INT. J. COMPUTER INTEGRATED MANUFACTURING, VOL. 1, NO. 1, 44–49

The integrated manufacturing data administration system (IMDAS)—an overview

DON LIBES and ED BARKMEYER

Abstract. Automated manufacturing requires sharing of data among control, sensory and administrative processes. Since these processes are invariably distributed over many different computer systems, we claim that a distributed data system is necessary. Unlike most extant data systems, a manufacturing enterprise requires support for: diverse computer systems, data systems and databases; real-time data access; and integration of new systems into a running complex. This paper briefly discusses these issues and describes a prototype implementation of a distributed data system that addresses them.

1. Background—issues in manufacturing data access

Increasing industrial automation results in factory floors containing large numbers of computer systems controlling and monitoring physical processes. *Computer integrated manufacturing* (CIM) refers to the integration of these systems into an automated production complex closely coupled to the engineering and administrative systems that support and drive it. The critical element in such a complex is the ability of all the associated programs and users to share data.

There are several problems in the automated manufacturing environment which are not necessarily encountered in other distributed computing environments.

A *fundamental characteristic is the diversity of computer* systems in a single CIM complex. Rarely does the same kind of computer system perform engineering support, real-time control and administrative applications. Indeed, most industrial facilities contain computer systems from many different manufacturers. Consequently, data sharing is complicated by differing operating systems, hardware architectures, data systems and access methods. To get the same information from two different machines, expect to use two different interfaces, phrase the request in two different 'languages' and get back the information in two different forms. Since data systems range from very limited to very powerful, it is possible for one of two data systems to lack a capability that another has.

Another problem is data organization. Because organization of data has significant performance implications for most database systems, serious effort is expended in getting the optimal organization. In the CIM environment, almost all data is significant to more than one application, but the way in which it is best organized for each application area may be different. What usually results is separate databases with overlapping information units.

For example, Production wants to keep track of component inventories by *part-type* and *location*, while Purchasing wants to organize the components by *supplier*, *shipment* and *order* and Accounting wants to track them by *part-type*, *invoice* and *target product*. Naturally, having been organized for different applications they have different schemas and often use entirely different database management systems. Because some of the key information units are the same, a change in one of the databases often necessitates corresponding changes to the others.

A direct consequence of multiple databases is the problem of representation. The same logical information units may have entirely different encodings from one database to another. For example, one system may identify a product by its production code while another identifies the same product by its commercial catalogue number. And because different machines store and communicate datatypes differently, even the same logical representation may have different physical representations on different machines. For example, one data system may store strings in EBCDIC and another in ASCII.

It is possible for each application to locally take responsibility for integration. This is what is done in most existing manufacturing enterprises. Once the appropriate *data repository* is identified, an interface to that database and system is then encoded in the application.

The emerging Manufacturing Automation Protocols

Authors: Don Libes and Ed Barkmeyer, Integrated Systems Group, National Bureau of Standards, Gaithersburg, MD 20899, USA.

(MAP) (General Motors 1986) standards effort has somewhat ameliorated this problem by providing a standard method of interchanging files between any two computer systems. Unfortunately, the more interesting databases are not simple 'files' and interchangeable files of the wanted information must be extracted from those databases before the standard transmission mechanism can be employed.

This means that application programs must be aware of the location and organization of the data they use. In practice, the location and organization of most data changes and multiplies over time as new applications and equipment are added and existing ones are upgraded. Each change requires reprogramming the local integration in every affected application.

As the automated enterprise grows, this approach becomes intractable. The alternative is to provide applications with an interface to a 'common data system'. The data system becomes responsible for the vagaries of individual data units. Most application software will then be unaffected by the addition or replacement of equipment. The only applications which will be affected are those being added or those wishing to make use of newly available data.

Finally, a problem which is probably unique to the manufacturing floor is *real-time* access to data. With increasingly sophisticated automation, it is common to see several separate systems cooperating in a single physical operation, such as a robot handing off a workpiece to an automated fixture. While not usually viewed as a 'database' problem, this is just as much a 'data sharing' problem as is obtaining the robot control programs from an engineering database. It has different constraints, but many of the same concerns.

So we see that a manufacturing complex comprises a large number of dissimilar computer systems, data systems and databases. The need for sharing data among these systems results in overlapping databases with representational inconsistencies. We have argued that instead of trying to solve the integration problem in each application, we need to build a 'common data system' which solves the integration problem and which all the applications can use. Finally, we assert that such a system must support real-time use. How does one go about constructing such a system?

2. Approaches

The simplest approach to development of a common data system is the centralized system: a single common database on a central computer system with communication paths to client systems. The inherent simplicity of this architecture, particularly with regard to management and maintenance, makes it highly desirable if it can be made practicable. With current computer technology, it is possible to create a central system with sufficient redundancy that a total failure is extremely unlikely. However, the ability of such a system to handle all of the database transactions for the whole manufacturing enterprise is very questionable. In all but the smallest organizations, performance and cost considerations will dictate distribution of function and data.

Another possibility is maintaining distributed overlapping databases with conversion of data between them. This is viable if the number of databases is small and the interactions between them are carefully timed and controlled. The number of conversion programs will grow proportionally to the square of the number of databases. Any two databases must be idle for a certain period of time in order for information to be transferred between them. Moreover, such databases will normally be inconsistent and be brought into synchronization only at the times of transfer between them.

The alternative of modifying programs to access data from multiple databases has been discussed above. It avoids the consistency problems inherent in the database conversion approach, but it is simply impractical in an environment of numerous systems and regular changes. If there are n databases and m programs, every new program will have up to n interfaces to access the necessary data, and every change to the data architecture can require modification of m programs.

By using a common interface between programs and databases, this complexity can be avoided. Each new program has only one interface to all data. A change to the architecture of one database requires modification of only its interface to the common service, not to any of the applications. It is our belief that most enterprises will benefit from a system of distributed databases with common interfaces. This will enable integration of new and existing data systems and relieve programs and programmers of network, access and conversion problems.

The remaining choice is whether to provide the common interface from a single server with interfaces to each of the distributed databases, or to distribute the common interface service over several systems with interfaces to each other. Again, the centrally controlled system is simpler and currently practicable, but its ability to handle all of the transactions, even when it can distribute the actual data manipulations, must be in doubt. On the other hand, the protocol problems that result from trying to do distributed data management by committee has been the subject of much academic discourse and few, if any, sound solutions.

3. IMDAS approach

Our approach to providing a common interface to distributed data is the *Integrated Manufacturing Data Administration System*. IMDAS is characterized by:

(*a*) a common interface to user programs, and
(*b*) a common interface to underlying databases

Application programs communicate with IMDAS using an SQL-like language, referencing data names from a common dictionary. ANSI-standard SQL (ANSI 1986) was designed for interaction with a user at a terminal. With additional features, the IMDAS data manipulation language allows programs to specify files or buffers as the sources and destinations of data.

On the other side, IMDAS has a common interface to underlying *data repositories*, which can be commercial database systems, file systems or home-grown application-specific data systems. This minimizes the work needed to incorporate new databases into the common data system while allowing existing systems to continue operating without change.

The dual common interfaces afford programmers a generalized view of data access. They see data manipulation as operations on information units, not databases, and are not concerned with what system or machine has the data. The result is conceptually simple. The user sees a single common database managed by the IMDAS (Fig. 1).

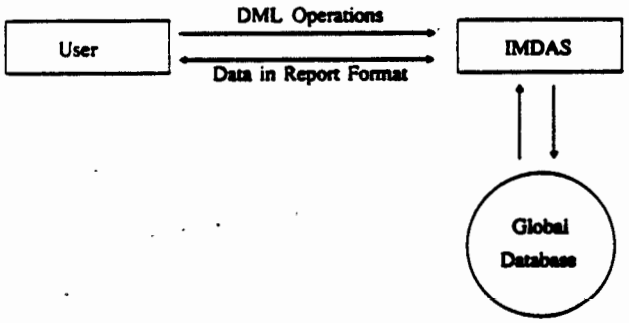

Figure 1. IMDAS concept.

4. IMDAS architecture

The internal architecture of IMDAS is a 4-level hierarchy (Fig. 2). The levels are distinguished primarily by scope of responsibility for data management. The higher the level, the more data is administered.

At the bottom level of the IMDAS are the data repositories, such as commercial DBMS. Also included are other repositories of sharable information, such as file systems, controller memories, and home-grown application-specific data managers.

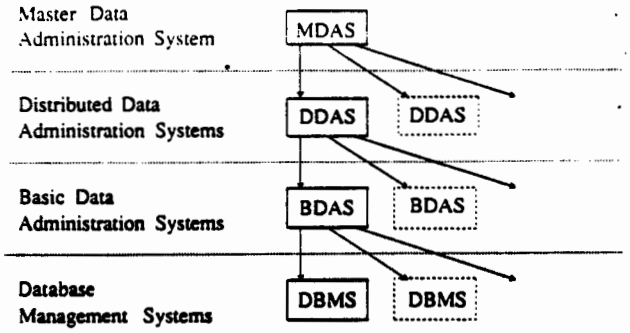

Figure 2. The IMDAS hierarchy.

On each computer system is a basic interface between the local data repositories and the rest of the IMDAS. This is the *basic data administration system* (BDAS). It is also at this level that the access to user data areas is provided. The BDAS and the associated DBMSs *execute* data manipulations.

At the *distributed data administration system* (DDAS), the collection of data repositories managed by a group of BDASs is logically integrated into a *segment* of the global database, using a dictionary describing the distribution of the data. The DDAS becomes the data manager for that segment and *supervises* all manipulations on it. In addition, each DDAS provides the IMDAS interface to some set of the user programs.

In order to integrate segments managed by separate DDASs and to execute user transactions that require this level of integration, a single system is designated the *master data administration system* (MDAS).

5. BDAS—basic data administration system

The BDAS integrates data repositories into the IMDAS. The BDAS must convert the IMDAS internal form of a transaction to that accepted by the DBMS which has to execute it, pass it to the DBMS and interpret the status which comes back. This can be quite complicated since some DBMS use very high-level languages while others are quite primitive. It may be further complicated by the local operating system.

The BDAS also converts any data involved between the DBMS-dependent form and the standard IMDAS form. In addition, the BDAS must access user data areas and convert between the user representation and the standard IMDAS form. The interface to a particular DBMS is encapsulated by a separate process called the command translator/data translator (CT/DT) for that DBMS (Fig. 3).

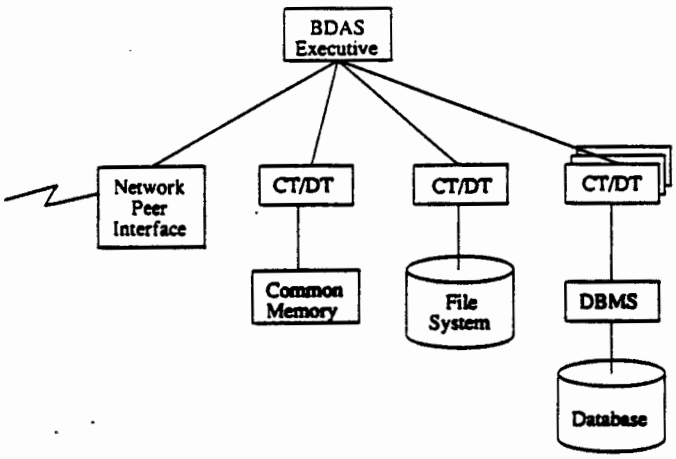

Figure 3. Typical BDAS.

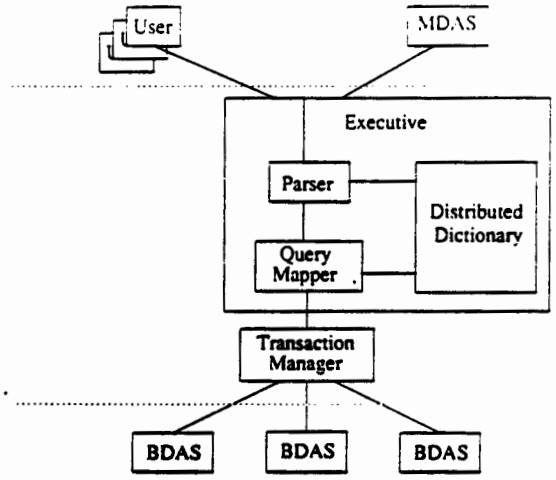

Figure 4. DDAS (between dashed lines).

The BDAS receives commands from and returns status to the DDAS which supervises it, but it also deals with other BDASs as *network peers* for the purpose of delivering data. In this way, data moves directly between the user and the data repositories rather than following the IMDAS hierarchy.

6. DDAS—distributed data administration system

The DDAS:

(*a*) integrates a collection of BDASs and their databases into a segment of the global database, and

(*b*) provides the IMDAS transaction interface to a collection of user programs.

User programs issue transactions to the IMDAS, represented by the DDAS executive, which accepts them, oversees their execution and returns status. User transactions are stated in an SQL-like language. These transactions are parsed into an IMDAS standard internal form and then modified to reflect the differences between the user's *external view* and the IMDAS *global conceptual view*.

The DDAS attempts to map the transaction into a set of operations on elements of the global database which are managed by individual DBMSs. In order to do this, it consults a *fragmentation dictionary* describing how data is distributed over the integrated databases. The result is a set of tasks to be executed by specific DBMSs. If any of the data is outside the segment managed by the DDAS, the transaction is sent to the MDAS. Otherwise the collection of tasks is given to the DDAS transaction manager. When the transaction is completed, status is returned to the user.

The transaction manager schedules execution of the entire transaction based on other activity in the system. The current strategy used is *two-phase locking* (Eswaran *et al.* 1976). This means that one cannot reference something which is currently being modified or referenced. Any transaction which would violate this rule waits until the conflicting transaction completes. If there are no conflicts, transactions proceed in parallel.

In the simplest case, the transaction consists of one task which is sent to one BDAS, and the whole transaction, including interaction with user data areas, is executed there. But more complex transactions may involve distributing different tasks to several BDASs on different computer systems and controlling their timing so that the results of one task can be used as input to another.

7. MDAS—master data administration system

The MDAS is an optional component of the IMDAS which is made necessary by having more than one DDAS. So, from our point of view, the issue of centralized versus distributed control of the distributed databases does not have to be resolved at the outset. If a single system can manage all data activity in an enterprise, one installs the sole DDAS there and makes its controlled segment the whole global database. But when more than one DDAS becomes necessary (as we expect must inevitably occur), rather than trying to solve the crossover problems by committee, we appoint a master DAS.

The MDAS supervises, in a limited way, all DDASs. When a DDAS receives a user transaction which it does not have the data to complete, it passes the transaction to the MDAS. Like the DDAS, the MDAS has a fragmentation dictionary which describes the location of data. However, while the DDAS dictionary describes the actual data repositories, the MDAS dictionary describes the DDASs as the data repositories. Hence the MDAS will

fragment the transaction into a set of tasks to be performed by one or more DDAS, the MDAS transaction manager will schedule the transaction and pass each subtask to the designated DDAS for execution at the proper time. When the transaction is completed, the MDAS will report completion to the originating DDAS, which will report to the user. Consequently, the DDAS executive is both a user of the MDAS services and a subordinate of the MDAS transaction manager.

The internal structure of the MDAS is similar to a DDAS with a simplified fragmentation dictionary. Since the MDAS always receives transactions in internal form (from a DDAS), it does not need a parser. The value of this close resemblance is that an MDAS can be instantiated on any system which has a DDAS, since almost all of the code is common. This allows the choice of MDAS to be made by the system manager when the system is started or resumed after a crash, thus avoiding the single point of failure.

8. Current system

An IMDAS prototype exists and supports another prototype—the Automated Manufacturing Research Facility (AMRF) (Nanzetta 1984) at the National Bureau of Standards in Gaithersburg, MD, USA. The AMRF is a test bed of manufacturing equipment and systems that researchers from NBS, industrial firms, universities and other government agencies can use to experiment with new standards and to study new methods of measurement and quality control for automated factories.

The AMRF implementation consists of one DDAS and a number of BDASs. Together with user programs, the IMDAS communicates over the AMRF network. The IMDAS software is written in C and Pascal and runs in both UNIX® and VMS™ environments. We anticipate minimal effort will be necessary for ports to other systems. The user programs have no language constraints and currently include C, Pascal, Lisp, BASIC and Forth. User systems include Symbolics, Sun Microsystems, HP and VAX™.

ASN.1 (ISO 1984) is used to encode all data units for transmission. This machine-independent representation supports primitive types (e.g. integers, strings) as well as complex user-defined types (e.g. relations, query trees). This enhances portability at a minimal expense of time and space and is another example of the use of a single-common interchange format.

At the BDAS level, interfaces to a variety of data systems exist, including RTI/Ingres (Ingres 1986) and BCS/RIM (Boeing 1985). In addition to commercial DBMS, there also exist interfaces to the AMRF geometry modeling system (Hopp 1987) and the AMRF Process-Planning System (Brown and McLean 1986) and to common memory (Libes 1985) and file systems for UNIX and VMS.

Near-term plans include expanding the system to include the IBM 4381 running VM, building the interface to SQL/DS (IBM), and constructing the first MDAS. Existing subsystems of the IMDAS are undergoing continuing testing and refinement. We envisage that a number of efforts in improving performance and reliability will become important as the project is transferred from the research environment to actual users.

Credits

The IMDAS was designed and implemented by members of the Database Systems Research and Development Center at the University of Florida, and the staff of the Factory Automation Systems Group at NBS.

The IMDAS is partially supported by funding from the Navy Manufacturing Technology Program.

Mention of commercial products in this paper is used only to adequately specify the experimental facility. It does not imply endorsement by NBS nor does it imply that it is necessarily the best available for the purpose.

References and related readings

ANSI, 1986, American National Standard Database Language SQL. American National Standards Institute, Inc., New York, December.

BARKMEYER, E., MITCHELL, M., MIKKILINENI, K., SU, S. and LAM, H., 1986, An architecture for an integrated manufacturing data administration system. NBSIR 863312, Gaithersburg, MD, USA, January.

BOEING 1985, *Boeing RIM User's Manual*, Version 7.0, 20492-0502, Boeing Computer Services, Seattle, WA, USA.

BROWN, P. and MCLEAN C., 1986, Interactive process planning in the ARMRF. *Proceedings of the 1986 ASME Winter Annual Meeting*, Anaheim, CA, USA, December.

ESWARAN, K. P., GRAY, J. N., LORIE, R. A. and TRAIGER, I. L., 1976 The notion of consistency and predicate locks in a database system. *CACM* 19, 624–633.

GENERAL MOTORS ADVANCED TECHNICAL STAFF, 1986 Manufacturing automation protocol—a communications network protocol for open systems interconnect. General Motors, Warren, MI. USA, August.

HOPP, T. H., 1987, AMRF database report format: Part model. NBSIR 87-3672, Gaithersburg, MD, USA, November.

IBM, SQL/DS Concepts and Facilities for VMSP. IBM manual GH24-5065.

INGRES, 1986, Relational Technology, Inc, Alameda, CA 94501, USA, January.

ISO, (International Organization for Standardization) 1984, Information processing—open systems interconnection—basic encoding rules for abstract syntax notation one (ASN.1). ISO/TC 97/SC 21 N25, July.

LIBES, D., 1985. User-level shared variables. *Proceedings of the Summer 1985 USENIX Conference*, Portland, Oregon, USA, June.

NANZETTA, P., 1984, Update: NBS research facility addresses problems in set-ups for small batch manufacturing. *Industrial Engineering*, June, 68–73.

SU, S., 1985 Modeling integrated manufacturing data using SAM*. *Proceedings of GI-Fachtagung*, Karlsruhe, FRG. reprinted as *Datenbank-Systeme fur Buro, Technik und Wissenschaft*. (New York: Springer-Verlag).

Trademark acknowledgments

UNIX is a registered trademark of AT & T. VAX and VMS are trademarks of Digital Equipment Corp.