physical representation on disk. SQL (ISO/IEC 9579, 1991), based on the relational approach to database management, emerged as the first widely accepted standard database language and is widely used in business applications. This standard continues to evolve and future versions are planned which will support more advanced capabilities. The existence of a database language standard such as SQL is important for the widespread acceptance of database technology. A standard provides a basis for testing database management systems for reliability and security (see following section on "Database Management System Standards". This level of maturity in a software system is needed before these systems can become trusted components in the workplace.

In the last decade several object-oriented database management systems, based on object-oriented techniques, have been commercialized (Dabrowski, 1990). Surveys of object-oriented database products and prototypes can be found in several sources (ANSI-OODBTG, 1991) (Cattell, 1991) (ACM, 1991) (Ahmad, 1991) (Wood, 1992). Object-oriented database management systems are designed to support engineering applications by providing many of the features described in this article, such as richer data modeling techniques and more sophisticated versioning and transaction capabilities. However, these systems have not yet reached the level of maturity of their predecessors. Standards for these systems are under development. These standards will provide a basis for testing their reliability and serve as a basis for widespread introduction into the workplace.

Many of the object-oriented database management systems combine the characteristics of object-oriented programming with database management techniques (Zdonik, 1990). Two approaches to developing object-oriented database management systems are

- to add database management system services to an object-oriented programming language and
- to extend conventional database management systems (such as relational) to support a broader range of functions.

Many commercial object-oriented database management systems available today have added database services to an object-oriented programming language such as C++, common LISP, or Smalltalk. While some of these systems do not provide much more than persistent data storage, other systems include support for transaction management, concurrency control and recovery, query languages, and performance techniques.

The approach of extending the conventional database management systems has focused on supporting a broader range of built-in data types and graphical user-interface tools. These systems have the benefit of being based on more mature software.

Although some commercial object-oriented database management systems are available and many more are emerging, this technology is still in the process of rapid development and is likely to remain so for at least the remainder of this decade. Many of the commercial systems are suitable for use within a work group or by isolated applications. However, many of the issues involved in large-scale systems continue to be topics of intensive research.

Standard Interfaces

Simultaneous with the development of database technology has been the development of standards. The infrastructure of standards needed to support the sharing of data across the broad range of people and systems involved in a product's development is emerging. The trend in production computing environments is to integrate database systems with sophisticated end-user tools for managing a cooperative engineering environment. This trend requires interfaces between a wide variety of software tools such as database management systems, graphical user interfaces, CASE and other software development tools, information resource repository systems, and software libraries in the form of mathematical subroutine packages, finite element analysis packages, and geometry handling packages. Standard interfaces are essential for such an environment.

Today computer-aided engineering tools and database management systems, typically work in isolation from each other. Each tool operates in its own self-contained environment. Integration of tools and database management systems requires standard interfaces. To piece together a complete suite of software tools for engineering, standard interfaces are essential. These standards development activities are supported by numerous consortia of industrial corporations and government sponsored activities (Congress, 1992).

The computing standards relevant to engineering systems can be divided into two categories: *open systems* and *product data* standards. Open systems standards provide the capability to make connections, find information, and transmit data across the network without regard for vendor-specific hardware and software architecture and implementations (Boland, 1991) (Rose, 1989) (Stallings, 1990). Open system standards cover a broad range of computing technology. A subset of the open system standards specifically address database management systems. Product data standards provide mechanisms for communicating about the meaning and intended use for engineering data (Carver, 1991). Both types of standards are necessary to fully automate the software environment. A discussion of the aspects of these standards which specifically address data management for engineering follows.

Database Management System Standards

SQL (once referred to as Structured Query Language) is a national and international standard language for defining and manipulating tabularly structured data[1]. It provides portability and interoperability of database definitions and database application programs among conforming implementations (ISO/IEC 9075, 1992). The first SQL standard, in 1986, provided basic language constructs for defining and manipulating data that is structured as tables. The initial SQL standard focused primarily on satisfying the needs of business applications and never gained widespread acceptance in the engineering community. Currently, the SQL standardization committees are focusing on future extensions for meeting the requirements of engineering applications. These extensions include many of the features described in this article.

---

1. SQL is a project of the ISO/IEC JTC1/SC21 — Joint Technical committee on Information Technology (JTC1) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), Subcommittee on Information Retrieval, Transfer and Management for Open Systems Interconnection (SC21) — and the ANSI X3H2 Database Languages.

Remote Data Access (RDA) is an emerging standard protocol for establishing a remote connection between a database client and a database server (ISO/IEC 9579, 1991). The goal is to promote distributed processing in a client/server environment. The specification is specified in two parts, a Generic RDA for arbitrary database connection and an SQL specialization for connecting databases conforming to database management system with SQL.

Many groups are working to identify potential functional standards for object-oriented technology. The groups are coming together under the umbrella of a new ANSI technical committee on Object Information Management (ANSI-OODBTG, 1991)[1]. The ANSI committee was formed to develop a reference model for object-oriented systems and to identify object-oriented services that are suitable for standardization.

Information Resource Dictionary System (IRDS) is an emerging standard that supports repository technology[2] (ANSI X3.138 IRDS, 1988) (see the section on "Managing Changes: Tools for Managing the Environment"). The services of the IRDS consist of utilities and systems necessary to catalog, document, manage, and use information such as that contained in a conceptual design and/or database schema. The IRDS standard specifies interfaces for a dictionary system for recording, storing, and processing descriptions of an organization's software resources. The first version of IRDS is designed to support data administration functions. The next revision to the IRDS specification is expected to support communication of information between applications and other data management tools.

Engineering Data Standards

Several engineering data standards are emerging. These standards define structures for exchanging and sharing engineering related data based on its semantics. Some of the larger activities are described below; however, similar activities are on-going in these and other domains.

Two widely used standards for the exchange of CAD drawing and related information are DXF (Autodesk, 1992) and the Initial Graphics Exchange Specification (IGES) (IGES5.1, 1991). DXF is a de-facto industry standard used to exchange two-dimensional geometry. IGES supports the exchange of more complex two-dimensional and three-dimensional line geometry, as well as non-geometrical information. IGES-based CAD data transfer is limited. An additional mechanism is needed to share and integrate a broader range of product information.

The emerging, international Standard for the Exchange of Product Model Data (STEP) (ISO1, 1992)[3] developed as an outgrowth of the IGES standard. The United States' effort in support of the international standard is called PDES, Product Data Exchange using STEP.

STEP addresses the need for communicating product information at all stages in a product's lifecycle, covering all aspects of product description and manufacturing specifications. The funda-

1. The Object Information Management Technical Committee is ANSI X3H7.
2. IRDS is a project of ANSI's X3H4 Technical Committee.
3. The Standard for The Exchange of Product Model Data (STEP) is a project of the International Organization for Standardization (ISO) Technical Committee on Industrial Automation Systems (TC184) Subcommittee on Industrial Data and Global Manufacturing Programming Languages (SC4). For an overview of the standard refer to *Part 1: Overview and Fundamental Principles* [ISO1].

mental components of STEP are conceptual models of product information and standard mechanisms for sharing information corresponding to such models. STEP includes the definition of a conceptual modeling language called Express (ISO11, 1992). The first version of the standard uses exchange files for sharing information about products. However, an effort is now underway to define a standard mechanism for sharing such information more dynamically using database technology. In particular, an interface for accessing an engineering database, called the Standard Data Access Interface (ISO22, 1992), is being developed as part of this standard.

Since STEP is an outgrowth of the IGES standard, the focus for the initial version is on mechanical products. However, future versions also will address other domains and a broader span of applications than are covered in the initial version. In particular, efforts are underway to extend STEP to support electrical products. Several existing standards for electrical products will need to be harmonized.

One of the most established standards for electrical products is Electronic Design Interchange Format (EDIF) [ANSI/EIA-EDIF]. The EDIF standard is a format for describing patterns for semi-conductor chip fabrication. It supports two-dimensional graphics and interconnection information for integrated circuits and printed circuit boards.

The Future

Databases will eventually serve as the backbone for an automated product development and manufacturing environment. However, many advances in the technology and standards to support the technology are needed to fully automate the environment. Much of the functionality described in this article is currently available in proprietary or commercial systems; however, robust implementations incorporating many of the approaches are only just emerging.

In order for database systems to maintain and integrate large quantities of information from numerous organizations involved in a product's development, advances in data management techniques and hardware are still needed. Some recent developments in the database area focus on knowledge representation techniques to support reasoning, process control techniques to allow access to systems in a distributed environment, and improvements in hardware for higher reliability and speed of access.

Knowledge-base management system technology combines reasoning capabilities from the artificial intelligence area with database technology (Brodie, 1986) (Ullman, 1988). One thrust of the technology is to develop mechanisms for reasoning based on data stored in a database. Researchers in this area are investigating how to represent and organize the information needed to extract hypotheses or conclusions from data and how to capture and present reasoning to provide expla-

nations in support of these conclusions. Research in knowledge-base systems is leading to techniques for

- reasoning based on traditional data types and relationships, and non-traditional types of data such as spatial, temporal, auditory, and visual;
- knowledge acquisition so that knowledge will be stored along with data during a development project;
- natural language interfaces which support dynamic explanations of the reasoning process and more flexible interactions between the computer system and a human operator; and
- reasoning based on incomplete, uncertain, or contradictory information.

Process control techniques are important for integrating diverse systems to fully automate the product development process. Issues involved in this area relate to inter-process communication and system security.

Developments in hardware technology for database systems are addressing the needs for very large databases and for access to archival data. Investigations in this area include the use of parallelism and optimization techniques for searching, updating, and scanning data. Techniques for providing faster access to data by anticipating an engineer's information needs and for better access strategies using parallel processing are also being explored.

Summary

The engineering process is complex and the software needed to manage it is equally complex. However, the application of software to engineering problems is essential to providing an efficient development environment for complex products. Industry is recognizing information management systems as one of its most valuable resources. Database management systems play a key role in these environments with mechanisms for providing data to the right people at the right time. The needs of engineering applications are emphasized in the newly emerging generation of database management systems and are a primary consideration for the development of future generations. In addition, the next generation of database management systems will be developed in the light of the need to integrate computing systems and, thereby, provide a smooth operating environment for tomorrow's engineers.

Today's database management systems provide mechanisms to manage engineering data. The data representation capabilities of these database management systems allow engineering systems to begin to manage data at a logical rather than physical level. Repository technology provides a means of managing product data and other software in the environment. Advanced repository systems will also provide additional capabilities to manage changes within the engineering environment and to provide the control needed to support engineering teams and cooperative development.

Research and standards development is on-going in the areas needed to support full-scale integration of the computing environment for manufacturing and engineering. The computing infrastructure needed to take advantage of these future systems is being developed today.

REFERENCES

1. S. Ahmed, A. Wong, C. Sriram, R. Logcher, "A Comparison of Object-Oriented Database Management Systems for Engineering Applications" in the *Proceedings of the 7th Conference on Computing in Civil Engineering,* American Society of Civil Engineering, Washington, D.C., 1991.

2. (ANSI/EIA-EDIF) American National Standards Institute, Electronic Industries Association, *Electronic Design Interchange Format*, Document ANSI/EIA-548-1988, New York, 1988 (future versions are due out in 1992 and 1993).

3. (ANSI-IRDS88) American National Standards Institute, *Information Resource Dictionary System*, Document ANSI X3.138-1988, New York, 1988. Also available as Federal Information Processing Standards FIPS 156, April 1989.

4. (ANSI-OODBTG91) American National Standards Institute, Information Processing Systems, Database Systems Study Group, Object-Oriented Databases Task Group (ANSI X3/DBSSG/OODBTG), *Object-oriented Database Task Group Final Report*, September 1991. D.

5. Appleton, "The Technology of Data Integration," Datamation, pp. 106-116, Nov 1985.

6. Association for Computing Machinery, *Communications of the ACM — Special Issue: Next Generation Database Systems*, vol. 34, no. 10, October 1991.

7. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik, "The Object-Oriented Database System Manifesto," in W. Kim et al (eds.), *First International Conference on Deductive and Object-Oriented Databases*, 1989.

8. Autodesk, Inc., *DXF: AutoCAD Release 12 Reference Manual*, Aug 1992.

9. F. Bancilhon, P. Buneman (eds.), *Advances in Database Programming Languages*, ACM Press, Addison-Wesley Publishing Company, New York, 1990.

10. F. Bancilhon, C. Delobel, P. Kanellakis (eds.), *Building an Object-Oriented Database System*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

11. J. Banerjee, W. Kim, H. J. Kim and H. F. Korth, "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", *Proceedings of the ACM SIGMOD Conference*, June 1987.

12. C. Batini, S. Ceri, S. B. Navathe, *Conceptual Database Design*, Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1992.

13. C. Batini, M. Lenerini, S. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, vol. 18, no. 4, Dec 1986.

14. E. Bertino, "Object-Oriented Database Management Systems: Concepts and Issues," *Computer*, April 1991.

15. A. Björnerstedt, C. Hultén, "Version Control in an Object-Oriented Architecture", in (Kim, 1989).

16. T. Boland, (ed.) "Working Implementation Agreements for OSI Protocols," GOSIP 3 reference document, National Institute of Standards and Technology OSI Implementor's Workshop, National Institute of Standards and Technology and IEEE Computer Society, December 1991.

17. A. Borgida, "Features of Languages for the Development of Information Systems at the Conceptual Level," *IEEE Software*, vol. 2, no. 1, January 1985.

18. Y. Breitbart, "Multidatabase Interoperability," *SIGMOD Record*, vol. 19, no. 3, 53 - 60, (Sept. 1990).

19. M. L. Brodie, "On the Development of Data Models," in *On Conceptual Modeling* (Brodie, 1984).

20. M. Brodie, J. Mylopoulos (eds.), *On Knowledge Base Management Systems,* Springer-Verlag, Inc., New York, NY, 1986.

21. M. Brodie, J. Mylopoulos, J. Schmidt (eds.), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages,* Springer-Verlag, New York, NY, 1984.

22. J. V. Carlis, S. T. March, "A Computer-Aided Physical Database Design Methodology," *Computer Performance*, vol. 4. no. 4., December, 1983.

23. G. P. Carver, H. M. Bloom, *Concurrent Engineering Through Product Data Standards*, NISTIR 4573, National Institute of Standards and Technology, Gaithersburg, Maryland, May 1991. Also in (Jackson, 1992).

24. R. G. G. Cattell, *Object Data Management*, Addison-Wesley Publishing Company, Inc. 1992.

25. S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, Inc. 1984.

26. T. R. Chase (ed.), *Proceedings of the Sixth Annual ASME Database Symposium — Engineering Data Management: Key to Integrated Product Development*, American Society of Mechanical Engineers, New York, 1992.

27. P. P. Chen, "The Entity-Relationship Model: Towards a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, March 1976.

28. H. T. Chou and W. Kim, "Versions and Change Notification in an Object-Oriented Database System," *Proceedings of the 5th International Conference on Data Engineering*, pp. 275-281, Los Angeles, CA, 1989.

29. S. N. Clark, *Transformr: A Prototype STEP Exchange File Migration Tool*, NISTIR 4944, National Institute of Standards and Technology, Gaithersburg, Maryland, October 1992.

30. U. S. Congress, Office of Technology Assessment, *Global Standards: Building Blocks for the Future*, TCT-512, Government Printing Office, Washington DC, March 1992.

31. C. Dabrowski, E. Fong, D. Yang, *Object Database Management Systems: Concepts and Features*, National Institute of Standards and Technology Special Publication 500-179, April 1990.

32. C. J. Date, *An Introduction to Database Systems: Volume 1*, Fifth Edition, Addison-Wesley Publishing Company, Inc., 1990.

33. C. J. Date, *An Introduction to Database Systems: Volume 2*, Addison-Wesley Publishing Company, Inc., 1985.

34. B. Devlin, P. Murphy, "An Architecture for a Business and Information System," *IBM Systems Journal*, vol. 27 no. 1 pp. 60-80, 1988.

35. K. R. Dittrich, and R. A. Lorie, "Version Support for Engineering Database Systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 4, pp. 429-437, April 1988.

36. P. Dwyer, J. Larson, "Some Experiences with a Distributed Database Testbed System," *Proceeding IEEE* p. 633-648, vol. 75 no. 5, May 1987.

37. R. Elmasri, S. Navathe, *Fundamentals of Database Systems*. The Benjamin Cummings Publishing Company, Inc., 1989.

38. J. L. Encarnação, R. Lindner, E. G. Schlechtendahl, *Computer Aided Design: Fundamentals and System Architectures*, Second, Revised and Extended Edition, Springer-Verlag, Heidelberg, Germany, 1990.

39. J. Fedorowicz, "Database Performance Evaluation in an Indexed File Environment," *ACM Transactions on Database Systems*, vol. 12, no. 1. March, 1987.

40. E. Fong, B. K. Rosen, *Guide to Distributed Database Management*, National Institute of Standards and Technology Special Publication 500-154, National Institute of Standards and Technology, Gaithersburg, Maryland, April 1988.

41. E. Fong, C. L. Sheppard, K. A. Harvill, *Guide to Design, Implementation and Management of Distributed Databases*, National Institute of Standards and Technology Special Publication 500-185, National Institute of Standards and Technology, Gaithersburg, Maryland, Feb. 1991.

42. A. Gupta (ed.), *Integration of Information Systems: Bridging Heterogeneous Databases*, IEEE Press, NY, 1989.

43. S. Heiler, "The Integration of Heterogeneous Computing Environments," in chapter 6 of Fong, E. and Goldfine, A. (eds.), *Information Management Directions: the Integration Challenge*, National Institute of Standards and Technology Special Publication 500-167, September 1989.

44. D. Heimbinger, D. McLeod, "A Federated Architecture for Information Management," ACM Transactions on Office Information Systems, vol. 3 no. 3, July 1985.

45. R. Hull, and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, vol. 19 no. 3, 1987.

46. (IGES5.1) *The Initial Graphics Exchange Specification (IGES)*, Version 5.1, IGES/PDES Organization, National Computer Graphics Association, Fairfax, VA, September 1991.

47. (ISO/IEC 9075, 1992) International Organization for Standardization *Database Language SQL*, ISO/IEC 9075:1992, American National Standards Institute, ANSI X3.135-1992, New York, NY 10036, November 1992.

48. (ISO/IEC 9579, 1991) ISO/IEC 9579. International Organization for Standardization/Joint Technical Committee 1, *Open Systems Interconnection - Remote Database Access (RDA), Part 1: Generic Model and Part 2: SQL Specialization*, U.S. public review text, document ANSI BSR X3.217-199x, Global Engineering Documents, Irvine, CA 92714, November 1991.

49. (ISO1, 1992) International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration— Product Data Representation and Exchange — Overview and Fundamental Principles*, Draft International Standard, ISO TC184/SC4, 1992.

50. (ISO11, 1992) International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Description Methods: The EXPRESS Language Reference Manual*, Draft International Standard, ISO TC184/SC4, 1992.

51. (ISO22, 1992) International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Standard Data Access Interface Specification*, Working Draft, ISO TC184/SC4, 1992.

52. (ISO-TC97, 1987) International Organization for Standardization, Technical Committee 97: Information Processing Systems, *Technical Report 9007: Information processing systems — Concepts and terminology for the conceptual schema and the information base*, 1987.

53. R. H. F. Jackson, C. T. Leondes, editors, "Three Pillars of Manufacturing", *Control and Dynamic Systems, Volume 45: Manufacturing and Automation Systems: Techniques and Technologies*, Academic Press, Inc. 1992.

54. D. K. Jefferson, "The Development and Application of Database Design Tools and Methodology," *Proceedings of the Very Large Database Conference*, Montreal, October 1- 3, 1980.

55. R. H. Katz, *Information Management in Engineering Design*, Springer-Verlag, 1985.

56. R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases," *ACM Computing Surveys*, vol. 22, no. 4., pp. 375-408, December 1990.

57. W. Kim, F. H. Lochovsky (eds.), *Object-Oriented Concepts, Databases, and Applications*, ACM Press, NY 1989.

58. W. Kim, et al, "Architecture of the Orion Next-Generation Database System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, 1990.R.

59. R. Kohout, S. N. Clark, *Considerations for the Transformation of STEP Physical Files*, NISTIR 4793, National Institute of Standards and Technology, Gaithersburg, Maryland, March 1992.

60. H. Korth, W. Kim, F. Bancilhon, "On Long-Duration CAD Transactions," *Information Science*, 1988. Reprinted in (Zdonik, 1990).

61. V. Krishnamurthy, Y. Su, H. Lam, M. Mitchell, E. Barkmeyer, "A Distributed Database Architecture for an Integrated Manufacturing Facility," *Second Symposium on Knowledge-Base Integrated Information Systems Engineering*, May 1987.

62. H. T. Kung, J. T. Robinson, "On Optimistic Methods of Concurrency Control," *ACM Transactions of Database Systems*, vol. 6, no. 2, June, 1981. pp. 213-226.

63. G. S. Landis "Design Evolution and History in an Object-Oriented CAD/CAM Database," *Proceedings of the 31st COMPCON Conference*, San Francisco, CA, 1986, pp. 297-305.

64. W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, vol. 22, no. 3, September 1990.

65. M. Loomis, "Data Modeling - the IDEF1X Technique", *IEEE Conference on Computers and Communications*, Phoenix, Arizona, March, 1986, pp. 146-151.

66. S. T. March, "Techniques for Structuring Database Records," *ACM Computing Surveys*, vol. 15, no. 1., March, 1983.

67. S. T. March, J. V. Carlis, "On the Interdependencies Between Record Structure and Access Path Design," *Journal of Management Information Systems*, vol. 4, no. 2. Fall, 1987.

68. M. Mitchell, *A Proposed Testing Methodology for STEP Application Protocol Validation*, NISTIR 4684, National Institute of Standards and Technology, Gaithersburg, Maryland, September 1991.

69. J. Mylopoulos, M. L. Brodie, *Readings in Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1989.

70. K. Narayanaswamy, K. V. Bapa Rao, "An Incremental Mechanism for Schema Evolution in Engineering Domains, *Proceedings of the Fourth International Conference on Data Engineering*, pp. 294-301, IEEE Computing Society Press, 1988.

71. G. Nijsen, T. Halpin, *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice Hall, Englewood Cliffs, NJ, 1989.

72. B. Nixon, J. Mylopoulos, "Integration Issues in Implementing Semantic Data Models," in (Bancilhon, 1990).

73. S. Osborn, "The Role of Polymorphism in Schema Evolution in an Object-Oriented Database", *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 3, September 1989.

74. J. Peckham, F. Maryanski, "Semantic Data Models," *ACM Computing Surveys*, vol. 20, no. 3, September 1988, pp. 153-189.

75. G. Powell, R. Bhateja, "Data Base Design for Computer-Integrated Structural Engineering," *Engineering with Computers*, vol. 4, no. 3, pp. 135-144, 1988.

76. Ranft, M. A., Rehm, S., and Dittrich, K. R., "How to Share Work on Shared Objects in Design Databases," *Sixth International Conference on Data Engineering*, IEEE, LA, CA, Feb 1990, p 575-583.

77. M. T. Rose, *The Open Book: A Practical Perspective on OSI.*, Prentice Hall, Englewood Cliffs, NJ, 1989.

78. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

79. J. R. Rumble, V. E. Hemple, *Database Management in Science and Technology*, North-Holland, Amsterdam, 1984.

80. A. P. Sheth, J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, vol. 22, no. 3, September 1990.

81. J. Smith, D. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, vol. 2, no. 2, June 1977, pp. 105-133.

82. D. Spooner, D. Sanderson, G. Charalambous, "A Data Translation Tool for Engineering Systems," *Second International Conference on Data and Knowledge Systems*, pp. 96-194, IEEE Computing Society Press, 1989.

83. J. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publishing, Reading, Massachusetts, 1984.

84. J. Sowa, *Principals of Semantic Networks: Explorations in the Representation of Knowledge*, 1991

85. S.M. Staley and D. C. Anderson, "Functional Specification for CAD Database," *Computer-Aided Design*, vol.18, no. 3, pp. 132-138, 1986.

86. W. Stallings, *Handbook of Computer-Communications Standards Volume 1: The Open System (OSI) Model and OSI-Related Standards*, Macmillan, New York, NY, 1990.

87. S.Y. Su, "Modeling Integrated Manufacturing Data Using SAM*," *IEEE Computer*, vol. 19, no. 1, pp. 34-49, January 1986.

88. T. Teorey, J. Fry, *Design of Database Structures*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

89. D. Tsichritzis, A. Klug (eds.), *The ANSI/X3/SPARC DBMS Framework. Report of Study Group on Data Base Management Systems*, AFIPS Press, Montvale NJ, 1977.

90. J. Ullman, *Principles of Database and Knowledge-base Systems*, vol. 1, Computer Science Press, Rockville, Maryland, 1988.

91. G. Wiederhold, *File Organization for Database Design*, McGraw-Hill, Inc., New York, 1987.

92. C. Wood, "Choosing an Engineering Object Data Management System" in (Chase, 1992).

93. S. Zdonik, "Object-Oriented Type Evolution," in (Bancilhon, 1990).

94. S. Zdonik, D. Maier (eds.) *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Publishers, Inc., Palo Alto CA, 1990.

95. R. Zicari, "A Framework for Schema Updates in an Object-Oriented Database System," in (Bancilhon, 1992).

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4987

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
DECEMBER 1992

4. TITLE AND SUBTITLE

Database Management Systems in Engineering

5. AUTHOR(S)

Katherine C. Morris, Mary J. Mitchell, Christopher Dabrowski, Elizabeth Fong

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Defense Advanced Research Projects Agency (DARPA)
3701 North Fairfax Drive, Room 739
Arlington, VA 22203-1714

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

Until recently, the applicability of database technology to engineering systems has been limited. Early database systems addressed large-scale data processing needs of easily automateble applications. These applications were characterized by very uniform data and well understood processing methods. Engineering applications, on theother hand, are characterized by highly complex data with very variable structure. The need to represent engineering data has driven advances in database technology.

Engineering domains also impose unique, new requirements on other aspects of database technology. In particular, to support the evolutionary nature of the engineering environment, recent developments in database technology have focused on to the temporal dimensions of data management. In addition, the present trend in manufacturing towards concurrent engineering raises new considerations for the cooperative use of data in a distributed engineering environment. All of these factors are reflected in the new generation of database systems and described in the article.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

engineering database, object-oriented database, engineering requirements for data management, software engineering

13. AVAILABILITY

X	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
X	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
49

15. PRICE
A03

ELECTRONIC FORM