Object-oriented database systems in manufacturing: selection and applications

Qingyu Zhang
The University of Toledo, Toledo, Ohio, USA

Keywords
Object-oriented computing, Database management, Manufacturing

Abstract
As manufacturing systems change from island of automation to enterprise-wise integration, object-oriented database and database management systems have many superior features to meet the new requirements. Based on the comparison with relational databases, this paper discusses the selections and characteristics of the object-oriented database and database management systems (OODBMS) in manufacturing and summarizes the current studies and applications. It helps managers to choose appropriate OODBMS products based on the degree of complexity of their firm's entity and data items. It provides a direction for future research.

Introduction

The trend in manufacturing systems is to change from island of automation to enterprise-wise integration, from physical processing workers to information processing workers, and from management of people/activity to management of information about people/activities. Many companies are going in for computer integrated manufacturing (CIM) to improve productivity and competitive advantage and to meet survival needs of world class manufacturing enterprises in the 1990s and beyond. CIM incorporates a wide range of information technologies such as EDP, MIS, DDS, ES, CAD, CAM, CAPP, FMS, etc. Although progress has been made over the years, there are few methodologies to assist with the system planning and development of these complex systems (Flatau, 1988; Ciampa, 1988; Nalder and Robinson, 1987; Banerjee, 1986). This lack of methodologies has led to many problems (Yeomans *et al.*, 1986; Ciampa, 1988; Gunn, 1987) such as a lack of integration methods, lack of standards, lack of understanding of CIM variables, island of automation paradigm, and a confusing and narrow definition of CIM.

Many authors (Leavitt, 1965; Galbrait, 1977; Yadav, 1983; Grant *et al.*, 1992) have pointed out the importance of synergy among task, technology, people, communication, and structure in CIM implementation. So object-oriented modeling of information systems has been suggested (Bailin, 1989; Bulman, 1989; Coad and Yourdon, 1990) such as object-oriented product modeling and design (Usher, 1993), object-oriented shop floor control and distributed scheduling (Kim *et al.*, 1996), object-oriented process planning (Usher, 1996a, b; Gu and Zhang, 1994), object-oriented product data exchange integration (An and Leep, 1995), and object-oriented bill of materials (Chung and Fischer, 1994). At the same time, making CIM work effectively calls for a high level of interoperability, integration, and data sharing, therefore bringing databases and database management systems (DBMS) to the forefront.

Today, the relational database model and relational database management system (RDBMS) have been the *de facto* industry standard for organizing and managing data in most CIM environments. Some authors (Vasilash, 1990; Lockemann *et al.*, 1991) hold that the relational data model is very powerful and serves as a bridge to connect the islands of automation. It is mature and reliable, and it has proven to be a flexible platform for evolution toward new applications, furthermore, the relational model is based on the formal mathematical model while the object-oriented model is not. Although the object-oriented model incorporates many useful concepts such as inheritance, abstraction, behavior encapsulation, reuse, and message passing, it can not be as robust and rigorous as one that is grounded in theory. In contrast, in recent years, many other scholars (Schatz, 1988; Tonshoff and Dittmer, 1990; Cattell, 1991; Rasmus, 1991; Gomsi and DeSanti, 1992) have stated the limitations and inadequacies for using RDBMS for CIM and lean toward object-oriented database management systems (OODBMS). Especially an object-oriented approach to modeling integrated systems is emphasized by many authors (Motavalli, 1997; Do, 1997; Watterson, 1998)

Database requirements of CIM applications

CIM encompasses the use of computers and the integration of all activities necessary to transform purchased materials into products, to deliver products to customers, and to support the performance of production. A CIM system is created by

Industrial Management & Data Systems
101/3 [2001] 97–105

[ISSN 0263-5577]

superimposing an information system and decision support system over the manufacturing infrastructure including facilities, products, and material flow. The hub of the CIM information system is a DBMS and database, which integrates all the information on the essential aspects of a company's manufacturing activities so that it is shareable among the subsystems. Tonshoff and Dittmer (1990) state that today substantial heterogeneity exists in data processing in production systems where all processing systems maintain their own data. Such subsystem-oriented data management creates some apparent problems: data redundancy, data inconsistency, expensive integration of systems, and minimal manufacturing transparency. This is further reinforced by Doll and Vonderembse (1987), who state that an effective implementation of CIM would require a shared database, a high level of data management capacity and communication network linking engineering (CAD, CAE). manufacturing (CAM, CAPP, GT, AGVS, JIT, MRPII), and business information systems (product entry/exit, product design and process selection, quality management). The heart of CIM information resource management is the concept of data management.

The next generation manufacturing applications require new data management approaches and the following requirements have to be addressed (Joseph *et al.*, 1991).

Rich data modeling

Object-oriented concepts can be used to model application data and relationships in a natural manner, many applications need to deal with persistent data (data that live after the processes that created them terminate) and share the data among multiple users.

Query access to objects

It must be possible for large application to navigate through each object by a query and retrieve definition of all functions, thus retrieval based on predicates needs to be supported.

Sharing of objects among application systems

It is imperative that the database can handle data generated by different programming languages. Since adaptability to change is crucial to database, it must be possible to use old data (objects) in new environments (languages).

Seamless

It is imperative that the integration of a database with the rest of the programming environment is achieved in a nonobtrusive manner, that is, the database supports as rich a data model as those found in programming languages.

Transactions appropriate for cooperative design work

Long-duration transactions (such as CAD) need to lock data for the entire duration of transactions in cooperative design work.

Support for the evolution of object instances and classes

Application programs are not static entities, they undergo change. So it is necessary to support for life cycle schema evolution.

Distributed, platform independent, object storage

The next generation applications deal with large gigabytes of information of varying size, which need to distribute storage. Such environments may be heterogeneous, and data storage must be platform independent and migrate gracefully from one generation of hardware and operating system platforms to the next.

Other requirements

Some baseline requirements like adequate performance, reliability, robustness, and easy-to-use interfaces will remain applicable.

Review of relational DBMS

The core of a typical database includes the application data itself; the data dictionary, which contains the description of the database; and overhead data, consisting of linked lists, indexes, views, and similar data, which are required to store the relationships among the data. The role of the DBMS is to manage all aspects of the database, and to serve as the sole interface between the database and all other applications. Database technology has undergone three main paradigm shifts, starting with first-generation DBMS in the 1970s (hierarchical and network), through second generation DBMS in the 1980s (relational), to the third generation DBMS in the 1990s (object-oriented/hybrid).

The relational model, first proposed by Dr Codd (1970), has achieved the greatest acceptance among all the approaches to data modeling and DBMS design, and is *de facto* the industry standard. The prime reason for the success of the relational approach is its firm foundation on relational mathematics, end-user friendliness, and quick response to SQL queries. Relational DBMS present data

Qingyu Zhang
Object-oriented database systems in manufacturing: selection and applications
Industrial Management & Data Systems
101/3 [2001] 97–105

to users in the form of simple two-dimensional flat files or tables, called relations. If two tables share a common field (attribute), called primary/foreign key, the relational model can relate any data stored in one table to any data stored in the other. This cut-and-paste capability of the relational model is the source of its enormous flexibility. The relational model uses eight operators (select, project, join, union, intersect, difference, product, and divide) from relational algebra which have come to be implemented in a standardized form called structure query language (SQL). Integrity rules (entity integrity, referential integrity, and business integrity) not only provide integrity of data structure but also realize the complex business rules. Normalization concepts in the relational model eliminate various types of modification anomalies that may arise due to insertion, deletion, and updating of information in a database.

The analysis and design of a relational database involves three levels of abstraction, namely, conceptual, logical, and physical. The conceptual design entails modeling real world entities using a high level semantic model, e.g. entity-relationship (ER) diagrams, proposed by Peter Chen in MIT. The ER diagram employs the concept of entities, their attributes, and their relationships for modeling purpose. Logical design represents real world entities as data values in a symbolic model, e.g. the relational model, using constructs such as primary and foreign keys, domains, and referential integrity through a normalization process. Such a normalization process depends on the concept and diagram of functional dependence, proposed by E.J. Date, a famous implementer. Physical design represents symbolic values and access paths as bits and bytes on tracks and sectors of disk drives, e.g. cluster table, hash index, and denormalization.

The relational model has been widely applied in business organizations such as airlines, insurance companies, bankers, manufacturers, government agencies, and so on. The data they use are alphanumeric and cast in a record-oriented format. But real world objects for many applications such as text and image processing, office automation system, GIS, CAD, robotics, and CASE are difficult to capture in a flat, record-oriented model. In fact, only 10 percent of an organization's data may be characters and numbers (Ricciuti, 1992), and the rest are in a form that does not easily translate into rows and columns, so very few of the data can be identified by the simple relations of tuples of defining characteristics in relational database schema.

RDBMS has some limitations in supporting the design of CIM information systems. The relational model fails to address the complex semantics associated with 3D geometrical graphics and image, complex nested entities, and the like in CAD/CAE, and fails to manage arrays and groups of data values associated with the production control process (Weber and Moodie, 1989), and fails to represent bill of materials (BOM) efficiently in MRPII, and fails to express the complex unstructured data type such as photos, maps, sound.

In total, the relational database provides:

- transaction management for correct, efficient, and concurrent access by multiple users;
- access control for limiting data access to authorized users only;
- long-term reliable storage of data and recovery from media and system failure; and
- support for one or more query language for data definition and data manipulation.

But they are inadequate for the following reasons:

- lack of expressive data modeling power;
- the so-called "impedance mismatch" between programming languages and database systems;
- inadequate interactive performance to support next application;
- lack of appropriate mechanisms for supporting long transactions; and
- lack of appropriate mechanisms for supporting schema evolution and version management.

Object-oriented and hybrid models

Object-orientation provides a natural way to map real world objects and their relationships directly to computer presentations. Fundamental concepts and characteristics are as follows (Table I).

OODB systems (OODBs include OODB and OODBMS) represent the confluence of ideas from object-oriented programming languages such as Lisp, C++, and SmallTalk, which provide rich data abstraction capabilities including the powerful modeling capabilities based on the ability to define abstract data types and construct type hierarchies that permit property inheritance, and database management, which provide long-term reliable data storage, multi-user access, concurrency control, query, recovery, and security capabilities.

Table I
OODB's concepts and features

Terms	Meaning
Objects	Entities that are used to represent abstract or concrete real-world things
Attributes	The set of values of the local state of an object (called instance variables, properties, data members, or slots)
Message and type	The external requests; the collection of all messages
Method and behavior	Change mode in response to message; the collection of all methods of an object
Encapsulation	The local state and method not visible to the users of the objects
Object identity	A unique identifier, which is a logical pointer
Class	The means of grouping objects with the same attributes and behavior
Class composition hierarchy	The domain of an attribute may be a class that, in turn, may have attributes with domains as classes (nested structure)
Inheritance	The new class derives from the old class and inherits all attributes and methods of the original class
Dynamic binding and polymorphism	The ability to bind message to different methods depending on type
Seamlessness	Integration of the database with the rest of the programming environment in a nonobtrusive manner
Secondary storage management	Efficient data access by supporting clustering, index, buffering, and query optimizations
Persistence	Existence of objects beyond the life time of the processes that create them
Transactions and concurrency control recovery	Concurrent access to data by means of atomicity, controlled sharing via locks, serializability
Query facility	Efficient high level declarative access to objects in addition to navigational programmatic access
Design transactions	Long running and nested transactions
Change management	Database support for managing the evolutionary life cycle of objects and classes

These models have emerged in an attempt to store, search and manipulate data about objects which have complex inner data structures. Object-oriented database management systems (OODBMS) are systems which are designed from scratch, whereas hybrid DBMS are some combination of RDBMS and OODBMS. Traditional DBMS including RDBMS store just data, without the procedure required to manipulate the data. This provided the long sought-after independence between application programs and their operational data.

In contrast, OODBMS store objects. An object contains data about an entity, and also the methods that process those data. An object may be anything to which a concept applies, e.g. a number (including lists and arrays), a document or graphics (vector or bit map), a sound or an image, or even procedure and self-adaptation (e.g. a join command under certain conditions or a self-referential conditional change within the database itself). Whereas in conventional DBMS, any kind of procedure can process data, in OODBMS the data can only be accessed through methods stored with them as part of a class. Objects can be composed of other objects, which in turn can be composed of other objects, and so on. This enables the capture of highly complex data structures. A major difference between the two approaches is that RDBMS databases are passive, meaning that they contain only data. OODBMS databases are active, because an attempt to read or update the data would trigger certain action automatically. This is known as instilling intelligence in the database just like semantics-capturing schema being used in AI.

Unlike the relational database model, the object-oriented database is not based on a formal database model, but a collection of concepts such as data and behavior encapsulation, inheritance, etc., which allow the OODBMS to capture more of the semantics of the real world. The flexibility in the use of subclasses and superclasses, and the ability to handle multi-valued attributes, further increase the semantic richness of OODBMS. Encapsulation combines certain processes that are themselves characteristics of the objects which incorporate the methods associated with the objects, and thus the programmer no longer has to write code to manipulate objects. Inheritance provides the capacity to build up new objects from existing objects without having to redefine attributes and methods. Further, OODBMS has the virtue of "seamless persistence" (a seamless database coexists with the rest of

the programming environment with a minimum of friction), and alleviates impedance mismatch, that is, spending too much effort getting the data structures of the programming language to make sense with those of the data manipulation languages (DML) (Braham, 1991).

Although OODB technology has great potential, OODBMS is not free from limitations, for example, the lack of a strong underlying theory and the lack of a standardized easy-to-use query language like RDBMS's SQL, and the variety of commercial products and research prototypes indicates that consensus and standards, feedback and improvement, benchmarks are just beginning.

Given the relative immaturity of OODBMS and the enormous investment in current RDBMS, there is a great deal of activity in terms of developing hybrid models, which incorporate both models. There are two general methods for achieving this. The first approach simply adds an additional layer on top of the existing RDBMS, to provide object-oriented capabilities (e.g. Oracle 8i). This preserves all the work and experience gained in developing highly successful RDBMS products. The second approach involves extending the relational database to accommodate object-oriented data. Thus, special extensions to existing products would have to be built to accommodate the complex data structures.

The OpenODB product (Hewlett-Packard), which most closely resembles the hybrid model conceptually, combines the old with the new by building an object database manager on top of all base/SQL, a more traditional DBMS. It is expected to allow users to query either relational or object data using a query language called object SQL (OSQL) developed by Hewlett-Packard.

POSTGERS is another existing hybrid DBMS with its own query language called POSTQUEL (Heintz, 1991). Apart from OpenODB and POSTGERS, there are few other commercial offerings for hybrid DBMS since the knowledge in this area is yet to mature fully. Other limitations are relatively high system cost and low performance due to the interaction of application programs written in non-object-oriented programming languages such as C or Cobol, with the object-oriented front end, which can really slow down the system (Gomsi and DeSanti, 1992).

DBMS selection

Although OODBs will support next generation applications such as CAD, CASE, CAM, multimedia and hypermedia information system, and artificial intelligence expert systems in manufacturing, OODBs are not expected to replace conventional databases in commercial applications for several years. Relational databases, their extensions, and OODBs will coexist to support different activities of an enterprise (Joseph *et al.*, 1991; Parker, 1997).

In total, we can select CIM DBMS according to the company's degree of complexity in two dimensions: entity and data item (Bordoloi *et al.*, 1994) as Table II suggests.

Survey of OODB products and research prototypes

The CACTIS system represents the first effort to develop an OODBMS from scratch (Hudson and King, 1988). On the other hand, work on extending object-oriented programming languages has produced GemStone, which is relatively the most complete of the commercially available OODBMS (Maier and Stein, 1986). It is based on the Smalltalk language, which provides capabilities such as abstract data tying, creation of complex objects, and object sharing. Ontos is also a commercial object-oriented database system by Ontologic. The basic product classifications (Table III) and comparison of OODBs (Table IV) are as follows:

An example

In the following section, we give an example about how GemStone systems can be used compared with ORACLE, a typical relational-object hybrid database management system. GemStone is a commercial OODB system from Servio Logic Development Corporation. It offers one uniform language to its programmers: OPAL, which is a modified version of SmallTalk-80. OPAL is to GemStone what PL/SQL is to Oracle. It is a complete language with assignment, conditional, and interaction constructs.

GemStone is designed to increase the database modeling power and reduce the development time of applications with complex information needs. Such applications include office information systems, CAD, and documentation of complex mechanical systems. The three principal concepts of the GemStone object model are object, message, and class. Theses correspond to record, procedure call, and record type in relational database systems.

Table II
Selection dimensions and criterion of DBMS

Dimensions	Criterion	Simple	Complex
Entity	Entity identification	Specific (e.g. part)	Abstract (e.g. shipment)
	Entity evolution	Static	Dynamic
	Entity complexity	Independent/non-nested	Nested/objects within objects
	Entity reuse	Low	High
Data item	Data type	Fixed number of alpha	Unstructured (user defined, voice, 3D graphical)
	Data attribute content	Numeric data types and operators	Arrays and group attributes
		Single-valued	
DBMS		Relational	Hybrid Object-oriented

Table III
Overview of product classifications

Product category	Object programming language bindings	Database engine	Database storage format
Persistent language	Yes	None	Same as language
RDBMS	None	Yes	Tuples
Obj.-rel. hybrid	Yes	Not all	Tuples and objects
OODBMS	Yes	Yes	Objects

Table IV
The characteristics of all kinds of OODBs

Database	Gemstone	IDB	Matisse	Objectivity	ObjectStore	Ontos	OpenODB	UNI	Poet	Versant
Architecture	OO	OO	Semantic	OO	OO	OO	Ex/rel	Ex/rel	OO	OO
Server based	y	n	n	y	n	n	y	y	n	y
Client based	y	y	y	y	y	y	n	y	y	y
Navigational	y	y	y	y	y	y	n	n	y	y
Long transaction	y	y	Possible	y	y	y	n	Scheme	n	y
Checkin/checkout	y	n	n	y	y	Scheme	n	n	Scheme	y
Versioning	y	y	y	y	y	n	n	Scheme	y	y
Lock level	Class	Object	Object	Object	Page	Page	Block	Record	Object	Object
Buffering	y	y	y	y	y	y	y	y	y	y
Blocking	y	y	y	y	y	y	y	y	y	y
Indexing	y	y	y	y	y	y	y	y	y	y
Compression	y	n	n	y	n	n	n	n	n	n
Clustering	y	y	y	y	y	y	y	y	n	y

Note: ex/rel = extended relational

In the OPAL, classes consist of data structure definition and a collection of operations called methods. The form of a method definition is:

```
Method <class name>
        <message format>
        <body of method>
%
```

Consider a common case: In order to define and manipulate employee records in one firm, with the software of Oracle, we define table emp with three attributes: emp [empNo, name, salary]. Then we use PL/SQL to write direct SQL, procedures, functions, triggers, and packages to manipulate it. But in OPAL, we present it in an object-oriented way as follows.

```
Object
  Subclass: "EmpType"
  InstVarNames: #["name", "empNo",
                  "salary"]
  Constraints: #[
  #[ #name, String],
  #[ #empNo, Integer]
  #[ #salary, Integer]
  ].
Method: EmpType
            getSalary
                  ^salary
%
```

```
Method: EmpType
                SetSalary: n
                        Salary := n
%
```

For the queries, "find all the employees whose salary is greater than 20,000" and assume that all employee objects are stored in a set called Employees.

```
OPAL:    highPayEmps := Employees
         select: [:e | (e getSalary) > 20,000]
PL/SQL: SELECT * FROM emp WHERE
         salary > 20,000
```

The difference is that the OPAL accesses encapsulated objects through methods with message passing; Oracle directly operates the table with SQL. If we want to combine the table record with programming, we have to use cursor and interactive loop in Oracle.

GemStone has the following main advantages as OODBMS:

- Sharing of objects: GemStone provides a distinctive list of dictionaries called a symbolist. This list can be shared and acts like a file directory. Thus, it is conducive to a multi-user environment.
- GemStone has security on the systems level and object level.
- Method execution: GemStone allows the designers to choose to copy an object's state to client side or to directly execute a message remotely on the GemStone Server.
- GemStone has uniform language, which prevents impedance mismatch.

Conclusion and implications

This paper provides a survey and summary of OODBMS products and research prototypes. The characteristics and features between the relational DBMS and OODBMS and among all kinds of OODBs are compared and described. OODBMS has superior features to meet the new requirements in manufacturing.

For managers, this paper helps a firm to choose an appropriate product based on the degree of complexity of their firm's entity and data items (Table II). If their firms need to define and operate a lot of complex entity types (e.g. sound, bitmap, and graphics) and complex relations (e.g. whole-and-part relation), they do need to consider adopting object-oriented database systems, at least hybrid systems. Then the firm decides to buy what kind of OODBMS based on the analysis of cost and benefit with regard to the specific product characteristics (Tables III and IV and the firm's current information architecture).

From a strategic perspective, a firm has to renew their technological capability in time, otherwise, they will fall far behind and can never catch up. Currently, it seems that programming languages, database management systems, and systems analysis and design approaches converge to an object-oriented philosophy. Thus, OODB has a great potential to become a mainstream product. Thus, whether a firm adopts OODB is not the issue that involves solving current operational problems, but the one that is related to the strategic reconfiguration of the organizational technological capability.

For researchers, given the immaturity of the OODB field in manufacturing, the following issues are likely to receive more attention in the next years:

- *Access control* – Because OODBs store active objects, it is possible for objects to perform their own access control. So an interesting issue is how access control will interact with querying, since queries are generally performed under control of the database rather than the application.
- *Remote database access* – OODBs can access heterogeneous databases and impress the users with a single OODB being accessed, so how to make the "foreign" databases compliant to ensure atomicity.
- *Interlanguage sharing* – Accommodating persistence in single mainstream programming language is supported by a number of commercial OODBs and research prototypes such as SmallTalk in Servio Corporation's GemStone (Purdy *et al.*, 1987), Common Lisp in TI Zeitgeist (Ford *et al.*, 1988), and C/C++ in Ontologic Ontos. But achieving seamless persistence with respect to multiple host languages that share persistent objects is still a research issue.
- *OODB standardization and benchmarks* – To meet interoperability and interchange ability, consensus on OODB functionality is necessary. In January 1989, the OODB task group (OODBTG) recommended ASC/X3 as the existing standard based on glossary, reference model, operational model, interface, and data exchange. In April 1989, the object management group (OMG) issued a common object request broker architecture (CORBA) standard (all communication goes through the object request broker) to the industry for a common enterprise-wide OODB system including CAD, CASE framework. So there is growing interest in building benchmarks for standard conformance testing to certify OODB interoperability compliance.

- *Enterprise-wide database systems* – How to effectively integrate, express, and manage the whole engineering, manufacturing, and business process data in an OODB (or distributed) with known OO technology to achieve efficiency and effectiveness is the main concern in the next few years.

References

An, D. and Leep, H.R. (1995), "A product data exchange integration structure using PDES/STEP for automated manufacturing applications", *Computers and Industrial Engineering*, Vol. 29 No. 4, pp. 711-15.

Bailin, S.C. (1989), "An object-oriented requirements specification method", *Communications of ACM*, Vol. 32 No. 5.

Banerjee, S.K. (1986), "Information systems design for CIM – a methodology", in McGeough, J.A. (Ed.), *International Conference of Computer-Aided Production Engineering*, Edinburgh.

Bordoloi, B., Agarwal, A. and Sircar, S. (1994), "Relational or object-oriented or hybrid?", *International Journal of Operations and Production Management*, Vol. 14 No. 9, pp. 32-44.

Braham, J. (1991), "Engineering your way to the top", *Machine Design*, Vol. 63 No. 17, pp. 65-8.

Bulman, D.M. (1989), "An object-based development model", *Computer Language*, August.

Cattell, R.G.G. (1991), "What are next-generation database systems?", *Communications of the ACM*, Vol. 34 No. 10, pp. 31-3.

Chung, Y. and Fischer, G.W. (1994), "A conceptual structure and issues for an object-oriented bill of materials data models", *Computers and Industrial Engineering*, Vol. 26 No. 2, pp. 1-339.

Ciampa, D. (1988), *Manufacturing's New Mandate*, Wiley, New York, NY.

Coad, P. and Yourdon, E. (1990), *Object-Oriented Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

Codd, E.F. (1970), "A relational model of data for large relational databases", *Communications of the ACM*, Vol. 13, June, pp. 377-87.

Do, N.C. (1997), "Constraint maintenance in engineering design system: an active object-oriented approach", *Computers and Industrial Engineering*, Vol. 33 No. 3/4, pp. 643-7.

Doll, W. and Vonderembse, M. (1987), "Forging a partnership to achieve competitive advantage: the CIM challenge", *MIS Quarterly*. June, pp. 205-20.

Flatau, U. (1988), "Designing an information system for integrated manufacturing systems", in Compton, D.W. (Ed.), *Design and Analysis of Integrated Manufacturing Systems*, National Academy Press, Washington, DC.

Ford, S. *et al.* (1988), "Database support for object-oriented databases", *Proceedings of the International Workshop on Object-Oriented Database Systems*, pp. 23-42.

Galbraith, J.R. (1977), *Organizational Design*, Addison-Wesley, Reading, MA.

Gomsi, J. and DeSanti, M. (1992), "A technical comparison of relational and object-oriented data bases for manufacturing applications", *Data Resource Management*, Winter, pp. 40-7.

Grant, D., Ngwenyama, O. and Klien, K. (1992), "Modeling for CIM information systems architecture definition", *Computers in Industry*, Vol. 18 No. 2.

Gu, P. and Zhang, Y. (1994), "OOPPS: an object-oriented process planning system", *Computers and Industrial Engineering*, Vol. 26 No. 4, pp. 709-31.

Gunn, T. (1987), *Manufacturing for Competitive Advantage*, Ballinger, Boston, MA.

Heintz, T. (1991), "Object-oriented database and their impact on future business database applications", *Information and Management*, Vol. 20, pp. 95-103.

Hudson, S.E. and King, R. (1988), "The CACTIS project: database support for software environments", *IEEE Transactions on Software Engineering*, Vol. 14 No. 6, pp. 709-19.

Joseph, J.V., Thatte, S.M., Thompson, C.W. and Wells, D.L. (1991), "Object-oriented databases: design and implementation", *Proceedings of the IEEE*, Vol. 79 No. 1, pp. 41-54.

Kim, K.H., Bae, J.W., Song, J.Y. and Lee, H.Y. (1996), "A distributed scheduling and shop floor control method", *Computers and Industrial Engineering*, Vol. 31 No. 3/4, pp. 583-6.

Leavitt, H. (1965), "Applied organizational change in industry: structural, technological and humanistic approaches", in March, J. (Ed.), *Handbook of Organizations*, Rand McNally, Chicago, IL.

Lockemann, C.P., Kemper, A. and Moerkotte, G. (1991), "Future database technology: driving forces and directions", *Future Generation Computer Systems*, Vol. 7, pp. 31-3.

Maier, D. and Stein, J. (1986), "Development of an object oriented DBMS," *ACM Sigplan*, Vol. 21 No. 11, pp. 472-82.

Motavalli, S. (1997), "Feature-based modeling: An object oriented approach", *Computers and Industrial Engineering*, Vol. 33 No. 1/2, pp. 349-52.

Nalder, G. and Robinson, G. (1987), "Planning and designing and implementing advanced manufacturing technology", in Wall, T., Clegg, C.W. and Kemp, N.J. (Eds), *Human Side of Advanced Manufacturing Technology*, Wiley, New York, NY.

Parker, K. (1997), "Are objects client/server?", *Manufacturing Systems, Enterprise Systems for Mid-sized Manufacturers*, supplement January, pp. 16A-24A.

Purdy, A., Schuchardt, B. and Maier, D. (1987), "Integrating an object-server with other worlds", *ACM Transactions on Office Information Systems*, Vol. 5, pp. 27-47.

Rasmus, D. (1991), "Object of your desire: the future of manufacturing systems", *Manufacturing Systems*, July, pp. 42-6.
Ricciuti, M. (1992), "The road to objects: RDBMS vendors face an object future", *Datamation*, November, pp. 41-8.
Schatz, W. (1988), "Making CIM work", *Datamation*, December, pp. 8-11.
Tonshoff, K. and Dittmer, H. (1990), "Object- instead of function-oriented data management for tool management as an example application", *Robotics and Computer Integrated Manufacturing*, Vol. 7, pp. 133-41.
Usher, J.M. (1993), "An object-oriented approach to product modeling for manufacturing systems", *Computers and Industrial Engineering*, Vol. 25 No. 1-4, pp. 557-60.
Usher, J.M. (1996a), "A step-based object-oriented product model for process planning", *Computers and Industrial Engineering*, Vol. 31 No. 1/2, pp. 185-8.
Usher, J.M. (1996b), "A tutorial and review of object-oriented design of manufacturing software systems", *Computers and Industrial Engineering*, Vol. 30 No. 4, pp. 781-98.
Vasilash, S.G. (1990), "Relational databases: perhaps not very interesting – but powerful", *Production*, August, pp. 76-8.
Watterson, K. (1998), "When it comes to choosing a database, the object is value", *Datamation*, Vol. 44 No. 1, pp. 100-07.
Weber, D.M. and Moodie, C.L. (1989), "From database systems to information management systems – requirement for computer integrated manufacturing and assembly", *Databases for Production Management, Proceedings of the Conference on Design, Implementing, and Operations of Databases for Production Management*, No. 10-12, May, pp. 141-65.
Yadav, S. (1983), "Determining an organization's information requirements: a state of the art survey", *Data Base*, Spring, pp. 3-20.
Yeomans, R.W., Coudry, A. and Hagen, P.J.W. (1986), *Design Rules for a CIM System*, North Holland, Amsterdam.

Further reading

Barry, D.K. (1996), *The Object Database Handbook: How to Select, Implement, and Use Object-Oriented Database*, Wiley Computer Pub., New York, NY.
Cheolham, K., Kwangsoo, K. and Injun, C. (1993), "An object-oriented information modeling methodology for manufacturing information systems", *Computers and Industrial Engineering*, Vol. 24 No. 3, pp. 337-53.
Maier, D., Stein, J., Otis, A. and Purdy, A. (1986), "Development of an object-oriented DBMS", *OOPSLA '86 Conference Proceedings*, pp. 472-82.
Ramamurthy, K. and King, W.R. (1992), "Computer integrated manufacturing: an exploratory study of key organizational barriers", *Omega*, Vol. 20 No. 4, pp. 475-91.