8621033

Cammarata, Stephanie Jo

AN OBJECT-ORIENTED DATA MODEL FOR MANAGING COMPUTER-AIDED DESIGN AND COMPUTER-AIDED MANUFACTURING DATA BASES

University of California, Los Angeles PH.D. 1986

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

UNIVERSITY OF CALIFORNIA

Los Angeles

An Object-Oriented Data Model for Managing Computer-Aided Design and Computer-Aided Manufacturing Data Bases

A dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science

by

Stephanie Jo Cammarata

1986

The dissertation of Stephanie Jo Cammarata is approved.

Harold Borko

Alfonso F. Cardenas

D. Stott Parker

Clay R. Sprowls

Michel A. Melkanoff, Committee Chair

University of California, Los Angeles

1986

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGEMENTS

I always claimed that I would keep my acknowledgements short and brief, but as I near completion, I realize just how many individuals contributed in a significant way to this doctoral dissertation.

First of all, I want to thank the members of my committee for their helpful comments and suggestions. In particular, I am most grateful to my advisor, Professor Michel Melkanoff, for his excellent guidance, encouragement, and technical expertise. His unending enthusiasm and interest in this work were vital at times when my motivation started waning. Rosetta Lindsey and Verra Morgan deserve much credit for easing the burden of UCLA's administrative details.

This work would not have been possible without the assistance of CAD/CAM and data management personnel at Lockheed, Rockwell, and Hughes. Observing CAD/CAM data management in practice was invaluable for the success of this project.

Many thanks to people at The Rand Corporation for providing a working environment which promotes professional development. Special appreciation goes to three of my Rand managers: Randy Steeb, Phil Klahr, and Lou Miller. In addition, I wish to acknowledge the excellent support offered by Rand's Information Sciences Laboratory, in particular, the services of Jim Guyton, Terry

West, and Colleen Collins.

A few very special people helped me keep this work in perspective and supplied some much appreciated recreation. Sincere thanks to this "group of seven" who have seen me through both elated and frustrated times.

My parents are very precious people who deserve more gratitude than I could ever repay. They were an unlimited source of encouragement over the years.

Finally, I could never express all the ways in which one individual, Dave McArthur, contributed to this effort. He devoted much time, energy, patience, and understanding to this cause. His soothing words in times of need will never be forgotten.

VITA

	Born, Batavia, New York
1974	B.A., New York State University College at Geneseo
1974-1976	Programmer Analyst, Community Bank, Montebello, CA
1976-1978	Research Fellowship, University of Pennsylvania
1978	M.S.E., University of Pennsylvania
1978-1986	Computer Scientist, The Rand Corporation, Santa Monica, CA
1984-1985	Lockheed Corporation Fellowship

PUBLICATIONS

Hayes-Roth, B.,Hayes-Roth, F., Rosenschein, S., Cammarata, S., "Modeling planning as an incremental opportunistic process", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo (August 1979).

Thorndyke, P., McArthur, D., Cammarata, S., "AUTOPILOT: A distributed planner for air fleet control", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver (August 1981).

Cammarata, S. "Deferring updates in a relational data base system", *Proceedings of the Seventh International Conference on Very Large Data Bases*, Cannes, France (September 1981).

McArthur, D., Thorndyke, P., Cammarata, S., "A framework for distributed problem solving", *Proceedings of the Third Annual National Conference on Artificial Intelligence*, Pittsburg (August 1982).

Cammarata, S., McArthur, D., Steeb, R., "Strategies of cooperation in distributed problem solving", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, (August 1983).

Cammarata, S., and Melkanoff, M., "An Interactive Data Dictionary Facility for CAD/CAM Data Bases", *Expert Database Systems*, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

Zaniolo, C., Ait-Kaci, H., Beech, D., Cammarata, S., Kerschberg, L., "Object-oriented database systems and knowledge systems", *Expert Database Systems*, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

ABSTRACT OF THE DISSERTATION

An Object-Oriented Data Model for Managing Computer-Aided Design and Computer-Aided Manufacturing Data Bases

by

Stephanie Jo Cammarata

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1986

Professor Michel A. Melkanoff, Chair

As a result of strong and steady CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing) growth over the past 20 years, special facilities for managing design and manufacturing data have been required. CAD/CAM Data Base Management Systems (DBMS) fill this role. The most widely used CAD/CAM DBMS manage data for only a single CAD or CAM application and cannot integrate graphical, geometrical, manufacturing, and administrative data. Furthermore, current modeling facilities are inadequate for representing semantic features and constraints captured by an engineering drawing. These limitations cause data flow gaps, inconsistent and redundant data, and unnatural data organization in existing CAD/CAM data bases.

The purpose of this dissertation is to develop sophisticated facilities for managing CAD/CAM data bases. This work focuses on mechanical design, engineering, and manufacturing, specifically *product definition data* generated

during initial design phases. Based on a detailed analysis of CAD/CAM data management requirements, and interaction with data management and manufacturing personnel at Lockheed Corporation and Rockwell International, I propose the following goals for integrated CAD/CAM DBMS:

- conceptual centralization
- part-oriented BOM hierarchies
- customized representation of assemblies and parts
- incorporation of domain knowledge

The product of this research is the theoretical design of an object-oriented data model, ODM, and the implementation of an ODM computer software prototype supporting CAD/CAM DBMS goals. The ODM software system is written in T, a lexically scoped dialect of Lisp, and currently runs on Vax and Apollo networks in UCLA's Computer Science Department. The ODM system provides the following unique features:

- object-oriented semantic modeling facilities
- dynamic schema capabilities
- semantic constraint maintenance
- heterogeneous data types

I conclude with an evaluation of ODM toward achieving the goals of integrated CAD/CAM DBMS. Data bases supporting Hughes' PWA (Printed Wiring Assembly) and *Producibility Feedback* applications were obtained for evaluation testing. Although most discussion concentrates on mechanical manufacturing; the developed methodology and tools for CAD/CAM data management also apply to other design and manufacturing domains such as architecture and electronics.

CHAPTER 1
INTRODUCTION

The goal of this dissertation is to develop sophisticated data management facilities for maintaining Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM) data bases. The product of this research is the theoretical design of an object-oriented data model, ODM, and the implementation of an ODM computer software prototype. This work focuses on mechanical design, engineering, and manufacturing, specifically *product definition data* generated during initial design phases. Detailed analysis of CAD/CAM application and data management requirements were conducted to produce the ODM functional specifications. This document presents the results of this analysis and an evaluation of ODM toward achieving the goals of integrated CAD/CAM data management systems. Although most discussion concentrates on mechanical manufacturing; the developed methodology and tools for CAD/CAM data management apply to other design domains such as architecture and electronics.

1.1 History of CAD/CAM and CAD/CAM DBMS

The first CAD systems were essentially computer drafting systems. In the early 1960s, general purpose graphics software and self-contained drafting workstations were introduced. Ivan Sutherland's Sketchpad [Sut65] system provided the theoretical foundations for future graphical representation. CAD systems entered the commercial market in 1963 when General Motors announced its first CAD workstation, DAC/1 (Design Augmented by Computers) [Tei85]. By the late 1960s, major aerospace corporations like Lockheed, McDonnell Douglas, and Boeing began to explore the use of computer graphics for aircraft and missile design.

CAM systems originated in the 1950s when Numerical Control (NC) machines were designed and built. In the 1960s Lockheed-Georgia started integrating CAD and CAM by using computer drafting systems for NC part programming. It wasn't until the 1970s that CNC (Computer Numerical Control) and DNC (Direct Numerical Control), as we know them today, were introduced to the manufacturing industry.

CAE (Computer-Aided Engineering) is another critical aspect of mechanical CAD/CAM environments which has become increasingly sophisticated. Engineers now rely on computer programs for structural analyses such as finite element and load stress analysis. Simulation of motion, friction analysis, and tolerance analysis enable the study of dynamic characteristics and behavior before production line fabrication and assembly is initiated.

Administrative and business accounting systems contribute to another segment of automation in the manufacturing industry. These systems maintain inventory, billing, and purchasing functions as well as employee systems such as personnel and payroll. Steadily over the past 20 years, comprehensive software packages are computerizing most administrative tasks.

Many independent computer application systems, such as those described above, have been built to support and promote CAD/CAM technology. Although some are specific to manufacturing and others are general purpose, each of these applications requires input data and produces results as output. The sources and types of data, and input and output methods, vary considerably. Until recently, the benefits of automating application tasks outweighed the cost of data preparation and dissemination. However, as the scope and use of these systems has increased, production inefficiencies are resulting from the overhead of data access, preparation, and distribution. In most cases, personnel extract input data from hard-copy worksheets or reports and manually code it to conform to the specifications of the software system. Because application systems are generating their own specialized data bases, managing the storage and archival of magnetic and hard-copy data becomes a task in itself. It is estimated that personnel spend 10-30% of their time searching for data sets; not necessarily accessing the data, but simply trying to determine which report, file, or data base contains a particular piece of information [Mel84].

In the future, the role of the computer will be amplified. A general consensus in the manufacturing industry is that a Computer Integrated Manufacturing System (CIMS) is the key to increased productivity [Mel84, Hes83]. New generation applications like expert systems for production control and process planning, Flexible Manufacturing Systems (FMS), and robotics, are performing decision-making tasks. However, the potential benefit from intelligent systems can only be achieved if data management inefficiencies are overcome. Integrated CAD/CAM DBMS can help solve the information bottleneck by streamlining the exchange of data between computer application systems.

1.2 Scope of this research

Based on a detailed analysis of CAD/CAM data management requirements, I identified four desirable goals of integrated CAD/CAM DBMS. These goals, presented in Chapter 2, promote effective generation and utilization of CAD/CAM data throughout the entire manufacturing life cycle. Three aspects of my requirements analysis included: (1) reviewing the current state of CAD/CAM DBMS tools and technology, (2) observing CAD/CAM data management in practice at three major aerospace corporations, and (3) projecting ahead to identify future data management needs for supporting next generation CAD/CAM application systems. I discovered that the *engineering drawing* is the main source of data in a CAD/CAM environment. Unfortunately, existing data management tools cannot represent the semantic information which designers, engineers, and manufacturers repeatedly extract from an engineering drawing. This research proposes data management methodologies capturing the conceptual organization of CAD/CAM data represented in an engineering drawing.

In Chapter 3, I discuss four DBMS capabilities contributing to the high-level goals of integrated CAD/CAM DBMS. Each of these features addresses a limitation in current data models and DBMS implementations. Based on my requirements analysis, I concluded that an *object-oriented* data model best fulfills the structural organization of CAD/CAM data. Chapter 4 describes object-oriented models adopted by programming languages, data management, and knowledge representation. I review the evolution of object-oriented systems and their resulting strengths and weaknesses.

The theoretical design of the object-oriented data model, ODM, is presented in Chapter 5. ODM, based on set theory and predicate logic, overcomes many deficiencies of previous object-oriented representations. Objects in the model are constructed from four basic components. Primitive relationships between components establish aggregation and generalization networks, and ODM inferences are derived from these networks.

The implementation of an ODM computer software prototype is detailed in Chapter 6. ODM is implemented in T, a lexically scoped dialect of Lisp, and currently operates on Vax and Apollo networks in UCLA's Computer Science Department. I discuss data entry and data manipulation languages which I developed for interfacing with the ODM software system. Dialogues of direct interaction with the system, presented in Chapter 6, serve as proof of concept by demonstrating how ODM fulfills the functional specifications prescribed in Chapter 3. Examples of heterogeneous and hierarchical data types, semantic constraints, and dynamic schemata in the ODM prototype system are described.

Chapter 7 discusses related CAD/CAM DBMS efforts in research and industrial environments. The objectives of corporate CAD/CAM DBMS projects differ significantly in scope and depth compared to the goals of research groups. In addition to CAD/CAM applications, I also review work addressing two extended DBMS capabilities: semantic constraint maintenance and dynamic schema facilities.

Evaluation of ODM and its prototype is presented in Chapter 8. Two application systems at Hughes serve as a test bed for evaluating ODM's goals. I demonstrate that ODM is sufficient for maintaining existing data bases extracted from Hughes' PWA (Printed Wiring Assembly) application. More importantly, I show how ODM supports a conceptual organization of PWA data most natural to design and manufacturing experts. ODM also promotes effective management of semantic data; nonexistent in current PWA data bases and management systems. The *Producibility Feedback* system at Hughes also benefits from the novel capabilities offered by ODM.

Chapter 9 concludes with the contributions of this research, limitations of the existing version of ODM, and directions for future research. I also discuss the applicability and relevance of this work to other domains. To aid the reader, a list of abbreviations and acronyms used in this document can be found in Appendix A.

CHAPTER 2

MOTIVATION AND GOALS

Computer-Aided Design and Computer-Aided Manufacturing (CAD/CAM) are indispensable in today's industrial centers. As a result of strong and steady CAD/CAM growth over the past 20 years, facilities for managing design and manufacturing data have been required. CAD/CAM Data Base Management Systems (DBMS) fill this role. In the following section, I introduce CAD/CAM DBMS by discussing the evolution of three different categories of CAD/CAM DBMS. Two of the categories, data bases for CAD drafting systems and data bases for geometric modeling systems, are used extensively but are limited in scope and functionality. This dissertation focuses on the third category, integrated CAD/CAM DBMS, which are rapidly emerging in design and manufacturing industries.

2.1 Evolution of CAD/CAM DBMS

CAD/CAM Data Base Management Systems maintain data used during design and manufacturing operations. The sophistication of these DBMS varies tremendously. The oldest and most widely used systems manage data for only a single CAD or CAM application. The two most popular applications which include facilities for data management are CAD drafting systems and geometric modeling systems. Below I discuss the uses of these systems and the role of their associated data bases.

2.1.1 CAD drafting systems

CAD drafting systems provide tools to generate engineering drawings on a graphics monitor. Facilities for drawing lines, curves, and other graphical entities help designers build a graphical model of a part or assembly. Data is usually entered into a graphics workstation using menus, function keys, and optional command language. Because these systems represent an object graphically, they are used mainly for initial generation of drawings and for future display of the designs. Automatic reproducibility of a drawing reduces the dependency on the traditional *engineering blue print*. With a CAD drafting system, drawings can be displayed at any time on a single workstation or on any remote workstation. Also, if the CAD system supports a graphics standard, such as IGES (Initial Graphics Exchange Specification) [Ini83], the graphical data can be transported to other CAD systems and displayed. The most popular drafting systems are CADAM, Computervision, Applicon, and Calma. At Lockheed, statistics have shown that turnaround time for a design has been reduced by 30% since the introduction of the CADAM drafting systems [Nas83].

Recent innovations in graphics hardware and software have advanced the development of sophisticated drafting systems [Mac80, Tei85]. Display facilities usually include graphical transformations such as scaling, translation, and rotation. On many systems, three-dimensional transformations are available for generating multiple perspectives. However, the data bases of drafting systems are system dependent. Except where data has been translated into a standard format, like IGES, the data bases can only be accessed and manipulated for graphical display. These systems are self-contained and impenetrable; therefore, it is very difficult, if not impossible, to extract symbolic information from

CAD data bases. Queries about graphical entities, such as lines or points are not supported. Textual information which is entered on a drawing cannot be easily accessed. For example, designers at Lockheed, using CADAM, must enter Bill of Materials (BOM) and Parts List (PL) data twice on the drawing and a third time as input to their Computerized Parts List (CPL) system. It is impossible to retrieve the BOM and PL data for use by other application systems. Systems such as CADAM and Computervision have additional facilities for geometrical computations from the graphical model, such as volume, surface area, moments of inertia, and structural analyses. They do not, however, provide the sophisticated modeling facilities of dedicated geometrical modeling systems, discussed in the next section.

2.1.2 Geometric modeling systems

Geometrical modeling systems generate a mathematical model of a three-dimensional part based on its geometric properties. Unlike drafting systems whose input is graphical, the input for geometrical systems is textual or procedural. In some systems, a graphical display may be produced as a visual aid to the designer, but the primary representation is in terms of geometrical properties. Many different types of geometrical models have been developed. The two best understood and most important representation schemes are boundary representations (B-rep) and constructive solid geometry (CSG) [Req]. B-rep models represent solids indirectly by explicitly representing the solid's topological boundary rather than the solid itself. A solid is modeled as a boundary representation by segmenting its boundary into a finite number of bounded subsets called *faces* or *patches*, and representing each face by its bounding *edges* and *vertices*. Figure 2.1 shows the boundary representation of a rectangular pyramid using a triangulation method. Intergraph, Calcomp, and Autotrol [Tei85] are three companies offering geometric modeling systems based on variations of B-rep models.

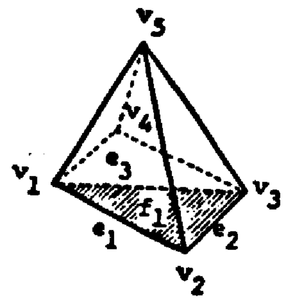


Figure 2.1 B-rep model for a rectangular pyramid

In a CSG model, solids are defined as combinations of solid *building blocks* similar to volumetric addition and subtraction. The representations are ordered binary trees whose non-terminal nodes represent set operators such as union, intersection, and difference; and whose terminal nodes are the building blocks representing regular solids such as cube, sphere, and rectangular solid. Figure 2.2 exemplifies a CSG representation. PADL [Voe], one of the original CSG systems, was developed at the University of Rochester. Both GMSolid [Tei85] and McDonnell Douglas' UNISOLID [Tei85] are based on the PADL system.

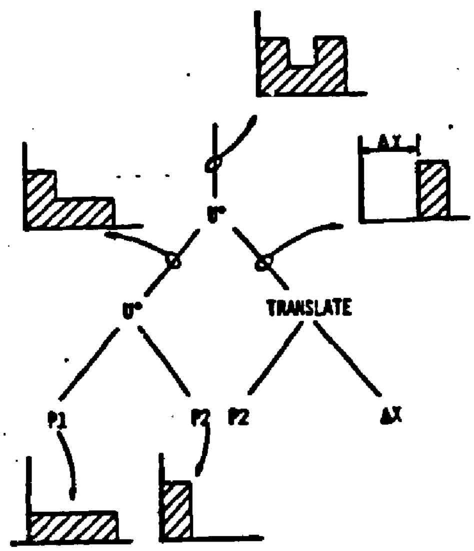


Figure 2.2 CSG tree

The analysis tools of geometric modeling systems compute basic engineering properties such as surface area, volume, and center of gravity. Finite element models are also generated from geometrical data bases for analysis of properties such as heat flow and elastic deformation. Unfortunately, geometric modeling systems suffer from the same drawback as CAD drafting systems. Their data bases are maintained in system dependent formats, therefore, data cannot be exchanged among other application systems. Only the analysis tools within one package can be applied to the data sets produced by that package. It is rare to be able to transport a geometrical data base from one modeling system to another. These systems do not support interactive access and query capabilities for geometric entities such as surfaces, edges, and vertices. Although there is a direct correspondence between the graphical representation of a part and its geometric representation, this relationship is generally not captured in the data bases. For instance, if the dimensions of a part are modified using a CAD drafting system, these changes affect the *graphical* data base. If a separate *geometric* modeling system is employed, corresponding changes in the geometry input data are also necessary. Until the appropriate modifications are made to both data bases, prior geometric analyses (based on previous graphical data) are no longer valid.

2.1.3 Integrated CAD/CAM DBMS

In the previous sections I have emphasized the difficulties of trying to integrate data bases used for drafting and geometric modeling. General purpose DBMS used in other facets of design and manufacturing such as BOM processing, process planning, and inventory all share this same limitation. In part, data management inefficiencies have resulted from the growth of application programs over the past 20 years. As CAD/CAM systems were developed, the primary goal was to automate a design or manufacturing task. The data flow to and from an application system was regarded as a minute operational detail, rather than a critical consideration. To overcome these *data flow* gaps, industries must focus on the task of data management as an integral part of the design and manufacturing life cycle, not merely a process driven by an application system.

The mandate of future integrated CAD/CAM DBMS is to facilitate access by humans and computer programs to information required in design and engineering, production planning and manufacturing, and administrative and business operations. During my analysis and evaluation of CAD/CAM data management systems, I did not find any fully integrated operational systems. This ambitious aim entails four CAD/CAM DBMS goals which I present in the following section.

2.2 Goals of integrated CAD/CAM DBMS

Manufacturing corporations are looking toward integrated CAD/CAM DBMS for achieving a Computer Integrated Manufacturing System (CIMS). New DBMS capabilities and functionality cannot, however, be formulated without a detailed analysis of information management needs. CAD/CAM data management techniques range from formal data entry methods to informal report distribution. For this research, employees at Lockheed Corporation and Rockwell International assisted me in the requirements analysis I present below. At both corporations, I interviewed designers, engineers, and manufacturers, in addition to data management personnel.

The first objective of these site visits was to understand how a manufactured product is represented during each of its production phases. At both corporations, I reviewed the content and organization of their data bases, and the types and uses of design, engineering and manufacturing data. I discovered that in addition to specific product data, there is auxiliary data supporting design and manufacturing operations. Another critical aspect of the modeling process is the exchange of data among CAD/CAM systems, between manufacturing processes, and from department to department. Second, I aimed to solicit employee recommendations for improving CAD/CAM DBMS functionality. I queried designers, manufacturing planners, and manufacturing engineers to help isolate deficiencies in their current systems and to request suggestions for improvements. Because I was interested in high-level user needs, I discussed these issues with manufacturing personnel rather than data management employees. During these interviews, I hoped at best, that users would verbalize some desired capabilities or, at worst, I would observe them at work and try to identify limitations of existing systems. I invited suggestions by posing questions of the form: *"What if you had the capability to ...?"*. Using their responses along with my observations, I obtained a good understanding of what is needed in an integrated CAD/CAM DBMS.

Some CAD/CAM DBMS researchers fear that end-users have not been consulted about existing deficiencies and desired improvements [Pro81]. As a result of my meetings at Lockheed and Rockwell, I was able to observe, first hand, CAD/CAM data management in practice. Through discussions and additional analysis, I have identified four key goals which integrated CAD/CAM DBMS should strive to achieve. In Chapter 8 I revisit these goals by evaluating the object-oriented data model, ODM, developed as a result of this research.

2.2.1 Conceptual centralization

Engineering drawings are the source of 90% of the data maintained in a manufacturing industry [Can83]. Drafting and design phases generate the engineering drawing, and throughout the entire manufacturing cycle, data is abstracted from the drawing. In no sense is the data explicitly represented, rather, design and manufacturing personnel must interpret the drawing and extract information relevant to their needs. For instance, a process planner identifies features in the drawing which require sequences of manufacturing processes. An electrical engineer looks for features pertaining to electrical components. A tool designer extracts data necessary for deciding which tools to use for fabrication. These diverse interpretations are recorded in textual verbiage on hard-copy forms and reports, batch-updated master and transaction disk files, and as annotations to hard-copy and on-line engineering drawings. Figure 2.3 shows a simplified chart of data flow at Lockheed [Can83, Lew83, Nas83]. This diagram illustrates the key role which an engineering drawing plays in providing data for other manufacturing systems. It also illustrates how the number and types of data repositories multiply as the design/manufacturing life-cycle progresses.

The absence of data centralization is cited as a major cause of data management inefficiencies [Ahr84, Liu85]. Unfortunately, because graphical data is generated first, it is regarded as the kernel of CAD/CAM data bases. As discussed in section 2.1, graphical representations are system dependent and single-purpose. Data bases from the CADAM system, used widely at Lockheed, are efficient representations for the drawing system, but afford no utility outside the confines of CADAM. Yet, all geometry, dimensions, and notes are recorded in the data bases. Instead of regarding a CAD data base as the kernel, we must adopt neutral representations for design and manufacturing data compatible with the modeling needs of all application systems. One option, discussed below, proposes a single integrated data base for all data management and processing.

Although the benefits of one integrated data base may appear desirable, a general opinion is that a single centralized DBMS is not a pragmatic solution [Bro84]. From a corporate point of view, the overhead involved in building such a system and the subsequent conversion is prohibitive. We are not yet con-

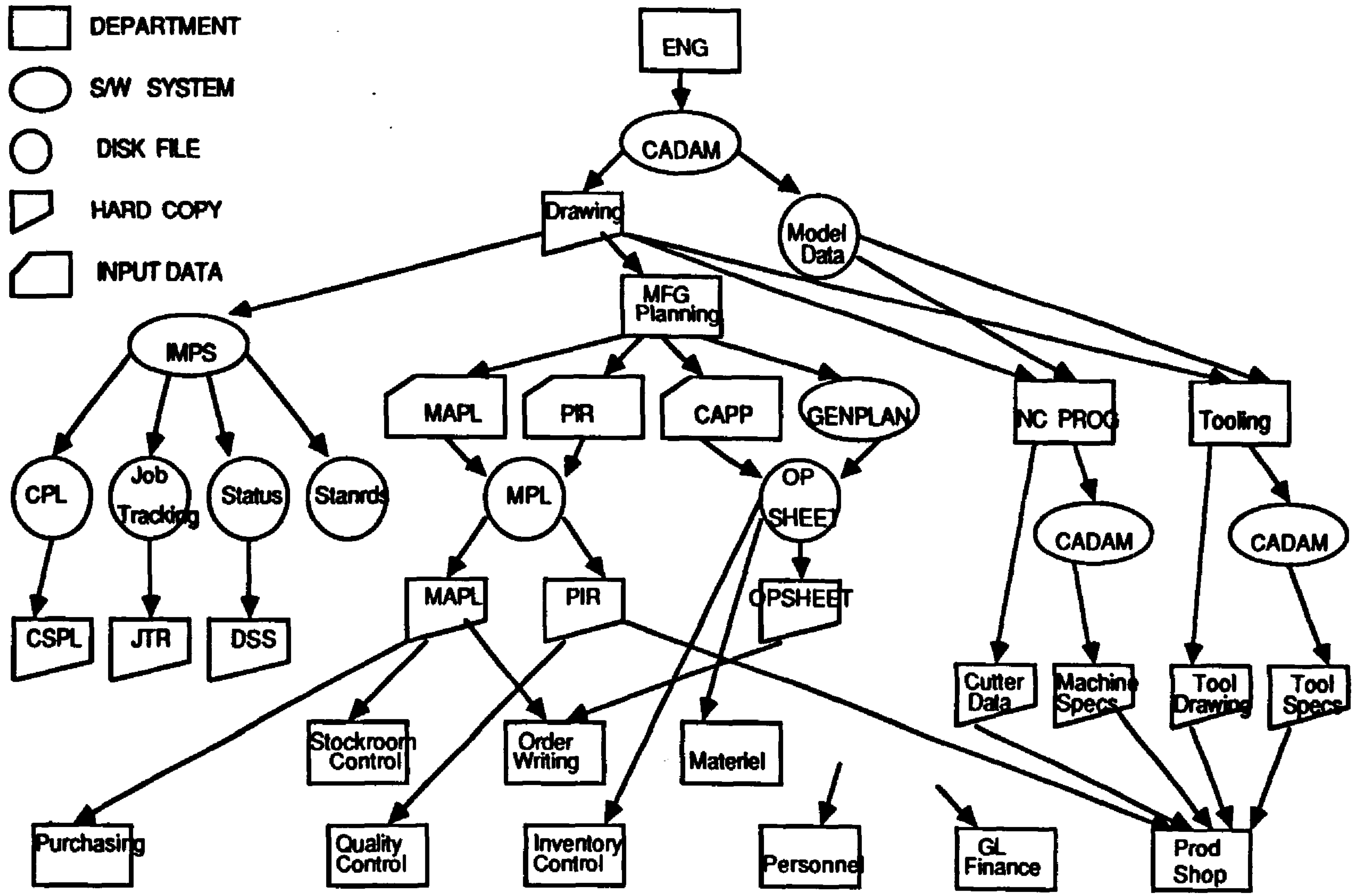


Figure 2.3 Data flow at Lockheed

vinced that a single DBMS can accommodate all the specialized requirements of CAD/CAM data. Furthermore, the amount of CAD/CAM data is voluminous. At Lockheed, a single L-1011 required 150,000 to 200,000 engineering drawings. Relying on a single DBMS under these circumstances is risky. Nevertheless, there are many instances where data can and should be integrated to streamline data flow throughout the manufacturing cycle. In lieu of providing a single, all-encompassing data base, an integrated CAD/CAM DBMS should support *conceptual centralization* of CAD/CAM data. Conceptual centralization does not promote a physically centralized data base, or even a single DBMS, however, it eases the task of accessing and retrieving information by proposing new methods for organizing distributed CAD/CAM data.

One aspect of conceptual centralization provides a directory or *map* for locating sources of data. This goal conceptually merges different data resources such as multiple data bases, computer installations, off-line files, and reports. At Rockwell, the amount of on-line secondary storage is limited, therefore, a semi-monthly archiving to tape is necessary. However, engineers generally need access to data sets for more than two weeks, resulting in a great deal of archive searching and loading. The archive management system at Rockwell is primitive and inefficient; engineers resort to manually scanning magnetic tape listings to locate archived data files. A directory organization which allows references to multiple DBMS, different computer installations, disk files, and hard-copy files would be invaluable [Noc84].

Another aspect of conceptual centralization promotes the integration of *data*. Below I present three interpretations of data integration; each desirable in a CAD/CAM environment and attainable through conceptual centralization.

First, integrating heterogeneous data types such as graphical, mathematical, manufacturing, and administrative data can result in improved efficiency for personnel who must consult different data medium to gather required information. At Lockheed, GENPLAN is an interactive system used by manufacturing planners to generate a process plan for detailed part fabrication [Kam83]. GENPLAN has been hailed as a success and is praised by the manufacturing planners who use it. The program solicits, in a dialogue fashion, product data including features, materials, and treatments. The program, however, is not integrated with any of the data management systems. It is a stand alone program whose output is a process plan. A manufacturing planner using the GENPLAN system must interpret an engineering drawing to extract features relevant to process planning which are requested by the program. The user must also reference additional documents for auxiliary data. Given the complexity of the drawings, considerable efficiency could be gained if the planner had access to data bases containing the required information. For example, if GENPLAN is planning hole drilling processes, the planner may need to respond to a question such as: *What is the diameter and tolerance of a particular hole feature?* For this example, as with many other feature identification queries, it would be easier to pose this query to a data base than to visually scan a drawing for the information.

Furthermore, other departments retrieve the same data in the same fashion. A hard copy of the process plan is included as part of a shop order instruction booklet delivered to the NC programming department, the tooling department, and the production shop. The process plan is also added to a master disk file storing all process plans (the Operation Sheet file). Any modifications to the plan must also be distributed to the departments. If the input and output data of systems like GENPLAN were integrated, subsequent uses of the data would be streamlined.

A second interpretation of data integration refers to the ease of adapting data for use by application programs. Ad hoc approaches in the past maintained separate data bases for individual applications. For *n* application programs and data bases, transferring data between one program and any other requires a total of $n(n-1)$ pre- or post-processors. A more intelligent approach keeps relevant data in a centralized DBMS and interfaces the application programs to the data management system. This method reduces the number of interfaces to *n*. Improved methods for transporting data between independent data bases are in development [Hoo85]. Transport mechanisms which are transparent to users and application programs, further promote conceptual centralization.

The ability to support multiple representations or *perspectives* for the same data object is a third interpretation of data integration [Eas78]. A graphical representation in terms of graphical entities differs significantly from a finite element model used for structural analysis. Nevertheless, both are representations for the same object and should be maintained in a consistent fashion. In current data base models, facilities for multiple views construct different subschemas from elements of the schema. Multiple perspectives differ from multiple views because each representation is complete and self-contained for a given category of data. For example, a graphical perspective completely describes a two-dimensional display of an object; an engineering perspective completely defines a structural model of the object; and an administrative perspective completely specifies an object in terms of its production schedules, marketing goals, and inventories.

In a static *read-only* data base environment, multiple perspectives are compatible with conventional data base access and query operations. However, when we allow updates, we introduce the need for maintaining consistency across all perspectives. The consequences of violating consistency in this environment are costly. Constraints can be imposed within a perspective, such as the following geometric equality: the sum of the angles of a triangular face equal 180 degrees. Constraints must also be enforced across different perspectives. For example, if an object's dimensions are increased, then a modification is also necessary for the amount of raw material needed to manufacture the object.

The lack of data integration results in an enormous amount of data redundancy and the overhead for maintaining consistency is excessive. At Lockheed, most departments maintain their own data, consequently, product data is replicated many times throughout its manufacturing life cycle. Conceptual centralization eliminates replicated data and ad hoc distribution of information because all data bases are available through a centralized directory. The result is a *conceptually*, though not *physically*, centralized view of CAD/CAM data.

2.2.2 Part-oriented BOM hierarchies

Although data sources and data medium vary throughout a manufacturing corporation, a single conceptual organization of design and manufacturing data dominates. It is a recursive object-oriented organization derived from the manufacturing principles that "*assemblies are composed of parts*" and "*parts are composed of features*". This organization typifies a BOM (Bill of Materials) hierarchy. Throughout each phase of manufacturing: design, process planning, fabrication, and assembly, a product is naturally viewed in this hierarchical fashion.

Application systems also regard a *"part"* as the major entity of the data base. Assemblies, sub-assemblies, features, and attributes are described with respect to a given part. Most data retrieval focuses on attributes of a part, rather than all parts exhibiting a given attribute. For instance, a process planner may retrieve the diameter, diameter tolerance, surface finish, and length tolerance of a precision hole of part *x*. However, a request for all hole features with diameter equal to .25, diameter tolerance equal to .001 and surface finish equal to 50 is unlikely.

To aid my understanding of CAD/CAM DBMS limitations, Lockheed compiled a list of 32 desirable queries (Figure 2.4) which their DBMS cannot process directly or interactively [Led83]. Only five of the 32 queries, (4,5,6,12,19), are not keyed on assembly, part, or feature. Three queries, (1,2,3), map directly to a BOM hierarchy. Users would like to access BOM data naturally and intuitively, by navigating through a BOM hierarchy. Unfortunately, the logical and physical data models of CAD/CAM DBMS, including those at Lockheed, do not directly reflect a conceptual BOM organization. Lockheed's IMS data bases, used for their BOM system, represent the *uses* and *used-in* relations in a standard *parts explosion* schema such as the one shown in Figure 2.5. This representation notoriously limits data manipulation during BOM processing. Traversing BOM hierarchies to an unspecified depth is non-trivial, and obtaining a BOM parts list requires an overnight batch job.

(1) What sub assembly/ies does this part relate to?
(2) What assembly/ies does this subassembly relate to?
(3) What are the part requirements for this model?
(4) What is the latest engineering change of this part?
(5) What is the aircraft effectivity of this change letter?
(6) What geometrical changes must be made to satisfy this change?
(7) What vertices compose this edge?
(8) What edges compose this surface?
(9) What surfaces compose this feature/detail?
(10) What features/details compose this part?
(11) What NC path is related to this surface?
(12) What is the geometry of the cutter related to this NC path?
(13) What is the NC access code for this surface?
(14) What is the finish geometry of this surface?
(15) What are the tolerance specifications of this feature?
(16) What are the tolerance specifications of this surface?
(17) What is the grain direction of this part?
(18) What fabrication stock relates to this part?
(19) What standard shape relates to this fabrication stock?
(20) What is the surface normal for this surface?
(21) What is the datum plane of this part?
(22) What general notes relate to this part?
(23) What general notes relate to this feature?
(24) What general notes relate to this surface?
(25) What general notes relate to this edge?
(26) What general notes relate to this vertex?
(27) What are the finish specifications for this part?
(28) What are the process specs for this part?
(29) What are the final condition specifications for this part?
(30) What are the material specifications for this part?
(31) What are the heat treatment specifications for this surface?
(32) What classification code(s) relates to this feature?

Figure 2.4 Lockheed sample queries

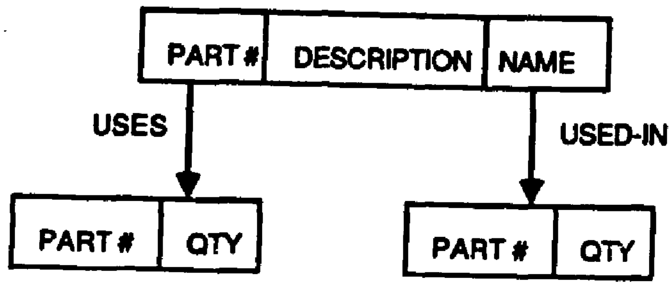


Figure 2.5 Lockheed BOM schema

If we extend the traditional notion of a BOM hierarchy to a more general *containment hierarchy*, we can use this organization to denote relationships between parts and features. In addition to expressing relationships like "*assemblies contain parts*", we can also express the relationship that "*parts contain features*". Four of the Lockheed sample queries, (7,8,9,10), request data about part or feature composition. Figure 2.6 shows a variation of a composition hierarchy for the geometry of a boundary-representation model. In Figure 2.6, objects like surfaces, curves, and points are features of a solid volume.

The remaining 20 queries of Figure 2.4 retrieve attribute data keyed on assemblies, parts, and features. By traversing a BOM and feature composition hierarchy, a user can isolate the assembly, part, or feature in question and access any attribute values of that object. At Lockheed, retrieving attribute data is accomplished through PDDS (Product Design Data System) installed in 1981 [PDD83]. PDDS is one phase within a corporate-wide IDB (Integrated Data

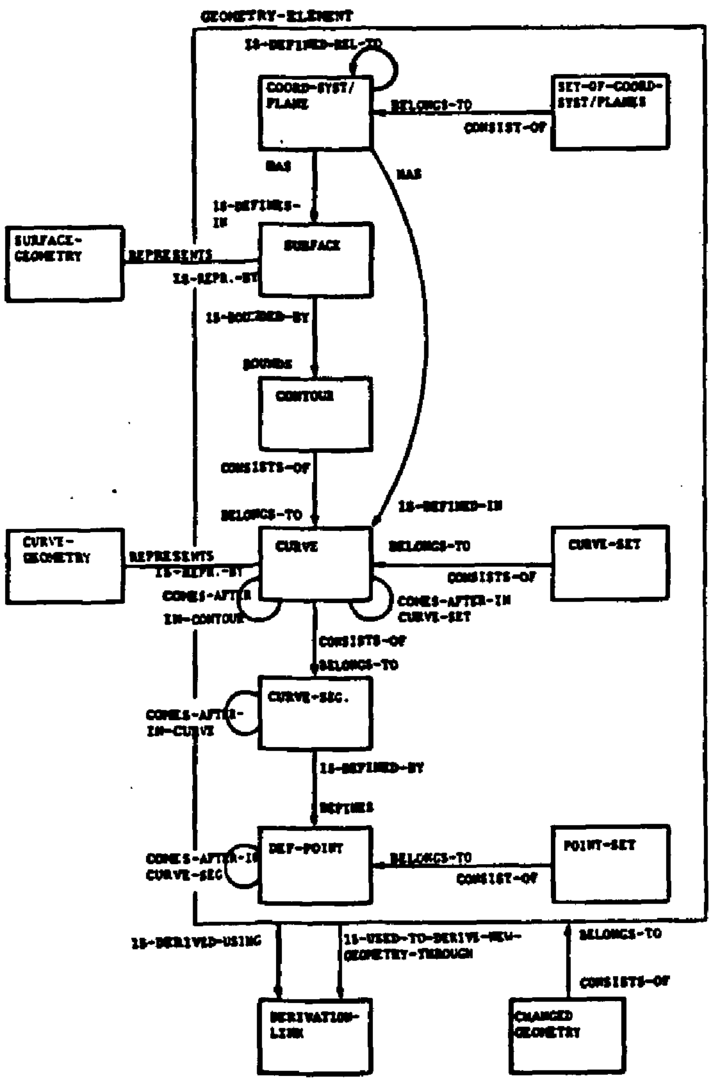


Figure 2.6 Boundary representation model

Base) project initiated in 1976. PDDS consists of four IMS data bases used by engineers during design phases. It operates interactively via menus, function keys, and formated output displays. To retrieve a particular data item, a user must determine which formated display contains the requested data, and retrieve that display for the given part. A variety of formated displays are available and interactive validation of input data is performed. Nevertheless, this mode of user interaction is a holdover from manual methods emphasizing forms and reports. For some functions, such as initial parts list input, the form approach is ideal. Usually, however, the formated screens present more information than necessary. This method of data retrieval reduces the efficiency of both the user, by necessitating visual data filtering; and the DBMS, by excessive data retrieval and formatting overhead. Currently, PDDS resides only within the engineering department. Hard copy reports are still the means of data communication between engineering and other departments [Lew83].

Reviewing Lockheed's sample queries and analyzing the conceptual organization of manufacturing data has served two purposes. First, these activities have justified my intuitions that physical *composition* and *containment* are central themes underlying design and manufacturing processes. Second, I observed a gap between the *conceptual* composition models used by personnel for manipulating data, and the *logical* DBMS models. In a domain so influenced by the composition and aggregation of physical objects, it is important to maintain the conceptual model by structuring data in a way that reflects its natural organization. An integrated CAD/CAM DBMS supporting part-oriented BOM hierarchies can bridge this gap. It will also encourage the ubiquitous *composition* methodology already practiced in CAD/CAM environments. The representation, however, must also be robust and comprehensive. It must be robust enough to respond to queries that are *not* object-oriented, and to allow data access, not only by traversing up or down the BOM hierarchy, but also by entry into the middle of the hierarchy. A comprehensive model will support other data in addition to part-oriented data. For instance, if relevant information is most naturally represented in a relational format, it should be possible to emulate a relational table or call an auxiliary relational DBMS.

2.2.3 Customized representations of assemblies and parts

Data bases of commercial applications differ from CAD/CAM data bases in a number of ways. One difference reflects the structured nature of commercial applications compared to the unstructured design reality modeled by CAD/CAM DBMS. The data in commercial applications is relatively static in format. For instance, in a bank data base, an *account* is the primary data entity. An account has an associated account number, customer name and address, balance, interest rate, and statement date. These data fields are well-defined and all accounts conform to this format. Although values of attributes like balance and rate vary over time, the relationships and data fields are fixed. In a CAD/CAM environment, the format and organization of the data *differs* from entity to entity. Objects, features, attributes, and relationships are assembly- and part-specific providing little uniformity in the structure or content of the data. To model each assembly or part, designers express many unique relationships which differ from part to part. No fixed set of relationships describes all entities. For example, all commercial airplanes do not contain the same parts; moreover, each L-1011 was customized and therefore contained different specifications. Few empirical studies have been devoted to the detailed structure of design, and formal literature on the subject is minimal. In practice, design procedures are determined by personal judgement and conventional methods, with few actions based on or derived from formal considerations [Eas78].

The conceptual schema of CAD/CAM data should be viewed as an abstraction of the engineering drawing. The drawing reflects the form, structure, and relationships of entities and features to be manufactured. Likewise, the conceptual model should be capable of depicting important design data and relationships, and facilitate their maintenance. It should aid but not restrict users in their conceptualization of design.

Ideally, this same preciseness should be captured in data management systems. Representation facilities should accommodate multiple design and manufacturing techniques. Representing a B-rep solid model (see Figure 2.6) requires different entities and relationships from those needed for a CSG model. If a rotational part, shown in Figure 2.7, is to be fabricated on a lathe, characteristics about inner and outer contours, faces, and slots are important. However, if a sheet metal part is being designed, (see Figure 2.8), relevant attributes include contour type, contour form, and feature characteristics such as cutouts, flanges, and joggles.

A second difference between commercial and CAD/CAM data results from the dynamic quality of CAD/CAM data. In a commercial data base, data entities, attributes, and relationships can be identified a priori, through a process called Data Base Design. Data base designers analyze the enterprise to be modeled to determine the conceptual and logical data base organization. For instance, the attributes of a bank account, and the relationships between customer

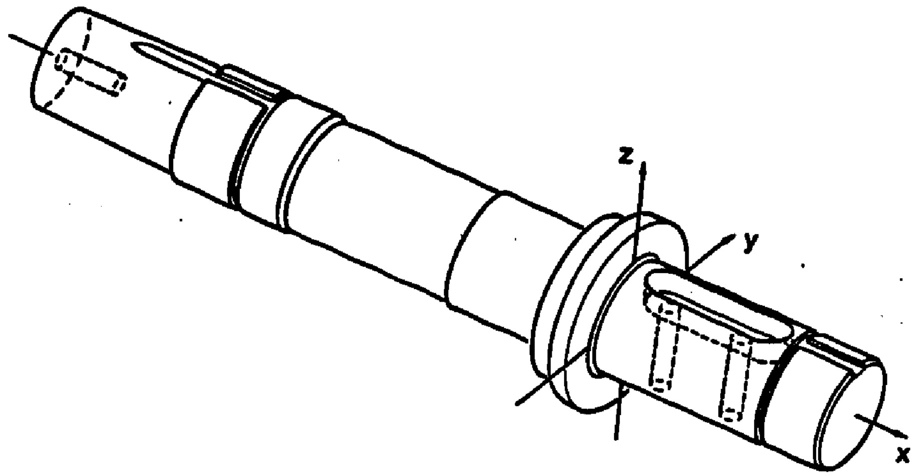


Figure 2.7 Rotational part

accounts and bank assests are established before any accounts are created. In a design environment, the data organization cannot be determined before hand. Engineering design simultaneously defines the structure of a data base and assigns values to the structures. The resulting design data base represents the artifact through many phases from early specifications to manufacturing instructions. The data entities and relationships are generated as a part is designed and

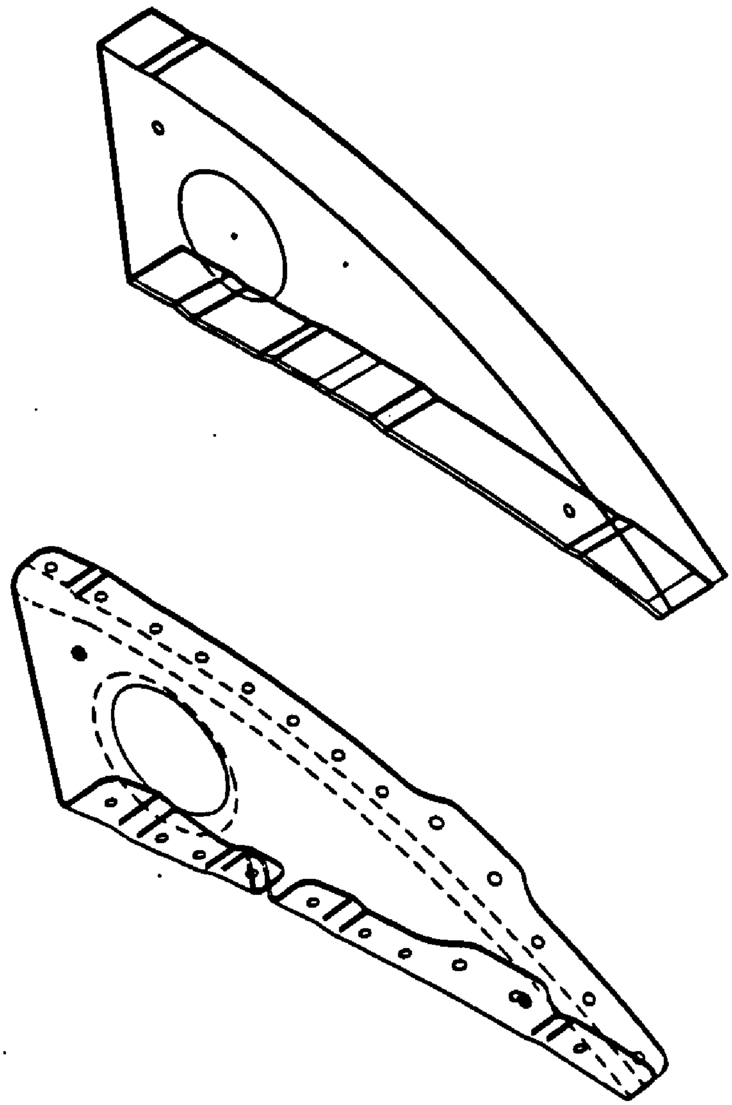


Figure 2.8 Sheet metal part

continue to evolve through the early stages of production.

Existing DBMS have been used successfully in domains where the structure of the data is fixed and constant. However, the use of generalized DBMS for CAD/CAM data has forced data base designers to define neutral organizations which will accommodate all designs. The resulting logical model is exceedingly general to insure that all engineering drawings map onto the logical schema. This phenomena also contributes to the gap between conceptual and logical models described in the previous section. Bridging this gap with existing data management tools means constraining the data to fit into existing models, thereby, losing the fidelity of the conceptual representation.

Customized representations for assemblies and parts help interleave the design of a product with construction of the corresponding data base. If manufacturing features such as flanges, webs, and cutouts are the building blocks for designing a detailed part, then data base objects representing these entities should also be available as building blocks of the data base. If objects exhibit unique properties and relationships, it should be possible for designers to record this information in a data base. Different composition of entities results in different attributes and relationships [Eas78]. At Lockheed, the structure of the IMS BOM data bases is static; therefore, critical design information which design engineers used to design a part, or process planners used to define a fabrication plan, is lost [Lew83].

2.2.4 Incorporation of domain knowledge

CAD/CAM data base users expressed a desire for including application knowledge within a data base system. Two strategies, presented below, can be pursued for adding more domain knowledge to CAD/CAM DBMS.

The first strategy augments CAD/CAM data bases with corporation and industry standards, and provides facilities for automatic enforcement. *Corporation* standards are conventions which the corporation has established and wishes to enforce. *Industry* standards are recommended guidelines for specifications of product attributes or processes. In every design field there are conventional means for treating common situations, usually described in handbooks and manuals. Some examples include: specification of hole tolerances and threading procedures; sequencing of machining operations; acceptable finishing treatments; and feature placement. Having a capability to verify standards and code requirements, during design and data entry, eliminates separate validation procedures.

At Lockheed, textual notes recorded on engineering drawings, including BOM and Parts List data, are verified as an off-line batch job. Generally, initial data is erroneous and must be re-entered. This validation process sometimes requires as many as three iterations, each time returning invalid entries to designers for correction. This iterative process may, in turn, necessitate modification of initially valid data, for compatibility with corrected values. An interactive approach for validating standards information would streamline the design process and reduce turnaround time for data validation. Interactive validation also affords designers some aspects of training as a by-product. Immediate feedback results in fewer errors of the same type in the future.

A second approach for incorporating domain knowledge is to encode design and manufacturing information which supports automated CAD/CAM processes. The CAD/CAM industry is introducing many interactive synthesis and analysis tools. These tools are aiding in tasks which were typically performed manually by manufacturing planners and engineers. Two areas which have shown potential for automation are generation of group technology codes, and process planning. Although these tasks are already semi-automated, in most cases it is necessary for users to manually enter part specification data and parameters for each job. Most of the necessary data resides in some form in the data bases or the engineering drawing. However, to take full advantage of these CAD/CAM tools, it is necessary to represent the data in a form which is amenable to application programs. *Expert system* technology is also helping to support analysis and decision making tasks. In many instances, the knowledge expressed in expert system rules can be incorporated within the schema structure of design data bases.

APPAS (Automatic Process Planning and Selection) is a generative process planning system developed by Chang and Wysk [Cha81]. The system plans milling and hole-making processes by selecting appropriate machining processes for a surface based on surface geometry and accuracy requirements. Figure 2.9 shows a sample dialogue between the system and a designer for planning a hole drilling process. Simple attributes such as reference point, hole diameter, and fillet radius must be entered manually. In most cases, designers are extracting the required data from an engineering drawing. Lockheed's process planning system, GENPLAN, suffers from the same limitations. Tedious and iterative data input would be eliminated if CAD/CAM data bases could represent feature data explicitly. Furthermore, automatic maintenance of feature data promotes global consistency throughout a manufacturing environment. A single interpretation of an engineering drawing, recorded accurately and intelligently, can be utilized by many subsequent CAD and CAM processes.

In the next chapter, I present the functional specifications for an integrated CAD/CAM DBMS supporting the following four goals detailed in the preceding sections.

- conceptual centralization
- part-oriented BOM hierarchies
- customized representation of assemblies and parts
- incorporation of domain knowledge

```
(User's input in lowercase and preceded with ``-->'';
 system response in uppercase)

      --> add

      SURFACE TYPE?

      --> hole

      REF POINT? X, Y, Z

      --> 3.5, 2.25, 2.5

      HOLE DIAMETER?

      --> .125

      CHAMFER? Y OR N

      --> y

      CHAMFER TYPE:
        1. SIMPLE LINEAR + UPPER
        2. OUTER FILLET
        3. INNER FILLER - REVISED

      --> 2

      FILLET RADIUS?

      --> .05

      HOLE LENGTH?

      --> .025
```

Figure 2.9 APPAS interactive session

CHAPTER 3

FUNCTIONAL SPECIFICATIONS

In the previous chapter I described four major CAD/CAM DBMS goals synthesized from my interviews with manufacturing personnel. These goals represent high-level operational aspects of CAD/CAM data management. They are not strictly independent but have many overlapping characteristics. None of the goals correspond directly to a single DBMS function; they are the result of many integral DBMS functions. I have proposed four novel DBMS capabilities contributing to these CAD/CAM DBMS objectives. The functional capabilities detailed below form the basis of the object-oriented data model, ODM, presented in Chapters 5 and 6. In this chapter, I also indicate how each function supports the goals of integrated CAD/CAM DBMS, and how existing DBMS are deficient.

3.1 Object-oriented semantic modeling facilities

The role of modeling systems is to represent and manipulate states of a real or imaginary world in a form as natural as possible. Semantic modeling facilities minimize the gap between the world and an electronic representation of the world. Two aspects of semantic modeling involve the data semantics to be captured, referred to as the schema; and second, the method of representing the semantics, namely, the logical data model. For example, relevant data for an airline reservation might include flight number, origin and destination cities, arrival and departure times, seat assignments, and fare information. These data items may be organized in many different ways; however, the aggregation of this information comprises the *meaning* or semantics of an airline reservation. The method of representation, or *model*, determines how the items are organized. Most commercial DBMS employ the network, hierarchical, or relational model. Below I discuss the need for high-level data semantics and models in CAD/CAM environments.

Capturing data semantics refers to the correspondence between the conceptual meaning of a concept and the representation of the concept. A natural association is best promoted when representational entities express significant or identifying properties of domain concepts. For most CAD/CAM design data, meaningful entities are expressed as assemblies, parts, features, and associated relationships. Graphical and geometrical representations discussed earlier are non-semantic models. Most queries involve data at the assembly, part, and feature level, not at the graphical or geometrical level. Non-semantic models are important for specific tasks like two-dimensional displays, however, for comprehensive and user oriented models, we must also represent entities such as holes, slots, cutouts, and flanges; and provide facilities to manipulate them as domain objects.

An analysis of Figure 3.1 emphasizes the difference between different models. We can identify the item drawn in bold as 3 different entities. Graphically speaking, the item is a circle. If a designer was using a CAD graphics system such as CADAM to produce this drawing, he or she would select a menu item or function key to generate a *"circle"*. In geometric terms, the bold marking in Figure 3.1 is a *"curve"* represented by a mathematical equation. If an engineer was using a boundary-representation system to describe this part, the user would input the equation of a curve. However, from a functional and operational point of view, a manufacturing planner would identify the bold mark as a "*hole*". Therefore, the object which is graphically recognized as a circle and geometrically identified as a curve, has an additional semantic interpretation based on its context and meaning within the engineering drawing.

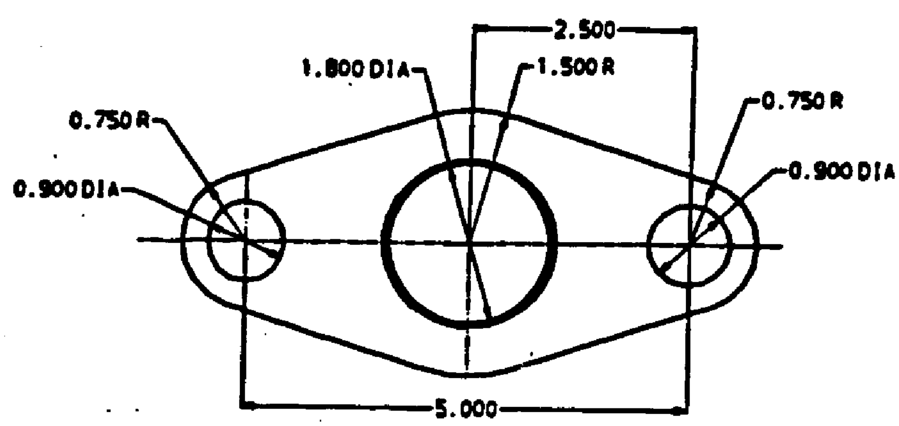


Figure 3.1 Engineering drawing of a gasket

Some research efforts are trying to automatically generate semantic data from graphical and geometrical data. Computer vision and pattern recognition research addresses the problems of object and scene identification from two-dimensional pictures [Win75,Han78]. However, converting an engineering drawing from a two-dimensional graphical image to a geometrical and semantic feature representation is an extremely difficult task. Vision research has not reached the sophistication necessary for these types of cognitive recognition and interpretation tasks. Furthermore, the identification of semantic features and associated properties depends on the interpretation or perspective of the designer or manufacturer. The *hole* feature identified in Figure 3.1 has different meanings to different people, depending on their use of the data. For electricians, a hole indicates a path for electrical wires; to a thermodynamic engineer, a hole means a source of heat loss. Each different interpretation affects the data which is extracted from an engineering drawing and maintained for subsequent use.

Because automatic generation of semantic data is not feasible, or even desirable under certain circumstances, DBMS must provide facilities for entering and managing semantic data directly. A DBMS which supports semantic entities encourages designers and engineers to build a data base for a product at the same time as a graphical model of the part is being generated. Manufacturing personnel who interpret engineering drawings for a specific application, like process planning, can enter or retrieve semantic data relevant to their own task. Data consistency is also promoted by semantic entity representations; semantic features are recorded once, and are then available for other users and application systems.

Data semantics refers to *what* is being represented; semantic modeling capabilities, however, refer to *how* a concept is represented. Established methods generally depend on one of three traditional data models: network, hierarchical, or relational. Three major models have evolved because applications may be more suitable for one data model than another. Advantages and disadvantages of each model are discussed in [Dat81, Ull80, Tsi82, Car79].

As discussed in section 2.3.2, I observed that a BOM hierarchy is the primary organization of design data, and the primary method of data access is through assemblies, parts, and features. Although relationships express information about assemblies and parts, data base users admit that the most frequent way of accessing CAD/CAM data is by entity, not relationship [Liu85]. Therefore, entities representing domain objects should be directly addressable. Nevertheless, CAD/CAM objects also exhibit structural descriptions other than containment or composition. Many descriptions preclude the use of a strict hierarchy, necessitating a network organization relating CAD/CAM objects. For example, geometrical entities, such as points, lines, and arcs compose topological entities, like faces and edges, which in turn compose solid objects. Relationships such as *inside, connected to, bounded by,* and *above*, convey structural descriptions of objects, and carry additional information about the object. For instance, if the relationship *connected to* holds between two beams, it implies that the length of the two connected beams is the sum of their separate lengths.

Current DBMS are used successfully for applications requiring relatively few relationships compared to the large amount of data. Furthermore, in these applications, relationships are constant and uniform across all data instances. In contrast, CAD/CAM data requires complex and part-specific relationships linking heterogeneous data items. Merely expressing M:N relationships is particularly cumbersome and restrictive in a CODASYL network and IMS hierarchy [Dat81, Car79, Enc83, Cod71]. These restrictions limit the semantic power of a representation, resulting in an unnatural modeling environment.

The *ease of use* criterion is becoming an important factor when selecting a data model. Based on this consideration, the relational model has gained popularity for the following reasons. The *table* organization of relational models is conceptually simple; the model supports a high degree of logical data independence; and the use of declarative query languages minimizes physical navigation. However, the relational model is not always compatible with the natural organization of application data. A row in a two-dimensional table represents a mapping of domains. Data access is based primarily on the values of domains denoted in a row of the table. This organization is unnatural and inefficient for CAD/CAM applications where most data access is by part and the primary organization is hierarchical. Forcing data to conform to a relational model can create two situations generally regarded as undesirable: ragged relations with repeating groups and null values [Gut82, Sto84], or expensive join operations across many relations [Ull80].

In theory, the hierarchical and network data models are best suited for representing relationships between assemblies, parts, and features. Unfortunately, existing hierarchical and network DBMS implementations, such as CODASYL and IMS, depend heavily on physical data base organization and cannot directly represent conceptual BOM models. Their DML (data manipulation language) requires procedural navigation through physical data paths. Figure 3.2 presents a simplified BOM listing for an automobile. In this two-dimensional indented format, it is easy to recognize the BOM relationships which exist between different automobile parts.

Car

Body 1

Fender 4

Bolt 6

Engine 1

Piston 6

Ring 3

Valve 12

Crankshaft 1

Bearing 10

Figure 3.2 BOM data for an automobile

Figure 3.3 shows the logical schema for a BOM hierarchy in the network model. Data instances in the network schema are shown in Figure 3.4.

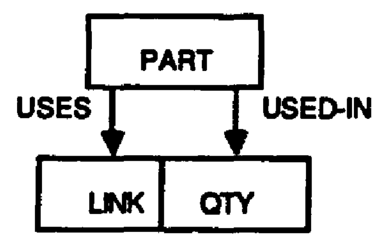


Figure 3.3 BOM schema for CODASYL network model

Similarly, Figures 3.5 and 3.6 are examples of the same schema and data in a hierarchical model. Neither model offers a clear, concise, and aesthetically pleasing graphical representation. Nevertheless, data base designers and data

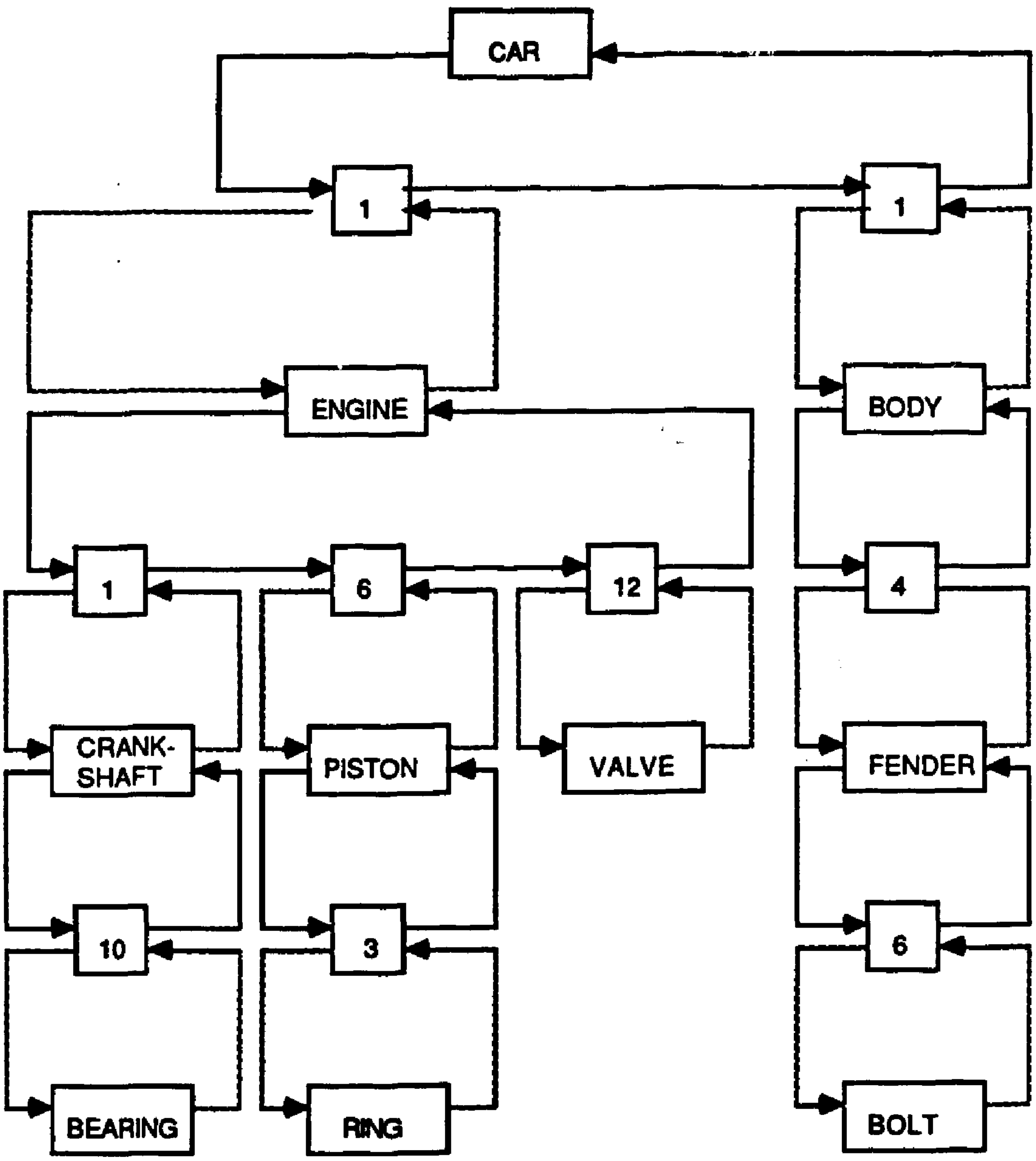


Figure 3.4 BOM data in CODASYL network model

base administrators (DBAs) must produce and manage diagrams in both models which are exceedingly more complex than these examples.

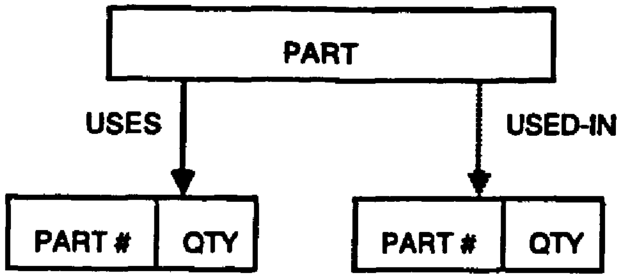


Figure 3.5 BOM schema for hierarchical model

Depicting BOM schema and data in a relational model, as in Figure 3.7, is an improvement. However, any evidence of a hierarchical organization is lost, a major criticism of the relational approach.

The entity-relationship (E-R) data model [Che76] has been used mainly as a data base design tool. The modeling facilities of E-R models allow a closer mapping to the conceptual schema of an enterprise than hierarchical or network models. In the E-R data model, an entity set represents the generic structure of an entity or object, and a relationship set represents the generic structure of relationships among entity sets. So far, the E-R model best represents semantic information, and many current DBMS projects are based on the E-R model [Bor80]. In Chapter 7, I review some of these efforts in detail.

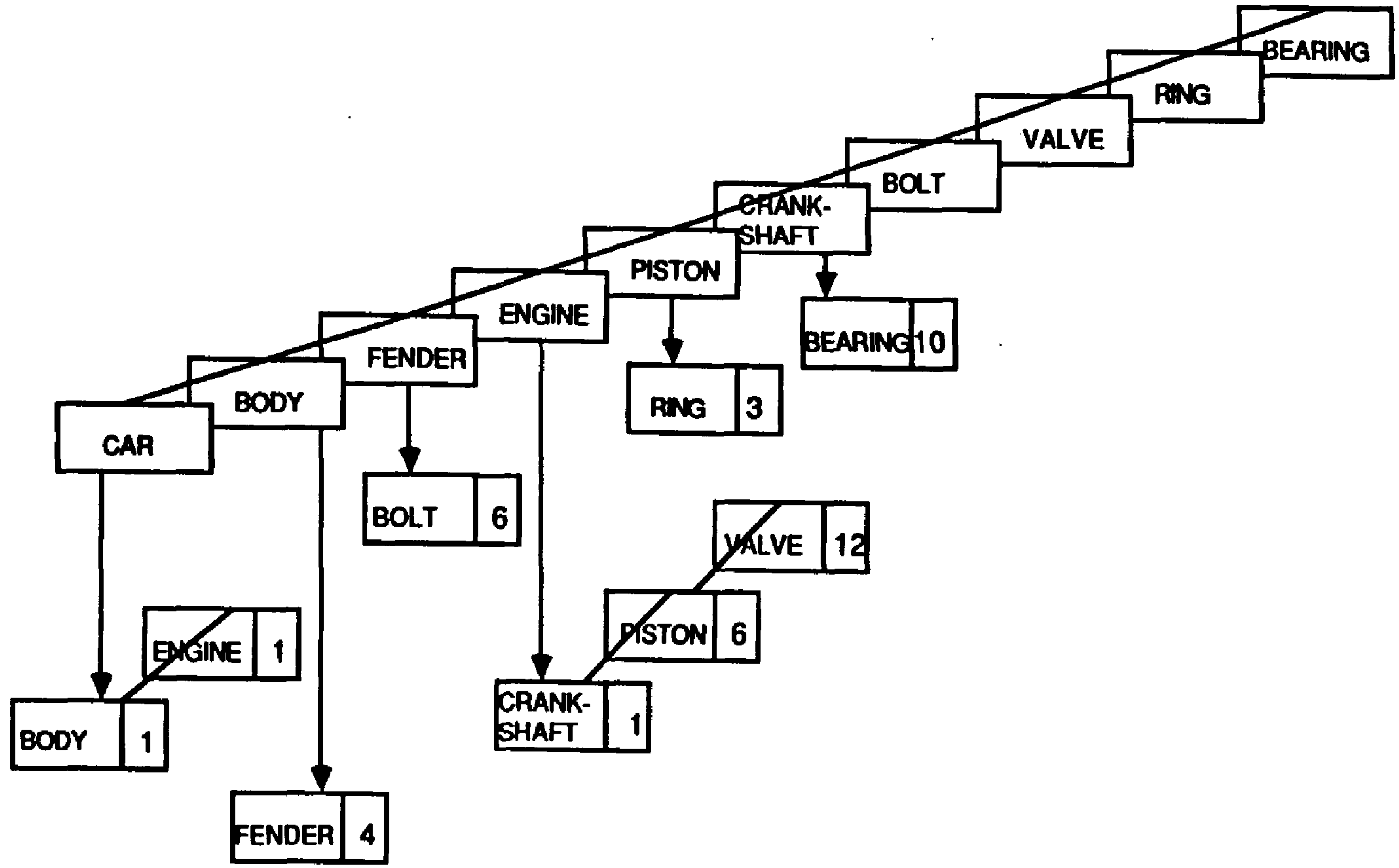


Figure 3.6 BOM data in hierarchical model

MAJOR	MINOR	QTY
Car	Body	1
Car	Engine	1
Body	Fender	4
Fender	Bolt	6
Engine	Crankshaft	1
Engine	Piston	6
Engine	Valve	12
Piston	Ring	3
Crankshaft	Bearing	10
etc.		

Figure 3.7 BOM schema and data in relational model

Based on my observations and analysis, an *object-oriented* model fulfills the requirements of CAD/CAM data. In an object-oriented data model, a semantic domain entity is expressed as a *concept* or *object* and is uniquely addressable. Objects are combined and related in many ways to create complex objects. Object-oriented models are characterized in detail in Chapter 4. I have defined an object-oriented data model, ODM, incorporating the modeling facilities prescribed above. ODM, described in Chapters 5 and 6, combines an object-oriented model with a network architecture. It provides a powerful, yet flexible framework for representing and manipulating CAD/CAM entities and relationships.

An object-oriented data model directly supports part-oriented BOM hierarchies. Two other CAD/CAM DBMS goals: customized representation for assemblies and parts, and incorporation of domain knowledge, are also aided by semantic modeling capabilities. In the part-oriented BOM hierarchy in Figure 3.8, nodes symbolize domain objects and links represent the *uses* or *contains* relationship. Because objects are contained in more than one part or assembly, BOM networks, a generalization of BOM hierarchies, allow multiple parents. For example, the *bolt* object in Figure 3.8 most likely is used in many other parts and assemblies, in addition to automobiles. Only the *contains* relationship is shown in Figure 3.8, however, other relationships can be merged with the BOM organization. In Chapter 5, I formalize the *contains* relationship and other relationships which are primitive in ODM. Chapter 6 describes domain-specific relationships, and how they are created and manipulated in the ODM prototype.

3.2 Dynamic schema capabilities

A DBMS schema is a static collection of data types defining allowable structures for data instances. The data types represent attributes, entities, and relationships of the application being modeled. Schema facilities are usually included as part of a comprehensive data dictionary package including a DDL (data definition language) for defining and manipulating schema specifications. In the following paragraphs I describe the role and capabilities of a dynamic schema, and introduce an object-oriented methodology as the underlying foundation of dynamic schema facilities in ODM.

Schema definitions are also referred to as *meta-data* because they define, control, and help locate data instances to which the schema pertains. Schemata,

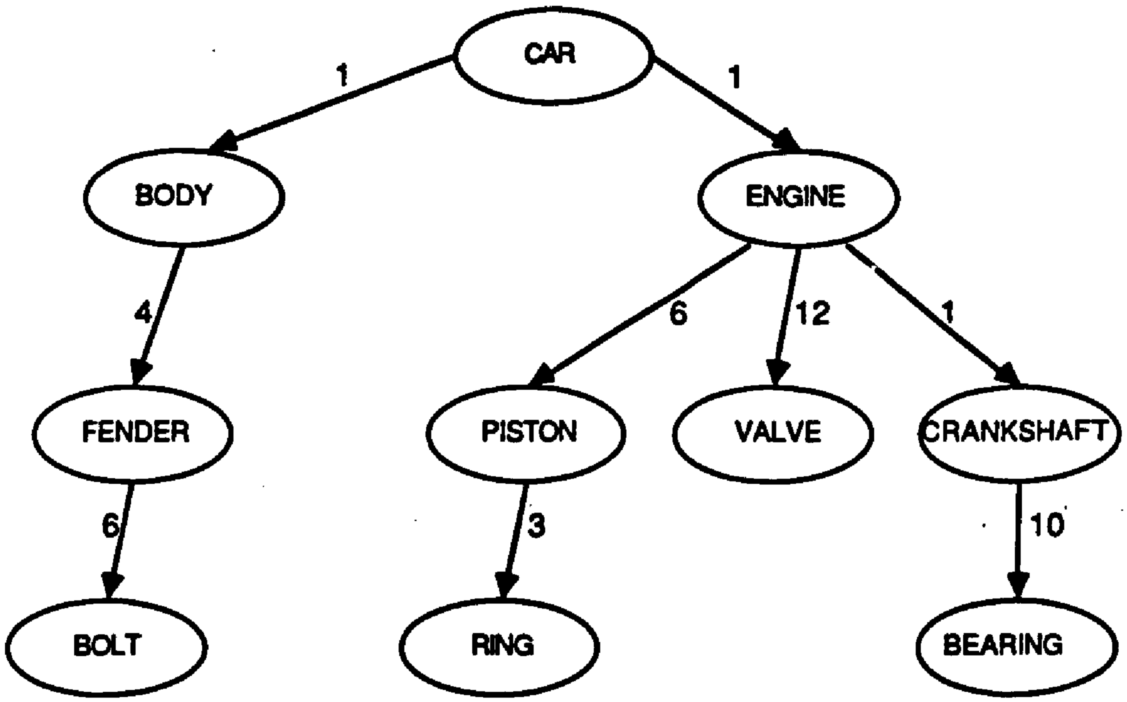


Figure 3.8 Part-oriented BOM hierarchy

therefore, are a management system for the structure of the data instances. Defining a schema and generating a data dictionary are expensive off-line tasks; therefore, most DBMS adhere to a static schema definition. Traditional static schemata define the organization of data by specifying data types and formats. Once the definitions are declared, they cannot be interactively modified and are expensive to change or extend. A static DBMS schema is analogous to the *data definition* section in a computer program. Declared data structures are fixed throughout the life of the program, and new data types cannot be defined dynamically within the program. To modify an existing data structure or add a new one requires recompilation of the program.

The enormous overhead for data base reconfiguration due to schema modifications has prohibited the practical use of a dynamic schema. Furthermore, in many applications the structure of data base entities can be completely defined during data base design phases. CAD/CAM data, however, is qualitatively different. As I discussed in section 2.3.3, CAD/CAM data differs from commercial data because the structure of CAD/CAM data grows with the design of the artifact. All products do not conform to the same fixed structure, therefore, static schema definitions are not sufficient. Design objects may have some properties in common, but features of assemblies and parts vary considerably. With dynamic schema facilities, schema specifications can be interleaved with the design of an object. These capabilities allow interactive additions, modifications, and deletions of schema structures during DBMS operation.

Active data dictionaries [McC82, Sch75] are being developed for browsing and viewing schema definitions. In some implementations of the relational data model [Eps77, Ora79] limited active and dynamic schema capabilities are available through user definable views, deletable relations, and addition of new attributes [Sto84]. Other efforts in these areas are reviewed in Chapter 7.

Dynamic schema capabilities are fundamental for achieving customized representations of assemblies and parts discussed in section 2.2.3. In a CAD/CAM data management environment with dynamic schemata, the distinction between schema and data begins to vanish. This effect reduces the artificial convention that information must be either *schema* or *data* [Mai84]. In an electronics design domain, a *resistor* is both schema and data. A resistor regarded as a *schema* item specifies properties which all resistors have in common. A resistor is a data *instance* when it is defined as a component part of a new PWA (Printed Wiring Assembly) being designed. In the past, DDLs (data definition languages) were only available to data base designers and administrators, and schema definition was decomposed from normal data base usage. With dynamic schemata and dictionaries, users can query the schema, in addition to modifying or adding new structures. New data structures such as entities, attributes, sets, and relationships are added in the same way as new data instances are added, modified, or deleted. To build dynamic schema, however, it is necessary to make data dictionaries more robust and user oriented. Figure 3.9 shows how the distribution of DBMS tasks would shift in an environment permitting interactive schema manipulation.

Dynamic schema facilities which I developed for the ODM prototype are detailed in Chapter 6. I adopted an object-oriented methodology by viewing the data dictionary as a management system for meta-data. In most commercial DBMS, a dedicated DDL (data definition language) is used for schema specification. Data base designers must specify physical characteristics and define navigation paths. In ODM, data base structures are *objects* which have knowledge about the behavior of meta-data. The system knows how to add a new attribute to a relation, generate a new entity structure, or establish a new relationship between two entity types in the same way that a DML (Data Manipulation Language) adds a new instance of a record, relation, or set. Because the system maintains information about schema and instance structures, it is possible to dynamically enforce consistency among existing structures and new enti-

	conventional DBMS with static schema	DBMS with dynamic schema
data base designer	-select logical data model -describe enterprise in terms of model primitives	-select logical data model -define model primitives -describe schema structures in terms of model primitives
data base administrator	-build data dictionary -build initial data base -maintain data dictionary	-describe enterprise in term of schema primitives -build initial dictionary -build initial data base
data base user	-create, access, query & update data instances	-create, access, query, & update schema description -create, access, query, & update data instances

Figure 3.9 Distribution of data management tasks

ties. Examples and discussion in Chapter 8 focus on the utility of dynamic schemata in an electronics design application at Hughes Corporation.

3.3 Semantic constraint maintenance

In general, constraints maintain a desired state in the real world. We constrain the temperature of our home freezers to below zero degrees Centigrade, so the freezer contents won't melt. In DBMS modeling, textual fields are constrained to some maximum length of characters; otherwise, the physical limitations of the DBMS and hardware systems might be exceeded. In the manufacturing domain, two holes drilled in a sheet metal part must be a separated by a minimum distance to prevent structural flaws. Much of the CAD/CAM data currently verified by human personnel falls into the category of restrictions or constraints on data entities, properties, and associated relationships.

In section 2.3.4, I discussed a recommendation to incorporate industry knowledge and standards into CAD/CAM data bases and I described scenarios using domain information. With the introduction of semantic modeling facilities in section 3.1, I now extend the use of semantic data by expressing constraints over entities, relationships, and properties.

Conventional DBMS constraints maintain the integrity and consistency of data instances. Validity constraints prevent polluted or contaminated data by restricting the values, data types, and format of data instances. Consistency constraints restrict the structure of data to prevent update anomalies. For example, if an employee data base contains information concerning an employee's children, and an employee is deleted from the data base; it is also necessary to delete the employee's children. Referential integrity addresses the maintenance of key attributes in records. If the value of a key attribute changes, all instances containing that attribute must also be updated.

Maintaining integrity and consistency has even greater importance in a highly robust CAD/CAM modeling environment. Users need to specify integrity constraints over a single data item or among many different data items. Constraints take the form of restrictions on data values, like a range of temperatures for heat treatment of a given material. They also express mathematical relationships between data values which must hold, such as the following mathematical equality between feed-rate, spindle speed, and feed for an NC operation: *"feed-rate = 2 (spindle-speed) (feed)"*. Structural relationships among features of physical objects also impose constraints. Relationships, such as *"part-A is-supported-by part-B"* and *"part-X is-inside part-Y"*, exemplify necessary design constraints.

A semantic constraint is a special type of semantic relationship between data base entities. A relationship such as *"surface-x is orthogonal to surface-y"* supplies data about the orientation of two surfaces. If this statement is represented in a data base, it furnishes information about the design environment. However, the constraint *"surface-x must be orthogonal to surface-y"* imposes a restriction on the values taking part in the relationship; or from a semantic viewpoint, imposes a restriction on the structure of the object being designed. *In design and manufacturing applications, constraints are relied on, not only to maintain the integrity of the data representing a part, but also to maintain the consistency of the design itself.* Therefore, semantic constraints express restrictions on the actual part, not simply on the data [Fen85].

Semantic constraint management requires sophisticated facilities for expressing and maintaining constraints. I first address the issue of semantic constraint specification. A general facility for expressing mathematical, procedural, and textual constraints is necessary. Many of the entities involved in a constraint are data instances themselves, therefore, referencing data instances from within a constraint specification must be supported. For example, in the constraint *''feed-rate = 2 (spindle-speed) (feed)''*, each item is a machining attribute of a sheet metal part. Therefore, this constraint is interpreted as *''feed-rate of part x = 2 (spindle-speed of part x) (feed of part x)''*. A constraint expressing a relationship between two different data objects such as *''surface x is-orthogonal-to surface y''* is defined as an instance of the relationship *orthogonal-to*, such that *surface x* and *surface y* are attributes of the relationship. When surface x and surface y are entered as data of the relationship orthogonal-to, the constraint system must verify that the constraint is fulfilled. Constraint enforcement, presented in the following paragraphs, considers another difficult issue of semantic constraint maintenance, namely, when and how to recognize the violation of constraints.

In contrast to conventional integrity and consistency constraints which are declared during data base design and schema definition, semantic integrity constraints may be entered at any time during data base processing. Three modes are possible for signaling constraint violation. *Incremental* consistency checking maintains only those data instances created after a new constraint is declared. Therefore, data entered before a new constraint is specified may be in violation of the new constraint. The second mode, *retroactive* checking, verifies all data instances when a new constraint is declared. This process inspects all data affected by a new constraint. The third mode combines retroactive checking with a switchable *enable/disable* setting to turn constraint checking on and off. In disabled mode, the overhead of keeping complete consistency during the design process is eliminated [Eas86]. Only when a design is to be *committed* does the designer want to verify its consistency.

Another consideration of semantic constraint management is the method of verifying constraint compliance or violation. In typical DBMS, datatype constraints are verified by computer operating systems; or by data dictionary facilities in the case of value, existence, and referential constraints. Many CAD/CAM constraints can also be verified by the DBMS or embedded programming language. For example, mathematical constraints which involve equality and inequality are generally verified by the DBMS implementation. Constraints with a well-defined universal meaning can be easily enforced. However, relationships which do not have a standard, quantitative definition and verification procedure require additional mechanisms. For a relationship like *orthogonal-to*, which represents a geometrical constraint, it is necessary for the user, data base designer, or DBA, to define the meaning of *orthogonal-to* in terms of data base entities and quantitative relationships. The definition and verification procedure is included as part of the constraint. In this example, if two surfaces are orthogonal, then the dihedral angle between the two surfaces is 90 or 270 degrees. To verify this constraint, an appropriate geometrical representation of surfaces and angles must be represented. If these entities and the relevant relationships are contained in the data base, and the definition of *orthogonal-to* is defined in these terms, then it is possible to verify this constraint.

A final issue for maintaining semantic constraints concerns the actions to be taken if a constraint is violated. Conventional DBMS simply reject unacceptable transactions. This approach can also be applied to incremental checking by rejecting new or modified data which does not fulfill associated constraints. However, with retroactive checking, inconsistent data may have already been committed. Most designers agree that in initial design phases, it is impossible to maintain complete semantic consistency [Fen85]. Therefore, designers would welcome a facility which simply recognizes inconsistencies, notifies users, and provides information about semantic violations. This method allows users to decide what action to take next. For instance, if the dimensions of a part are changed, the part may need re-engineering to adhere to structural requirements.

Another approach for maintaining consistency uses procedural constraints to automatically correct the data in error. Restating the constraint *"feed-rate = 2 (spindle-speed) (feed)"* as *"feed-rate <-- 2 (spindle-speed) (feed)"* helps automate constraint satisfaction. In this example, if the value of feed-rate doesn't adhere to the constraint equation, then the system is instructed to compute the value using the given equation.

A third option trys to undo a transaction which caused a constraint to be violated. Automatic backtracking requires detailed histories of data base transactions and complex dependency representations. Researchers are currently unclear of the implications of automatic backtracking on the design process. These considerations are being discussed in the domain of CAD/CAM DBMS and other design environment such as architecture and electronics design. Related topics such as dependency-directed backtracking, relaxation techniques, and constraint propagation [Bor79, Ste80, Bar81] are also critical elements of general purpose constraint maintenance systems.

A semantic constraint facility combined with object-oriented semantic modeling capabilities affords powerful tools for achieving design consistency. Tasks traditionally performed off-line by manual analysis of engineering drawings can now be interleaved with the design process thereby streamlining product development. Topics discussed in the previous paragraphs forms the basis of the semantic constraint facility in the ODM prototype. These facilities are detailed in Chapter 6. Examples in Chapter 8 demonstrate the use of semantic constraints to replace rules in a CAM expert system.

3.4 Heterogeneous data types

Management of heterogeneous data is necessary for both conceptual centralization of CAD/CAM data and incorporating application knowledge. In manufacturing environments, different types of data include graphical, geometrical, engineering, manufacturing, and administrative data. Facilities for querying all aspects of a manufactured product depend on modeling these heterogeneous data types. Automated manufacturing and engineering operations have resorted to specialized local data bases in order to maintain the different categories of data relevant to their needs. Some data is stored in electronic data files; however, much of it resides in hard-copy reports produced manually, or is generated as needed. Below I describe these different data types and how they are an integral part of a manufacturing domain.

Graphical data, generated during drafting and engineering design, is mainly used for two dimensional displays and includes entities such as lines, points, arcs, splines, and curves. Specialized CAD and drafting systems represent graphical data using entities most suitable for displaying graphical images. The format of graphical data is determined by the two dimensional display system and therefore obeys formats and constraints imposed by the corresponding system, such as output devices and coordinate systems. In this representational approach, semantic content implied by the graphical data is lost. For example, by viewing a display, it may be obvious that one surface is orthogonal to another; however, it is impossible to derive this fact by querying the graphical data file. Many research projects developing graphics standards, such as IGES [Ini83], GKS [Gra85], and Core [New78], are focusing on graphical data representation. Little work, however, has been directed toward integrating graphical representations with other CAD/CAM data.

Geometrical data represents three dimensional topological features such as faces, edges, and surfaces. Geometric data is used to construct a mathematical model of a part and therefore relies on mathematical representations. Currently, most geometrical data is managed by solid modeling systems described in section 2.2. Experts in solid modeling are starting to recognize the need for structured organization of geometric data, and some CAD/CAM DBMS research efforts [Ulf82b, Woo83, Ulf82a] are building their DBMS around a geometric representation.

Engineering data is generated after part definition and prior to manufacturing. Computations such as structural and thermodynamic analyses, simulation of motion, and material flow, produce and consume engineering data. Engineering data is mathematical in nature and consists of matrices, vectors, and algebraic equations and formulae. Until recently, most data has been associated with a specific engineering analysis. Each analysis program has unique requirements for data input, therefore, a great deal of overhead results from data preprocessing. Only now, as automation enters the design, engineering, and manufacturing phases has it become imperative to maintain engineering data in an integrated centralized data base.

Manufacturing data is least integrated into CAD/CAM DBMS. Because manufacturing has been primarily a manual operation, there was little motivation to store the required data electronically or consider automated retrieval and update. The advent of NC machining was a driving force toward electronic management of manufacturing data. Progressive manufacturing firms employing DBMS for manufacturing data have usually done so in conjunction with CAD systems for part definition and drafting. Most manufacturing data takes the form of a procedural plan or sequence of actions. Manufacturing phases requiring procedural data are process planning, tool design, fabrication, assembly, and testing. Data for process planning tasks include machine setup specifications and instructions for metal forming operations such as casting, cutting, and forging. Tool design requires knowledge of material types and part specifications. Machining processes use procedural data for generating NC programs and cutter path optimizations. At Lockheed, the Production Inspection Record (PIR) is a document generated and maintained manually by process planners at Lockheed. It consists of a sequence of detailed assembly notes for joining parts and inspecting them. The information on a PIR includes where and when to fasten pieces of an assembly; when to inspect the assemblage, when to heat treat, and what tools to prepare.

Textual data is found in all phases of CAD/CAM, from design through marketing, however, the heaviest use of textual data is for administrative functions. Managerial applications for manufacturing include production scheduling, cost estimations, and quality control. Other administrative applications such as sales, marketing, inventory control, and purchasing also require data management systems for effective processing. Of the heterogeneous data types discussed, administrative data is most commonly maintained by a data management system. The data consists of alphabetic or numeric types, and data access is based on pre-defined data paths. Because CAD/CAM administrative data closely resembles data in commercial domains, generalized DBMS are usually sufficient for administrative and managerial report generation, queries, and updates.

To further support integration of CAD/CAM applications, it is desirable to reference different sources of data from within a DBMS. *Directory* data is a meta data type for referencing other CAD/CAM data. This type of data allows symbolic pointers to auxiliary data bases or computer installations. For example, if outdated versions of a geometric model have been archived, it should be possible to retrieve the relevant information to manually or automatically access the off-line storage. Automatic access requires the DBMS to initiate a process for loading or unarchiving the desired data file. Another use of directory data allows access across different DBMS. If proper procedures are specified, retrieving data from another DBMS can also be executed as an external process. Recent work on the relational DBMS, Ingres, has progressed in a similar direction. Their approach supports DML commands as a data type in the DBMS. This proposal allows Quel commands as attribute or column values which can be evaluated and executed during DBMS processing [Sto84].

In the previous paragraphs I characterized different types of data which are required by an integrated CAD/CAM data base system. To date there doesn't exist a data management system which can efficiently accommodate all the data. Current DBMS do not have adequate facilities to maintain heterogeneous data. Existing commercial systems have evolved from record and file based systems to hierarchical and network set/owner models, and most recently to flat relational models. Given this heritage, the predominate data structure is still a strictly typed, textual record. As a result, generalized DBMS are best suited for applications with homogeneous, textual data. One goal of this research is to develop methods for maintaining the heterogeneous data presented above. Chapter 6 describes facilities in the ODM prototype supporting complex and heterogeneous data.

In this chapter, I have concentrated on four specific areas of CAD/CAM DBMS functionality: object-oriented semantic modeling, dynamic schemata, semantic constraint management, and heterogeneous data types. The capabilities described above serve as a functional specification for a new object data model, ODM, and a prototype implementation. In the next chapter, I lay the theoretical groundwork for ODM by focusing on computational object-oriented models.

CHAPTER 4

OBJECT-ORIENTED MODELS

Object-oriented models have appeared under many different guises. They have prominently evolved in the areas of *programming languages, data base management*, and *knowledge representation*. Only in the past few years have researchers in these areas recognized the similarities and distinguished the differences among object-oriented paradigms. In previous chapters, I discussed the motivation for adopting an object-oriented theory for the management of CAD/CAM data. In this chapter I present the evolution of object-oriented models in each of these computer science disciplines including a discussion of unique features and limitations.

4.1 Object-oriented programming languages

Object-oriented languages are characterized by their method for structuring and processing data. *Class* data structures are the main data type, and hierarchies of classes and subclasses are constructed using language primitives. Classes are instantiated to produce specific *instances* of class objects. A goal of object-oriented languages, derived from the study of data abstraction, is to manipulate class objects as self-contained entities or objects. Objects interact with each other through their global instance name, providing a clean interface between objects of similar or different classes. A class is defined by its own *attribute variables* and also inherits attribute variables from its superclasses. Likewise, subclasses inherit *procedures* for manipulating instances. The internal structure of objects, and methods for processing objects are hidden within an object's definition, realizing the concept of abstract data types.

Simula, developed in 1967 as an extension of Algol 60, is one of the pioneer object-oriented languages. Facilities for maintaining class structures and class hierarchies provide basic extensions approximating an object-oriented language. Two of Simula's builtin system classes are the *SIMSET* and *SIMULATION* classes. SIMSET provides an implementation of sets as doubly-linked lists, and the SIMULATION class defines process control and coroutining [Bir73]. Even today, object-oriented programming languages are frequently equated with simulation languages and facilities.

The successor to Simula and the purest object-oriented language is Smalltalk [God82]. Unlike its predecessor which includes traditional data types such as integers, reals, strings, and arrays; the only data types which Smalltalk supports are *classes* and *instances*. Smalltalk is strictly object-oriented because all data is accessed by unique object names. The internals of an object, namely, its properties and processing routines, called *methods*, are hidden from other objects. Another way in which Smalltalk differs from Simula is its procedure or method invocation. In Smalltalk, a *message template* is associated with each object method. Instead of explicitly calling a method name to invoke a processing routine; a message conforming to a message template is sent to an object. Receipt of a message triggers the retrieval and execution of a corresponding method by the receiving object. All computations are performed by message transmissions, therefore, the *message-passing* paradigm has come to be closely associated with object-oriented languages. Most object-oriented languages such as Flavors [Obj84], Ross [McA85], and Strobe [Smi84] employ a message-passing form of procedure invocation. Object-oriented languages entail other features such as overloading, late binding, and interactive interfaces [Zan86a]. These capabilities, however, further describe the functionality of an object-oriented language; they are not requisite definitional properties such as object identity, data abstraction, property and method inheritance, and message-passing.

Object-oriented paradigms for programming languages are being extended to the specialized fields of simulation, logic programming, and operating systems. Object-oriented languages are particularly successful as simulation languages because of the natural correspondence between real world objects and program objects. Object-oriented simulation languages usually employ a *clock* object for time and event management. A hierarchy or network of class objects represents a taxonomy for describing simulation objects and their specializations, and real world processes are modeled as methods of simulation objects. Simulations written in object-oriented languages have shown to be easier to design and code, easier to modify, and easier for a domain analyst to understand and critique [Kla82]. Object-oriented programming in Prolog has been proposed with primitives to support objects, methods, inheritance networks, and message-passing [Zan86b]. Cola [Sno83], an object-oriented command language for a capability-based operating system, is reviewed in Chapter 6.

4.2 Entity-based data management

In the field of data base management, *object-oriented* is usually synonymous with *entity-oriented* and is best described in contrast with the relational model. In relational models, data organization is based on the mathematical definition of a relation: the Cartesian product of two or more domains. A data base relation modeling a real world situation contains a subset of the cross-product of domain values of relevant attributes. Each element of the subset corresponds to a relational *tuple*. Data items or tuples are accessed primarily by a relation name and secondarily by values for attributes within the relation. Tuples in a single relation can only be distinguished by values of the composite attributes. To retrieve a complete description of an entity may require accessing many relations and selecting only the tuples whose values correspond to some *key* value for the entity in question.

An entity-oriented model, however, associates a unique identifier with a real world entity. Data retrieval is based primarily on object identity. An entity, along with its description, attributes, and values, is accessed directly by its entity name. Once an object is accessed, attribute values and relational components can be selected.

The Entity-Relationship (E-R) model was originally developed as a data base design tool to model reality in terms of entities and relationships among entities [Che76]. Although the original goal of the E-R model was to conceptually unify network, hierarchical, and relational models; the E-R model has gained its own recognition and is the foundation for many object-oriented data base models. The development of the E-R model combined with the introduction of Smalltalk, has contributed to object-oriented data base systems [Cop84], an object-oriented design for distributed data bases [Web83], and an object-oriented methodology for DBMS implementations [DeW81].

One important feature of object-oriented models, which the relational model lacks, is the concept of complex objects. In a pure relational model, attributes are single valued and tuples are accessed by the values of attributes, not by a tuple identifier. The difficulty of representing hierarchical structures by flat relational tables, limits the semantic power of the model. Recent attempts at extending the relational model to accommodate complex entities include RM/T [Cod79], an extension focusing on aggregation. Extensions to two of the oldest relational data base systems, Ingres [Sto76] and System-R [Ast80], also focus on improved semantic expressiveness. Additions to these systems include *tuple-ids* and *repeating groups* [Sto84, Lor82, Plo84]. With these extensions, the relational model is migrating toward an object-oriented paradigm where tuple-ids represent entity identifiers, and repeating groups simulate hierarchical aspects of class/subclass structures. These and other related efforts are discussed in Chapter 7.

4.3 Schema-based knowledge representation

The study of knowledge representation has gained significance with the emergence of artificial intelligence (AI) and expert systems research. A knowledge representation system attempts to encode domain or application knowledge, supplying data and context for AI applications such as expert systems, natural language understanding, vision, and robotics. Most AI applications require some *common sense* knowledge which we as humans accumulate through experience. For AI systems to achieve success, this common sense knowledge must be available for computational processing. In addition to the data maintained by conventional data base systems, knowledge base management systems must also store and maintain knowledge about processes, goals, plans, causality, time, and actions.

Many knowledge representation paradigms have been developed and used for different AI applications. Below I have classified three schema-based knowledge representation methodologies: *semantic networks* [Fin79, Fah79, Sow84], *frame representation systems* [Min74, Bar81], and *object-oriented models* [Bra85]. I derived this characterization from my observation that each representation entails a fixed framework into which relevant knowledge is stored. Semantics are defined for components of the framework, or schema. These semantics prescribe how knowledge is stored, and describe information contained within the schema. In a semantic network representation, the components are nodes and links, whose meaning must be defined. For frame systems, the semantics of frames, slots, fillers, and the relationship between frames must be denoted. Object-oriented models require clear specification for the semantics of classes, class hierarchies, instances, and attributes. These definitions must be unambiguously stated so that all knowledge is entered in a consistent fashion, and the correspondence between the computational model and reality is as close as possible. Knowledge representation systems not only store facts but also include inferencing mechanisms for making deductions based on facts and axioms. Inferencing capabilities for schema-based models require control mechanisms built to operate on the particular framework.

A modeling system which is not schema-based is a logic representation system. For example, a logic representation derived from predicate logic does not require a predefined schema for storing information. All data is represented as predicate formulae. Based on the axiomatization of predicate calculus we clearly understand the meaning of facts such as: *father(jane, ted)* or *color(sky,* *blue).* Furthermore, any standard theorem proving system for predicate logic can be applied to such a data base of facts and rules.

4.3.1 Semantic networks

In its simplest manifestation, a semantic network consists of a data structure of *nodes* and *links*. Nodes represent concepts, and links between nodes represent associations between the concepts. Specialized inference procedures operate on semantic networks to deduce new facts and relationships. The precise meaning of a node, and the semantics attached to a link are decisions left to the system designer. If links represent the relationship *is-a*, then a taxonomic hierarchy, like the one illustrated in Figure 4.1, is generated.

When links represent roles of a case grammar, a concept such as *gives* is associated with role fillers, through role links [Sow84, Mil76]. In Figure 4.2, role links such as *agent, object*, and *recipient* are connected to nodes representing the respective role fillers: *student, homework*, and *teacher*. The resulting network structure represents the assertion: *The student gives homework to the teacher*.

Semantic nets were introduced as an intuitive notion of associations [Qui68]. Because the idea was easy to grasp, researchers quickly adopted the use of semantic networks for knowledge representation. During the initial growth and development of semantic networks, people were experimenting with different meanings for the network formalisms. Over time, the semantics of a node/link data structure represented many different interpretations. To deduce valid inferences from a semantic network, consistent semantics must be attached to all nodes and links. Unfortunately, the intuitiveness of these network struc-

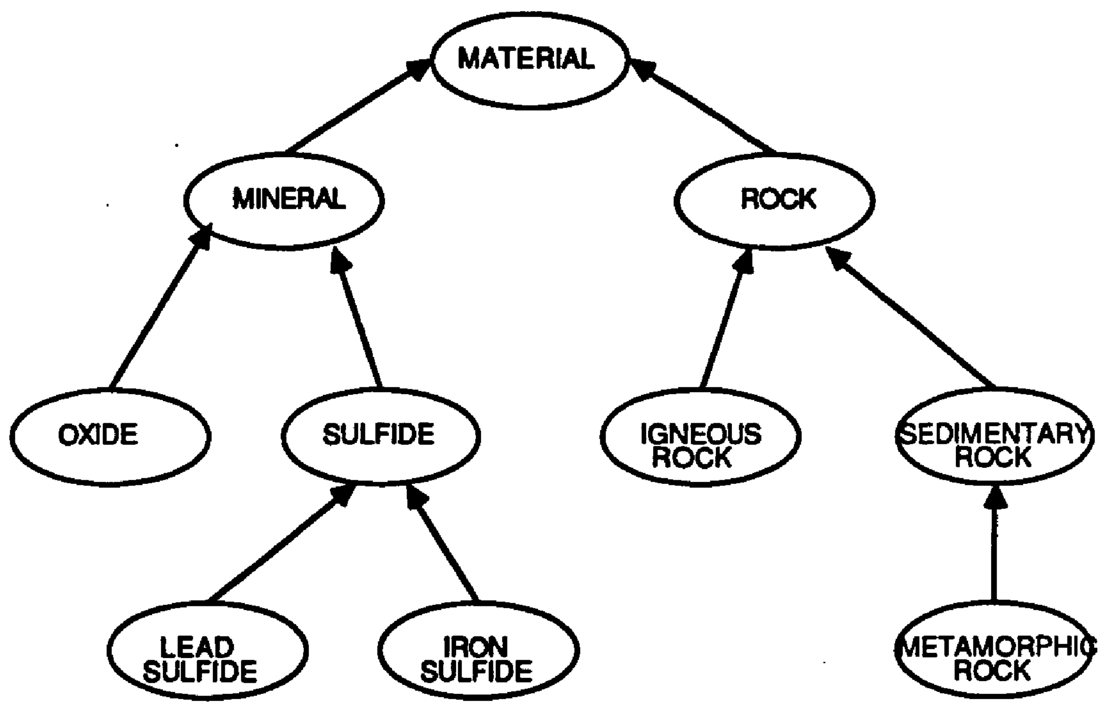


Figure 4.1 Semantic network representing a taxonomy

tures tends to obscure inconsistencies which many semantic network systems are guilty of [Bra83]. Researchers are attempting to sort out the different meanings and uses of semantic network representations [Ste78, Bra78]. Recent knowledge representation efforts are addressing the *meaning* of node/link associations, and formally stating the semantics they intend.

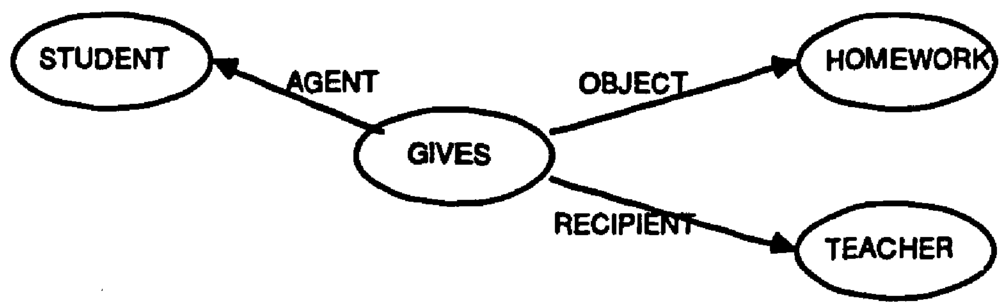


Figure 4.2 Semantic network representing an assertion

4.3.2 Frame representations

A knowledge base organized as a frame representation model views knowledge as modular decomposable chunks or *frames* [Min74, Ste78]. Dividing a knowledge base into frames is common in applications like computer vision and natural language understanding. A frame usually represents a prototypical organization of a concept. Slots or frame variables further describe a generic concept or object. When a frame is instantiated, its slots are filled with pointers to other frames. Frames are combined to form situations, and procedural knowledge is attached to slots for inferencing. Frames are organized as type or category hierarchies, similar to class hierarchies in object-oriented programming languages. Like semantic networks, many variations of frame-based languages have been developed. An example of a simple dining room frame is illustrated in Figure 4.3.

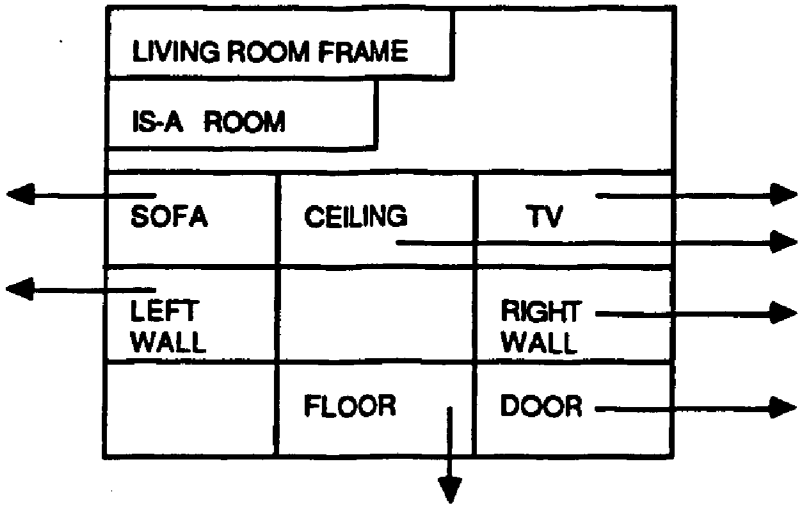


Figure 4.3 A frame representing a stereotypical living room

4.3.3 Object-oriented knowledge representation

In theory, the difference between a semantic network and a frame system is clear; however, operational systems built upon these formalisms cannot always be strictly identified as one type or another. If a system of frames is organized as a network, is the resulting model a semantic network or a frame system? Similarly, if nodes in a semantic network have structure with slots and fillers, do we have a frame system?

None of these categorizations are mutually exclusive, and systems depicting object-oriented models also suffer from this identity problem. Most object-oriented systems also have functional qualities of frames and semantic networks [Bra85]. However, based on the preceding discussion of object-oriented programming languages and entity-oriented data management systems, I can now present some guidelines for characterizing object-oriented knowledge representation systems. Often, the semantics attached to system primitives helps classify the representation model.

The primary and underlying organization of entities in an object-oriented model is a *taxonomic* structure. This view is consistent with the organization of objects in object-oriented programming languages: subclasses are specializations of classes, and a subclass is instantiated to represent a specific instance. If we denote objects as nodes and the *is-a* relationship as links, we can generate a taxonomy network for a particular concept. The relationships between objects in the resulting classification network represent a form of abstraction called *generalization*.

A taxonomic classification proves to be a key component of an inferencing method referred to as *inheritance*. Inheritance allows properties of objects to be distributed across a generalization hierarchy. Properties are explicitly attached to the most general concept exhibiting the property, and specializations of the concept are said to *inherit* the property. It is argued that this mechanism contributes to conceptual clarity and physical storage economy because shared properties are not replicated wherever they apply.

Many object-oriented models are guilty of the same flaws exhibited by semantic networks. The semantics associated with the intuitive notion of inheritance are not rigorously defined. This ambiguity results in obscure notation and invalid inferences. I discuss the implications of this vagueness in the next section. Ambiguity surrounding both the *is-a* relationship and property inheritance must be resolved before we can make full use of their power as valid inferencing techniques [Bra83]. Formal definitions of generalization and other abstraction mechanisms in ODM are detailed in Chapter 5.

Emphasis on concept definition and description also distinguish object-oriented systems from other knowledge representation techniques. The primary focus is on concepts or objects and their properties. Relationships between objects, other than generalization, are not explicitly supported. This distinction was also addressed in the previous discussion comparing entity-based and relational data base models. Nevertheless, it is necessary to represent relationships in an object-oriented system. By viewing the notion of *giving* as a concept, the properties of *giving* correspond to its roles; namely, *agent*, *object*, and *recipient*. Although these concepts differ from *object* concepts, object-oriented models support relationships through this approach.

During this discussion of schema-based representation techniques, it is important to note that knowledge representation systems maintain complex and unstructured knowledge compared to the data managed by a DBMS. Imagine representing all the knowledge (not just the character strings) of a murder mystery in a knowledge base. Posing a query such as *"Who killed the butler?"* requires much more semantic information and inferencing capabilities than is necessary for a DBMS query such as *"What are all the projects in Department 623?"* Nevertheless, data base management systems can profit enormously from semantic representations and inferencing techniques. Indeed, we would like computers to understand the meaning of a book by entering its text. Similarly, by entering an engineering drawing, we wish a computer system could understand the components and processes required to manufacture a metal part. With such an ambitious achievement, we could then present queries such as *"Do the holes in this bracket require reaming?"* or *"What cutter speed should be used for this gasket?"*. ODM is a step toward this goal of semantic data models by merging knowledge representation and data base management technology.

4.4 Deficiencies of object-oriented models

The proliferation of object-oriented languages has resulted in many variations of the object-oriented paradigm, each defining different terminology and meaning [Zan86a]. In part, object-oriented models have gained popularity through their intuitive character. The notions of *objects, classes, methods,* and *message-passing* are simple concepts to grasp yet provide substantial modeling power. Unfortunately, the multitude of variations and intuitiveness of the concepts often deter the development of formal definitions. As I discussed in the previous section, similar phenomena occurred during the early development of semantic networks. Frequently, the semantics of knowledge representation languages is defined by their implementation -- not the best way to develop consistent and long-lasting theories.

Object-oriented data structures are frequently described in terms of *classes, subclasses, instances, properties,* and *property values.* Much of the terminology has acquired an informal connotation, therefore, these systems rarely offer a formal definition of their terms. As a result, inconsistencies are difficult to detect. In this section I identify some of the issues which must be considered when describing object-oriented representations.

Some ambiguity revolves around the notion of a *class*. Does a class (or class object) represent the *set* of all objects fulfilling some qualification, or does a class object refer to a *prototypical entity* with a particular description? For example, if we define a class of *cats*, does the class refer to the set of all cats or a single generic cat? If the class of cats refers to the set of all cats, then properties describing the class will modify the set. With a *set* interpretation, it is sensible to ask about the cardinality of the class or set of cats. Querying about the color or weight of the class only makes sense if a class refers to a generic representative of the class. If a class object refers to a set, then how and where is the description of the prototype retained? Conversely, if class represents a prototype with a schematic description, how is the set or collection of instances referenced? My research has focused on developing a representation encompassing both interpretations.

These issues are compounded when the *is-a* connective is introduced to express relationships between classes, subclasses, and instances [Bra83]. It is essential that object-oriented representations make a distinction between the statements *"<subclass> is-a <class>"* and *"<instance> is-a <class>"*. If *class* and *subclass* are assumed to represent sets then *"tabbies are cats"* is an instance of the first statement. This fact expresses a subset relationship interpreted as *"the set of all tabbies is a subset of the set of all cats"*. The second statement, such as *"Isabella is a cat"*, represents the *member* relationship between elements and sets. The corresponding statement, *"Isabella is a member of the set of cats"* explicitly reflects this membership.

Under the assumption that class is a prototype object, the statement "*<subclass> is-a <class>*" is interpreted as follows: The description of a prototypical subclass object is subsumed by the description of a prototypical class object, as in "*a tabby is a cat*". Different semantics are intended, however, if *is-a* relates an instance and class under the same prototype interpretation of class. "*Isabella is a cat*" indicates that Isabella is an instance of a prototypical cat where the schematic description has been replaced by real values. In each of these cases, the meaning of an *is-a* relationship depends on the type of its arguments, namely instance or subclass. In ODM, I have eliminated this dependence by defining typed relationships to reflect the correct intended semantics.

So far, discussion has focused on the semantics of class, subclass, and instance. Another vital component of object-oriented representations is the *property description* of an object. Most often this description takes the form of attribute/value pairs and is the basis of a technique called *property inheritance*. In its abstract form, inheritance refers to a method of implicitly distributing attribute/value pairs from classes to subclasses and instances. For example, if cats have whiskers and tabbies are cats, then it follows that tabbies have whiskers. Furthermore, if Isabella is a tabby, she also has whiskers. The intuitive motivation for this technique relates to the *subsumption* of subclass objects by class objects and *instantiation* of class objects to produce instances. The class of cats subsumes the class of tabbies; therefore, we can infer that any properties of cats, like having whiskers, also apply to tabbies. Secondly, because Isabella is an instantiation of the class of tabbies, she assumes the properties of tabbies which again are inferred from the class of cats. Unfortunately, most object-oriented systems do not define an underlying principle for prescribing the distribution of property descriptions.

At least three variations of inheritance must be addressed [Ste78]. In the simplest case, the value of an attribute is constant over all subclasses and instances, and may be associated with the class object. This aspect of inheritance also applies for predicates defined as properties. For instance, the predicates *has-whiskers* and *has-claws* are true for the class of cats, therefore, they hold for tabbies and Isabella.

A second case arises if all instances are described by the same property but the value of the property is not constant across instances. In this situation it may be desirable to specify a set of possible values or enforce other conditions on the value, such as *"color is white or blue or brown"* or *"weight is less than 5000"*. Here, class to subclass inheritance implicitly passes a description to a subclass. However, class to instance inheritance indicates that the property is instantiated with a value fulfilling the description. Although it is true that the color of my car is *"white or black or brown"* and its weight is *"less than 5000"*, specific values are intended for the *color* and *weight* properties of a specific instance of the class of cars, namely, *my car*. The semantics of this variant depends on the types of objects being related. Inheritance from class to subclass differs from class to instance inheritance.

A more complicated situation must be faced when a subclass description is more restrictive that a class description. Although the weight of all Chevrolet cars is less than 5000, the weight of Chevette models is less than 3000. This case necessitates a specification such that the set of possible values for a subclass property is a subset of allowable values for the class property. Subclasses are *specializations* of classes, therefore, the values of a property may be more specialized or restrictive than the same property of the superclass. In the following chapter I present my alternatives to the informal inheritance techniques reviewed above.

CHAPTER 5

ODM: AN EXTENDED OBJECT-ORIENTED DATA MODEL

In Chapter 2, I discussed motivating DBMS goals relating specifically to CAD/CAM applications. Based on these objectives, I formulated functional specifications for CAD/CAM data management. As a result of this process, I determined that an object-oriented representation model best fulfills the proposed requirements. Next, I analyzed the strengths and weaknesses of different aspects of object-oriented representations. I observed that three computer science disciplines: programming languages, data management, and knowledge representation, have directly influenced the development of object-oriented models.

In this chapter, I first outline five representational goals concretized by my review of object-oriented models. The next section details the achievement of these goals by presenting the theoretical specification of a new object-oriented data model, ODM. Section 5.2 also describes how the extended capabilities of ODM are superior to those of current object-oriented models.

5.1 ODM goals

The following five capabilities were driving forces in the development of ODM. Discussion of each capability identifies its origin and refers to the corresponding ODM feature supporting the capability.

- represent complex hierarchical data structures
- model semantic objects and relationships
- include inferencing capabilities
- provide extensional semantics
- specify well-defined semantics for model primitives

Basic theories relating to *complex objects* and *hierarchical data types* were exported from the fields of programming languages and knowledge representation. Both areas have developed methodologies for complex class/instance structures, generalization hierarchies, and emphasised the importance of information hiding derived from the study of abstract data types. Complex object description is the subject of section 5.2.1 discussing concept representation.

Providing rich primitives for representing *semantic objects and relationships* has been explored primarily in knowledge representation work. Recent DBMS efforts at specifying semantic data models for improved expressibility have produced mathematical models, irreducible data models, semantic hierarchy models and direct extensions of the classical data models [Nil80, Tsi82, Bor80]. Section 5.2.2, concept relationships, presents techniques for extending the modeling power in ODM.

Research on *inferencing* mechanisms also stems from work in knowledge representation. In most modeling applications, it is impossible to explicitly identify and represent every piece of necessary information. Deduction systems like logic provide a formalism, ie. axioms, to declaratively extend the knowledge of a system by applying axioms to known facts. Production rules [Bar81, Nil80] are a formalism which procedurally infer new knowledge from existing data. The types of inferences which are profitable in CAD/CAM applications and have been included in ODM, are described in section 5.2.3 addressing concept inferences.

Extensional semantics is a feature we take for granted in DBMS. Extensional semantics refers to techniques for managing data instances; namely, the instantiation of schema descriptions by real world objects. Requesting all tuples in a relation or all the member records of a set owner record is a routine DBMS operation. In programming languages, however, no such concept is built into the languages. For example, if a new instance of a Simula class description is created, there are no automatic mechanisms to keep track of a pointer to the new object. The programmer is expected to maintain instances of data structures explicitly within the code. In contrast, in a relational DBMS, if a new instance of a relation is added, that tuple is remembered and becomes part of the extension of the relation. In knowledge representation, most efforts have been directed at understanding and specifying schemata for real world situations including notions of time, belief, actions, and state transitions. A taxonomic hierarchy of material compounds, such as Figure 4.1 illustrates, doesn't assert that any samples of these compounds exist, it merely provides a classification scheme in which to store potential instances of the class. Extensional semantics in today's knowledge representation systems use only ad hoc techniques for instance representations. In ODM, I have formalized extensional semantics through the use of four concept primitives detailed in section 5.2.1.

Specifying *well-defined object semantics*, the purpose of the entire next section, has been extensively addressed in programming languages and DBMS. Formal semantics describing an object-oriented model do not provide any explicit capabilities for modeling. Instead, well-defined semantics justifies the integrity of the modeling environment. Any inferences made by the system can be proven by expressing the relevant facts, rules, and conclusions in the formal definition language. Knowledge representation work has been exploring many diverse means of capturing and storing knowledge. Until now research has been concentrated on functionality at the expense of rigorous definitions. Researchers are beginning to adopt a more formal perspective on the semantic issues entailed by knowledge representation.

So far, I have discussed features of *generic* object-oriented models. In the next section, I compare aspects of ODM with facilities of particular systems. In Chapter 7, I review specific object-oriented implementations and describe how they differ from ODM.

5.2 ODM definition

The research presented in the rest of this chapter defines the object-oriented data model, ODM, by specifying formal semantics for its representation language. This work provides a theoretical framework for the ODM prototype presented in Chapter 6. The formalization discussed here is based on set theory and predicate logic and serves many purposes. First, it eliminates ambiguity inherent in intuitive definitions. A second benefit is afforded by the soundness of logical inferences derived from the model's axioms and theorems. Finally, the behavior of the model does not rely on a computer implementation. The result is a formal specification which can be used as a theoretical modeling tool or operationalized by a computer software system.

The research presented in the rest of this chapter also investigates the integration of *generalization* and *aggregation* principles in ODM. Although object-oriented systems are typically represented as generalization networks; my analysis of CAD/CAM data strongly recommends aggregation hierarchies as a compatible extension. My results have shown that integrating generalization and aggregation in ODM promotes a unique mix of logical inferences.

The semantic formalization of the model includes definitions of object primitives, axioms, and theorems. Section 5.2.1 introduces the object primitives from an intuitive standpoint to provide some conceptual correspondence between this representation system and other object-oriented representation languages. Following this informal discussion, I define four primitive components of the model in terms of set theory. The axioms described in section 5.2.2 are based on predicate logic and help support generalization and aggregation abstractions. Theorems, derived from the axioms, generate inferences in the modeling domain. These theorems are presented with examples in section 5.2.3.

5.2.1 Concept representation

Before presenting formal definitions of ODM, I discuss components of the model intuitively, through examples and analogies to other representation systems.

Four primitive components of ODM are *intensions*, *instances*, *descriptions*, and *extensions*. All concepts and objects of the modeling domain are composed of these four components.[1]

An *intension* corresponds to a prototype concept. It refers to a generic concept, such as an automobile or giraffe, not a specific real world instance. An intension includes properties and value sets describing the prototype. Properties describing an automobile might include *color, weight,* and *wheelbase*; the intension for a giraffe might contain the properties *color, height,* and *habitat*. Value sets associated with each property represent the set of allowable values of the property. For example, "*{(weight {x/x < 5000}),(color {red, blue, white}), (wheelbase {x/x < 150})}*" might represent the intension of an automobile, where, *color, weight,* and *wheelbase* are property names. The value set denoted by "*{x/x < 5000}*" indicates that the value of the *weight* property for an automobile must be less than 5000. Similarly, the value of *color* must be either red, blue, or white. In relational data base terminology, an intension corresponds to the schema of the relation; properties correspond to schema attributes; and value sets are similar to attribute domains. Properties of an intension are descriptions, not complete definitions. If two intensions have the same properties, they do not necessarily represent the same prototype. For example, giraffes and cheetahs both have *color, height* and *habitat* properties but are very different objects.

Intensions are not exclusive descriptions. The "*animal*" intension subsumes the "*giraffe*" intension, that is, animal properties can also describe giraffes but not vice versa. *House* and *vehicle* also are non-exclusive intensions; a *motor home*, for instance, may be described by both intensions. The model does not place any restrictions on what constitutes an intension. If an object can

[1]I use the terms "*concept*" and "*object*" interchangeably. Although "*object*" usually refers to a physical entity and "*concept*" connotes an intangible entity; they are modeled identically.

be labeled or identified as a generic type then it can be defined as an intension. Intensions correspond to *''class''* structures in Simula [Bir73] and Smalltalk [God82]. In the Flavors object-oriented language [Obj84], an intension is similar to a *''flavor''*, and *''instance variables''* correspond to properties. The NETL knowledge representation language [Fah79] refers to intensions as *''type nodes''* and properties as *''roles''*. In KL-ONE [Bra85], intensions are analogous to *''generic concepts''* and the properties are called *''dattrs''*.

An *instance* represents an object in the world being modeled and is an instantiation of an intension. The world being modeled may be a subset of the real world, or may be a self-contained imaginary world, such as that described in a fictional book. In either case, an instance stands for a unique identifiable object in that world. Whether the instance corresponds to a real world object or not, depends on the world being modeled. For example, if we are modeling the Los Angeles Zoo, and Juliette is a giraffe at the zoo, then Juliette is an *instance* of the giraffe *intension*. Note that Juliette is also an instance of the animal intension. Under these assumptions, however, we cannot cite any instances of the *unicorn* intension. Instead, if we are modeling a fictitious world where unicorns exist, instances of the unicorn intension can be identified. Flavors and Smalltalk also refer to objects in the modeling world as *''instances''*. NETL calls its instances *''individual nodes''* and KL-ONE's instances are *''individual concepts''*.

An instance refers to an identifiable object, however, the *description* component associated with each instance represents the instantiated property/value pairs of the corresponding intension. A description is derived directly from an intension and an instance. The intension provides the properties, or schema, and the instance supplies the property values, or data for the description. There is a one-to-one correspondence between instances and descriptions. In (traditional) relational data bases there is no notion of an identifiable *instance*, however, the *description* is analogous to tuple values in a relation. It is incorrect to relate a tuple with an instance because tuples are representations of *sentences* and instances are representations of individual *objects*. However, we can say that the instance together with its property values is also a representation of a sentence describing the properties of the object. Furthermore, tuples are identified by their attribute values, not by a unique identifier. If two tuples in a relation are identical in attribute values, they would be indistinguishable and collapsed into one tuple. If, however, two giraffes at the Los Angeles Zoo, Juliette and Oscar, had the same values for height, weight, and habitat, they would still be two separate and unique instances. Most representation systems do not define an explicit primitive comparable to a description. Instead, they assign values to properties of instances, in effect, producing an instance description.

The *extension* is a component representing a collection or set of instances. It models the extensional semantics of objects and concepts. Each intension has a corresponding extension, although, the extension may be empty if no instances have been defined. It is important to maintain the distinction between an extension, the set of all instances; and the intension, which represents a prototypical object in the set. As noted earlier, many systems are lacking this distinction or do not provide the notion of an extension at all. Flavors and KL-ONE do not explicitly maintain sets of instances. NETL defines "*set nodes*" which are similar to extensions to represent the set of all objects corresponding to a *''type node''*. The relational data base notion of an extension is similar in that it maintains a collection of data tuples; it is different because its data tuples are descriptions of instances, not instance objects themselves.

Figure 5.1 uses the four primitives, presented above, to describe the concept of a cat and its instances. Notice that the intension provides a template for the description, and the extension contains instances as elements.

The following discussion defines these four primitive components more rigorously in terms of set theory.

5.2.1.1 Intensions

An intension is represented as a set of ordered pairs, where the first member of each pair is a property name and the second member is a value set. Properties describe a prototypical object, and value sets constrain the value of a property. The intensions for *CAT* and *AUTO*, are expresses as follows. (As a notational convention, all characters of an intension name are capitalized.)

$$CAT: \{(color\,\{grey, black, brown\}), (weight\,\{x \mid x < 20\}), (food\,\{purina, 9\text{-}lives\})\}$$

$$AUTO: \{(color\,\{red, blue, white, green, yellow, brown\}), (weight\,\{x \mid x < 5000\}), (wheelbase\,\{x \mid x < 150\})\}$$

In vernacular terms, the *CAT* intension represents an object whose color is grey, black, or brown; weighing less than 20; which eats either purina or 9-lives. Because intensions are not *complete* definitions, other objects may be represented by equivalent intensions.

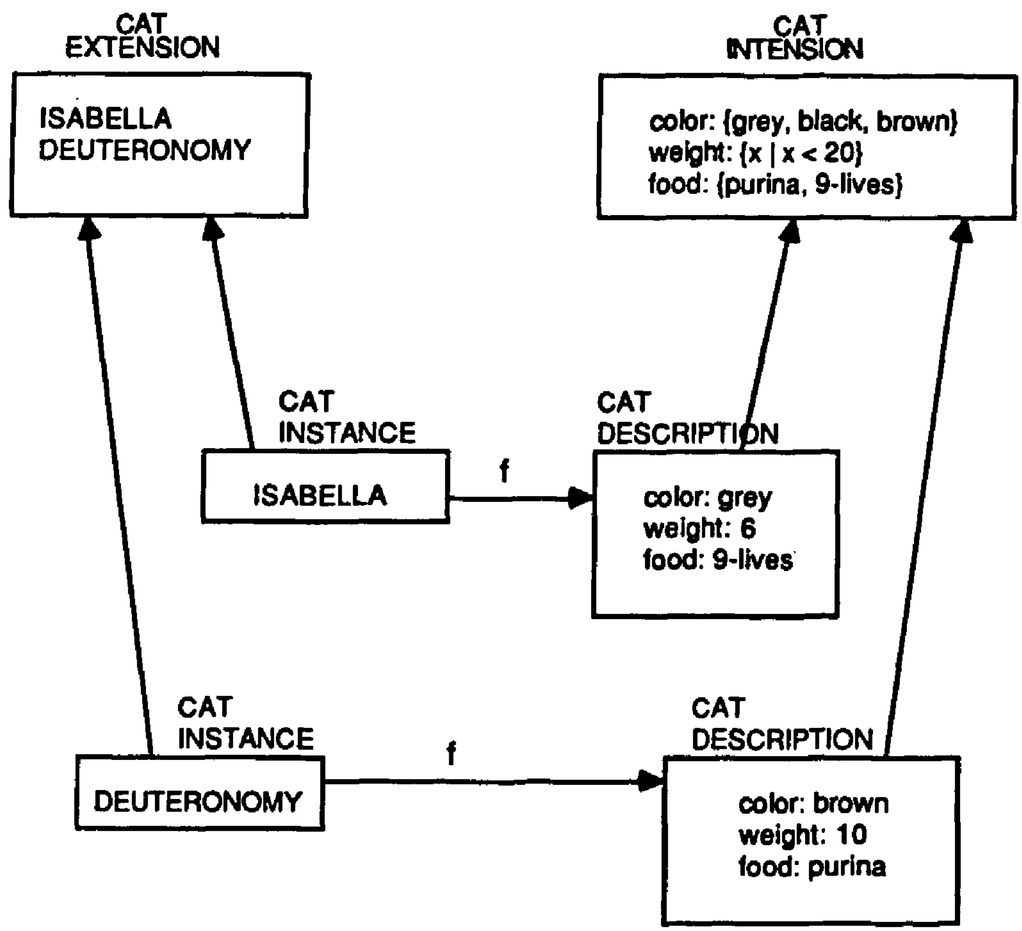


Figure 5.1 ODM primitives representing two cats

5.2.1.2 Instances

A constant represents an instance in the world being modeled and is the unique identifier assigned to a specific object. The associated identifier is the name by which an instance is accessed. In Figure 5.1, *Deuteronomy* and *Isabella* are two instances of the *CAT* intension. Any future reference to these instances is performed via their instance names. (Note that the first letter of each word of an instance is capitalized.)

Descriptions relate instances to intensions through property values. A description consists of a set of property/value pairs. The properties correspond to those properties of the object intension, and the value is a member of the property's value set. A one-to-one mapping, denoted as "*f*", is defined between an instance and a description, and the instance name provides an explicit reference to the object.[1] The name, or identifier, representing an instance may be considered as an abbreviation for a fixed collection of attribute values, some specified in the description and some unspecified. This assumption is consistent with an earlier statement that intensions, and therefore instances, are not *complete* definitions. In Figure 5.1, the description of the instance, *Deuteronomy*, corresponds to the *CAT* intension where value sets are replaced with specific values for *Deuteronomy*. The description of the *Deuteronomy* instance, $Deuteronomy_D$, in set notation is the following:

$Deuteronomy_D$: {(*color brown*) (*weight* 10) (*food purina*)}

[1]In ODM, a separate and unique description is generated for each instance, although in some cases, the content of the descriptions may be equivalent. As a result of this requirement, the function "*f*" is one-to-one rather than one-to-many.

The mapping *"f"* from instances to descriptions relates the instance *Deuteronomy* to its description, *{(color brown) (weight 10) (food purina)}*. Inversely, for every description, D, there exists a corresponding instance. This fact is expressed by the following definition in terms of two predicates, *description* and *instance*:

$$description(D) \Leftrightarrow (\exists x)\ instance(x)\ \&\ f(x)=D$$

5.2.1.4 Extensions

Extensions are sets whose members are defined instances. The set contains the collection of instances associated with a particular intension. For each intension, there exists a corresponding extension set, although the set may by empty if there are no instances. The extension of *CAT* from Figure 5.1 is expressed as:

CAT_E: *{Deuteronomy, Isabella}*

Class objects in other object-oriented models are represented by a single *"class"* primitive. In ODM, generic prototypes are distinguished from sets of objects by two separate primitives, intensions and extensions. Consistent with relational DBMS terminology, intensions characterize schemata, and extensions denote data. Furthermore, in other representation languages, instances are *atomic* structures denoting instantiations of a class object. ODM's instances, instead, combine a unique identifier, namely, the instance name, with a descriptive component through an explicit function *f*. Mappings from descriptions to intensions are implicitly specified by property names, and the relationship between instances and extensions is expressed by the set theoretic primitive "*is-element-of*". Although not explicitly illustrated in Figure 5.1, a close coupling exists between the intension and extension of an object. Further discussion of relationships between ODM's primitives are presented in the following section.

5.2.2 Concept relationships

The concept components described above are analogous to the data structures of conventional data models. A data model further enhances its expressive power by offering facilities for relating its data structures to one another. Based on the previous set theoretic definitions, I have augmented the concept components with six primitive relationship types providing a richer modeling environment. In most object-oriented systems, these relationships are discussed casually and have intuitive meaning. Below I present axioms to describe these inter- and intra-concept relationships.

5.2.2.1 Inter-concept relationships

Member expresses a relationship between instances and extensions. Intuitively, *member* identifies the extensions which an instance belongs to. The *member* predicate, axiom (1), tests for the set theoretic relationship *is-element-of* between an instance and an extension. In Figure 5.1 "$member(Deuteronomy, CAT_E)$" is true but "$member(Lassie, CAT_E)$" is false. Axiom (1) uses predicates *instance* and *extension* to test for instance and extension arguments. *Member(Ins,E)* is true if *Ins* is an instance component and *E* is an extension component, and *Ins* is an element of *E*.

(1) $member(Ins, E) \Leftrightarrow instance(Ins)\ \&\ extension(E)\ \&\ Ins \in E$

Instantiation relates descriptions and intensions. An intension is *instantiated* to yield the description of a specific instance. Referring to Figure 5.1, the description of Isabella is an instantiation of a generic cat. Notice that the instantiation relationship is over descriptions, not instances. However, I have defined a one-to-one function *f* that maps instances to their descriptions, therefore, it is easy to refer to the corresponding instance. Two main conditions define instantiation: (a) For every property/value pair in the description, *D*, there must be a corresponding property/value-set pair in the intension, *INT*, and (b) the value of the property in the description must be contained in the value set of its intension. Axiom (2), below, expresses instantiation more formally. In axiom (2), *description* and *intension* predicates verify the component types of "*D*" and "*INT*". Condition (a) is expressed by the implication (2.a) below, where *v* represents a property value and *y* is a value set for property *P*, of intension *INT*. The second condition, expressed in (2.b), requires that value set *y* contains *v*.

(2) $instantiation(D, INT) \Leftrightarrow description(D)\ \&\ intension(INT)\ \&$

$$(\forall (P,v))\ (P,v) \in D \Rightarrow$$

(2.a) $$(\exists y)\ (P,y) \in INT\ \&$$

(2.b) $$v \in y$$

Figure 5.2 shows the same components as Figure 5.1, plus the identification of *member* and *instantiation* mappings.

5.2.2.2 Generalization

The relationships *member* and *instantiation* and the function *f* express inter-concept mappings. Using these mappings, any component of a particular concept may be accessed. It is not particularly interesting, however, to consider any single concept in isolation. The hallmark of knowledge representation sys-

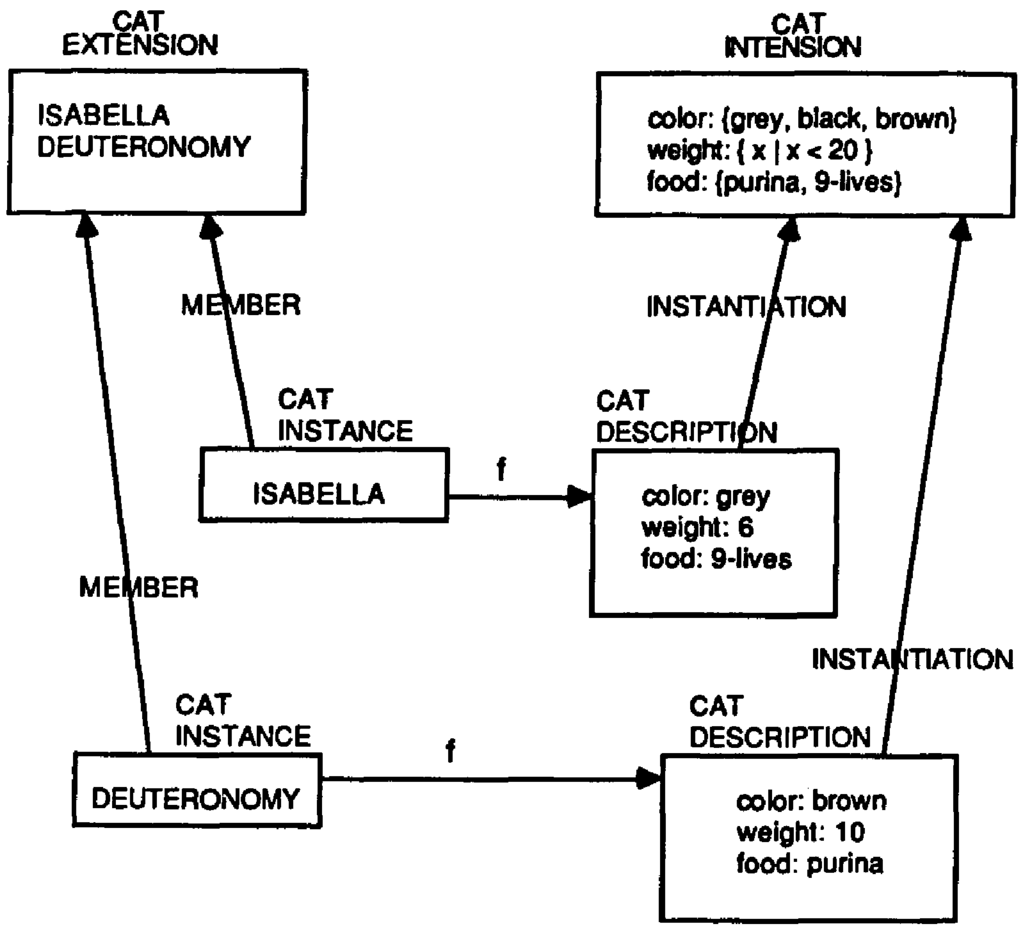


Figure 5.2 ODM primitives with inter-concept links

tems and object-oriented models is their facility for combining concepts into complex structures to reflect real world scenarios. The following two relationships express intra-concept linkages, establishing what is generally referred to as ***generalization hierarchies***. Generalization is a mechanism for building taxonomic structures for concept classification. The principle promotes *abstraction* of common properties of different concepts into a single concept. Reference to the single unifying concept encompasses the more specialized concepts. Below I present two relationships, *subclass* and *specialization*, for establishing generalization mappings between different concepts.

The *subclass* relationship maps extensions to extensions. One extension, E_1, is a subclass of another, E_2, if the set of instances of E_1 is a subset of E_2.

(3) $subclass(E_1, E_2) \Leftrightarrow extension(E_1) \ \& \ extension(E_2) \ \& \ E_1 \subset E_2$

The *subclass* relationship, expressed in axiom (3) above, follows naturally from the definition of member: every member of $HONDA_E$ is a member of CAR_E , therefore, $HONDA_E$ is a subset of CAR_E. Intuitively, the set of Hondas is indeed a subclass of the set of cars. Subclass establishes a generalization hierarchy for the extension components of two concepts. The analog relationship for intensions is the specialization mapping.

Specialization is a relation over intensions. Two aspects of an intension are involved in a specialization relationship. The first aspect is the *extent* of the intension and the second is the *specificity* of each property contained in the intension. The extent of the intension refers to the number and type of property/value-set pairs found in the intension. A *HONDA* exhibits all the properties of a *CAR* and also contains additional properties making it more specialized than a generic *CAR*, for example, the country it was imported from. In this respect, the extent of a *HONDA*'s intension covers the extent of a *CAR*.

The second aspect of specialization addresses individual property/value-set pairs of intensions. For each property found in both intensions, the value set of the property for the specialized intension is a subset of the value set of the same property of the more general intension. For instance, the color of a car may be black, red, blue, yellow, green, brown, or white, but the color of a Honda may only be white, red, or blue. Similarly, the weight of a Honda is less than the weight of any car in general. Although a Honda has more properties than a car, for each property that they share, the allowable values of the property for the Honda are more restrictive than for the car. Based on these two aspects of specialization, a *HONDA* is a specialization of a *CAR*. The axiomatic description of specialization given below, (read "INT_1 *is a specialization of* INT_2"), covers both extent (4.a) and specificity (4.b) of property/value-set pairs. Value sets of the more general intension, INT_2, are denoted by v, and w represents value sets of the specialized intension, INT_1.

$$\text{(4)}\quad specialization(INT_1, INT_2) \Leftrightarrow intension(INT_1)\ \&\ intension(INT_2)\ \&$$
$$(\forall (P,v))\ (P,v) \in INT_2 \Rightarrow$$
$$\text{(4.a)}\quad (\exists (P,w))\ (P,w) \in INT_1\ \&$$
$$\text{(4.b)}\quad w \subset v$$

Figure 5.3 illustrates the primitive components of two concepts, *car* and *Honda*. One instance of Honda, *MyHonda*, is defined, and inter-concept mappings *member* and *instantiation* are labeled. In addition, *subclass* and *specialization* relationships are shown.

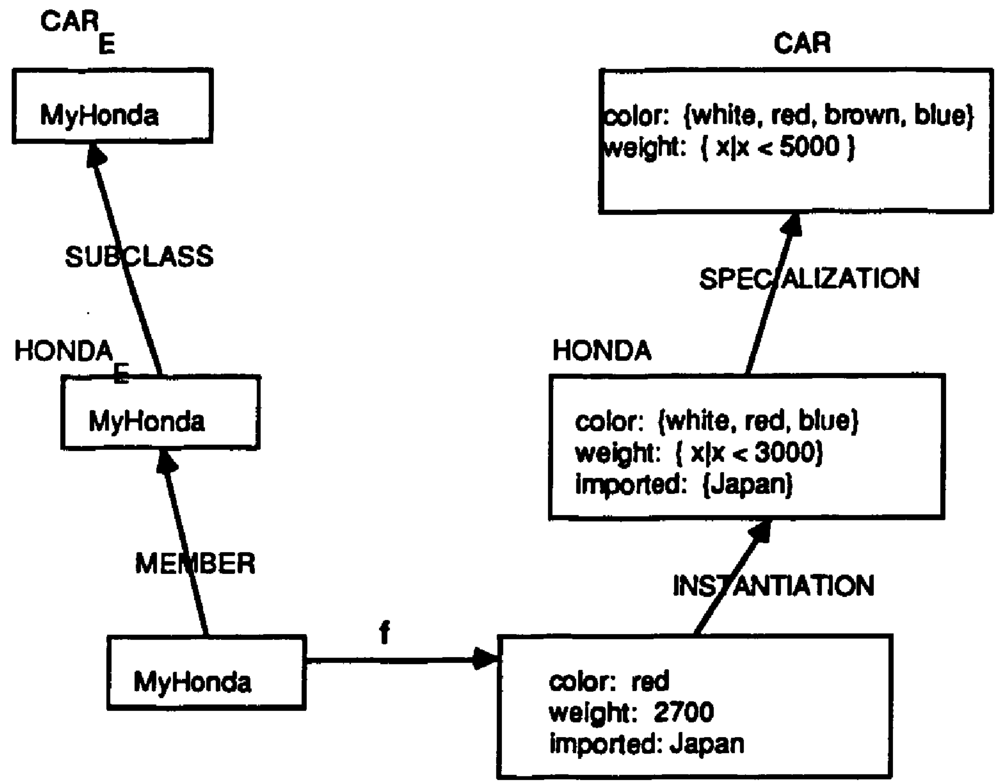


Figure 5.3 ODM primitives with intra-concept links

Although the examples shown so far have illustrated strict generalization *hierarchies*, nothing in the definition of subclass and specialization prevent an object from having multiple parents, thereby creating a *network* organization. For instance, the concept of a *MOTOR-HOME* is a specialization of both a *HOUSE* and a *VEHICLE*. Therefore, *MOTOR-HOME* inherits properties of both generalization objects. However, axioms (3) and (4) do prevent *cycles* in a generalization network by requiring the *proper subset* operator "$\subset$" between intensions and between value sets of intensions, rather than "$\subseteq$". This restriction is consistent with the semantics we want to model, namely, if *specialization(x,y)* and *specialization(y,z)* are true, then *z* cannot be a specialization of *x*.

5.2.2.3 Aggregation

Aggregation is an abstraction principle addressed extensively in data base research [Myl80, Smi77], but to a lesser degree in knowledge representation. In data base theory, aggregation refers to conceptual grouping of parts into a whole. The concept of a plane reservation is the aggregation of individual concepts such as airline, flight number, departure time, seat assignment, etc. In knowledge representation, aggregation is identified by the *"is-part-of"* relationship [Fah79] in the same way that generalization informally refers to the *"is-a"* relationship. Both are considered abstraction mechanisms for constructing complex structures from individual concepts. Most object-oriented systems, however, are based strictly on generalization. Object-oriented models, to date, have not explored the use of aggregation as an alternative or additional hierarchical organization.

In ODM, concept definition is independent of generalization. Intensions and instances can be defined without establishing specialization or subclass relationships. Furthermore, subclass and specialization are expressed *in terms of* ODM's concept primitives. Similarly, aggregation axioms are derived from the set theoretic definition of concept components but are independent of concept definition and generalization. In ODM, neither organization dominates. These premises contrast with other object-oriented languages whose default organization is generalization.

In ODM, aggregation is limited to the composition of *physical* (real or imaginary) parts of an object in the world being modeled. A car is the aggregation of its immediate subparts, namely, body and engine. Engine, in turn, is the aggregation of cylinder, piston, and crankshaft. There are two motivating reasons for this limitation. First, ODM was initially designed for CAD/CAM applications where physical containment is ubiquitous. A very close analogy exists between a Bill of Materials (BOM) hierarchy and the physical aggregation of objects. Furthermore, BOM and parts explosion processing is an awkward task in most data base management systems. Limiting aggregation to physical containment helps to focus on developing more natural representations for BOM and CAD/CAM data. Second, transitivity of the *is-part-of* relation holds under the assumptions of physical containment, and theorems integrating generalization and aggregation can be proven.

Aggregation principles are based on the property ''*primitive-parts*''. If x is an object, and y is a subpart of x which cannot be further decomposed, then y is a *primitive part* of x. This property is the basis of the *gcontains* and *contains* axioms described below, and in theory, can be specified as a property in object intensions and descriptions.

Gcontains (generic contains) expresses containment between intensions. Gcontains is derived from the following premise: If the primitive parts of y are a subset of the primitive parts of x, then y is a (non-primitive) part of x and *gcontains(x,y)* is true. Figure 5.4 shows a pedagogical BOM hierarchy. In this example, the primitive parts of a car are the leaf nodes: bolt, ring, valve, and bearing. Similarly, the primitive parts of an engine are ring, valve, and bearing. Based on the intuitive definition of primitive part and gcontains given above, an engine is a (non-primitive) part of a car and *gcontains(car, engine)* is true

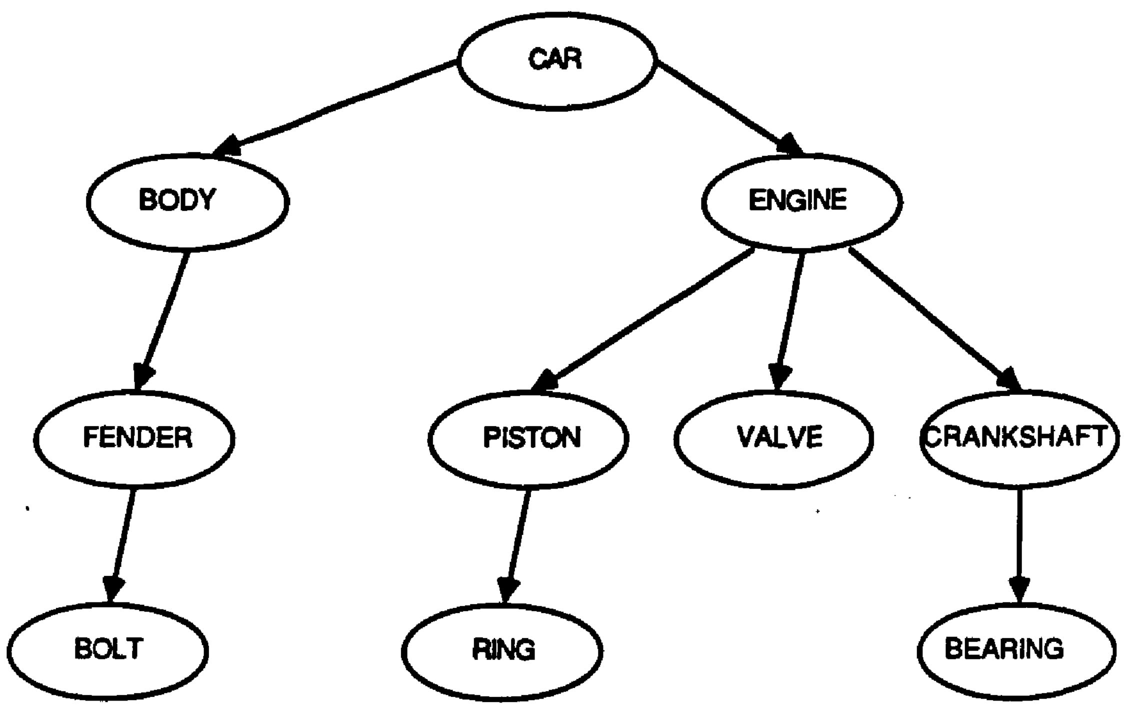


Figure 5.4 ODM BOM hierarchy

The primitive-parts property and corresponding value set adhere to the same semantics as any other property, such as color. That is, a value set represents the set of possible values for the corresponding property of an instance. Value set specifications of the primitive-parts property refer to specific instances of primitive subparts. In ODM, sets of specific instances are extensions. Therefore, value sets for the property *primitive-parts* represent some combination of extensions.

The specification of *CAR* and *ENGINE* intensions with *primitive-parts* properties is given below, where $\wp$ represents the power set.

$$CAR: \{(primitive\text{-}parts\ \ \wp(BOLT_E \cup RING_E \cup VALVE_E \cup BEARING_E\ ))\}$$

$$ENGINE: \{(primitive\text{-}parts\ \ \wp(RING_E \cup VALVE_E \cup BEARING_E\ ))\}$$

The value set for *CAR*, corresponding to the property *primitive-parts*, is the power set of the union of four extensions: $BOLT_E, RING_E, VALVE_E$, and $BEARING_E$. Elements of each set in $\wp$ are members of the relevant extensions. Therefore, the primitive parts of a specific *CAR* is a set whose elements are instances of the objects: *BOLT, RING, VALVE, BEARING.*

The above discussion of *primitive-parts* properties serves only to motivate the definition of gcontains presented in axiom (5). I have demonstrated that *gcontains* is based on ODM notation and formalisms, already discussed and understood. This methodology for representing physical containment extends the traditional functionality of object-oriented properties and values without sacrificing the formalism of the model.

Axiom (5) states that INT_1 contains INT_2, if and only if INT_1 and INT_2 are intensions; and, if *primitive-parts* is a property of INT_2 then (5.a) *primitive-parts* is a property of INT_1, and (5.b) the value set w is a subset of v. Using the *CAR* and *ENGINE* examples above, we see that *primitive-parts* is a property of both intensions; and the value set of *ENGINE* is indeed a subset of the extensions representing the value set of *CAR*. Therefore axiom (5) applies, and *gcontains(CAR,ENGINE)* is true.

$$\text{(5)}\quad gcontains(INT_1, INT_2) \Leftrightarrow intension(INT_1)\ \&\ intension(INT_2)\ \&$$
$$(\exists v)\,(primitive\text{-}parts, v) \in INT_2 \Rightarrow$$
$$\text{(5.a)}\quad (\exists w)\,(primitive\text{-}parts, w) \in INT_1\ \&$$
$$\text{(5.b)}\quad w \subset v$$

When a subpart is contained in more than one superpart, network structures of *gcontains* relationships can be defined. For instance, in Figure 5.4, a *BOLT* may be contained in many other assemblies. However, the definition of *gcontains* in axiom (5) prevents an assertion that a bolt contains a fender.

Contains is analogous to gcontains, but represents containment of instances, not intensions. Because instances do not have structure of their own, the function *f* maps instances to their descriptions, and containment is expressed between descriptions. The property *primitive-parts* is also the foundation underlying the contains relationship between instances. The value of the *primitive-parts* property of an instance, however, contains the names of instances of the corresponding primitive parts. Suppose *instantiation(f(Car005), CAR)* and *instantiation(f(Engine009), ENGINE))* are true, where *CAR* and *ENGINE* are intensions defined above. The descriptions of *Car005* and *Engine009* might be the following:

$Car005_D$: *(primitive-parts {Bolt005, Ring009, Valve004, Bearing002})*

$Engine009_D$: *(primitive-parts {Ring009, Valve004, Bearing002})*

In these descriptions, the value of *primitive-parts* is a set of instances. Using the definition for primitive part, we see that the primitive parts of *Engine009* are a subset of the primitive parts of *Car005*; therefore, *Car005* contains *Engine009*.

The formal definition of contains, axiom (6), corresponds closely to the definition of gcontains for intensions. Although contains relates instances, and properties are associated with descriptions; the function *f* supports the mapping from instances to descriptions.

$$
\begin{aligned}
(6)\quad & contains(Ins_1, Ins_2) \Leftrightarrow instance(Ins_1)\ \&\ instance(Ins_2)\ \& \\
& \quad (\exists D_1)(\exists D_2)\ f(Ins_1) = D_1\ \&\ f(Ins_2) = D_2\ \& \\
& \qquad (\exists\, v)\ (primitive\text{-}parts, v) \in D_2 \Rightarrow \\
& \qquad\quad (\exists\, w)\ (primitive\text{-}parts, w) \in D_1\ \& \\
& \qquad\qquad w \subset v
\end{aligned}
$$

As I stated earlier, these explanations provide the basis for gcontains and contains relationships. It is not expected that *primitive-parts* properties are explicitly recorded with intensions and instances. Rather, we now have a formal definition prescribing when gcontains and contains relationships are valid. Furthermore, although containment is expressed in terms of the property *primitive-parts*, in general, inheritance of properties across composition hierarchies is not semantically valid and is not implied by axiom (5) or (6). Figure 5.5 illustrates a network combining instantiation with gcontains and contains. In this diagram, individual components of each object are not shown; only relevant intensions and instances are displayed.

5.2.3 Concept inferences

Using the previous definitions and axioms, I have derived six theorems for inferring relationships or facts not explicitly stored in a domain model. The theorems may be regarded as declarative statements expressing new relation-

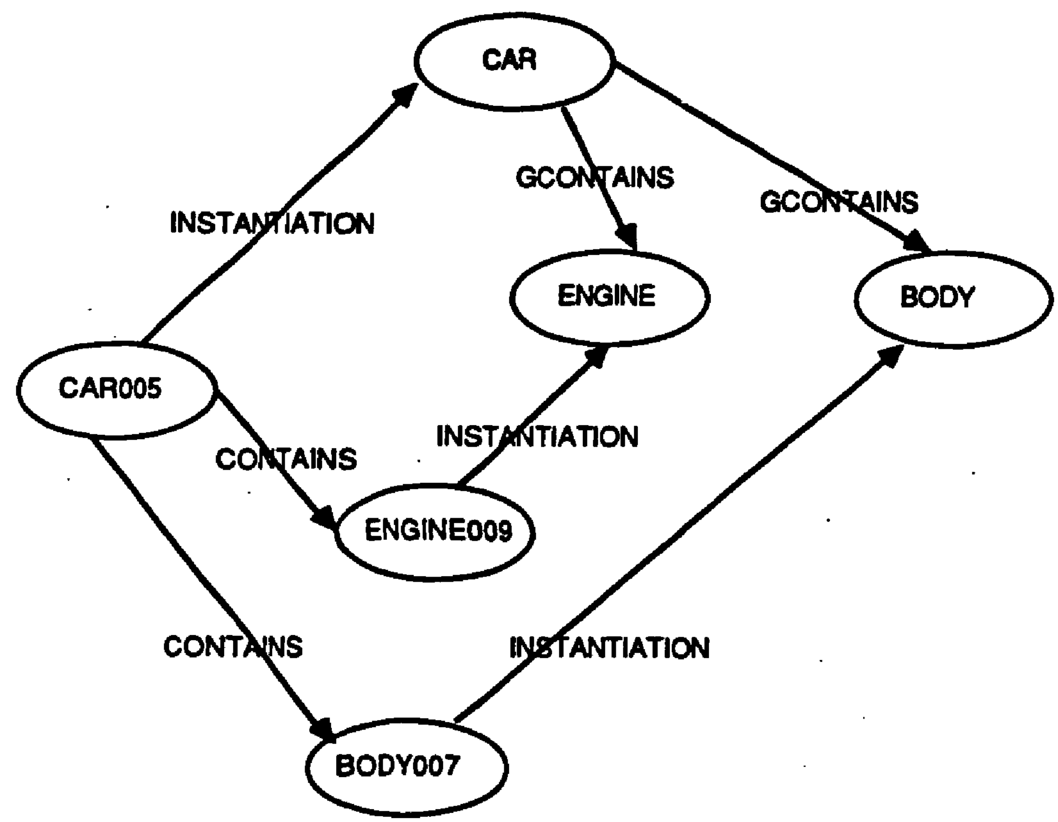


Figure 5.5 Generalization and aggregation links in ODM

ships, or as procedural rules for generating new facts. Below I discuss each theorem and outline the basis of its proof.

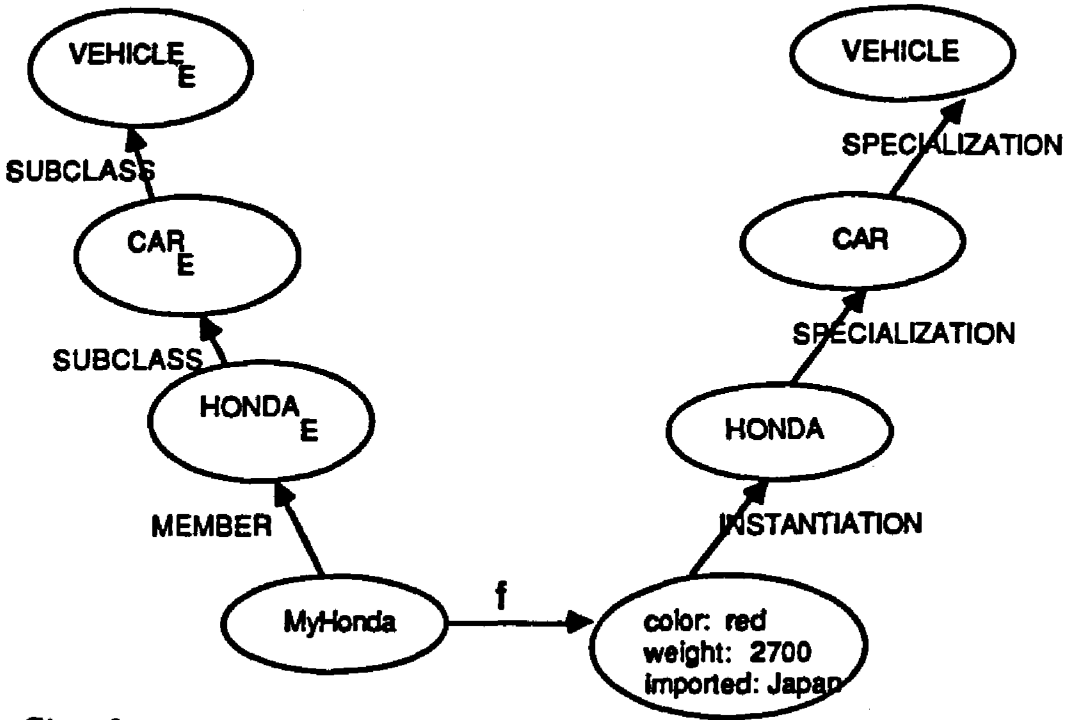
Theorems (7) and (8) express *transitivity* over subclass and specialization. Both of these proofs follow directly from the transitivity of *subset*. Although *inheritance* is not described in the model, I note that theorems (7) and (8) together with axioms (1) and (2) provide the functionality of inheritance. I feel that inheritance reflects implementation issues addressing trade-offs between storing and inferring information. Therefore, I have approached inheritance as an axiomatization of underlying principles supporting the implementation trade-offs.

(7) $subclass(E_1,E_2)\ \&\ subclass(E_2,E_3) \Rightarrow subclass(E_1,E_3)$

(8) $specialization(INT_1,INT_2)\ \&\ specialization(INT_2,INT_3) \Rightarrow specialization(INT_1,INT_3)$

If *specialization(CAR, VEHICLE)* and $subclass(CAR_E, VEHICLE_E)$ are combined with the relationships expressed in Figure 5.3, we can use theorems (7) and (8) to derive the new relationships shown in Figure 5.6. Only the given facts are drawn graphically, however, six derived facts are listed below the network.

A primary motivation for integrating aggregation into an object-oriented framework is to naturally facilitate transitive closure over physical containment. Transitive closure operations are advantageous for BOM and parts explosion processing discussed earlier. By defining *gcontains* and *contains* in terms of the property *primitive-parts*, transitivity of contains and gcontains relationships are preserved through the transitivity of subset. If *gcontains(CAR, ENGINE)* is true, ie., the primitive parts of *ENGINE* are a subset of the primitive parts of *CAR*; and *gcontains(ENGINE, PISTON)* is true, then the primitive parts of *PISTON* are a subset of the primitive parts of *CAR*; ie., *gcontains(CAR, PISTON)* is true. Below, axioms (9) and (10) express transitivity over gcontains and contains relationships.



Given facts:

member (*MyHonda* ,$Honda_E$)
subclass ($Honda_E$,CAR_E)
subclass (CAR_E,$VEHICLE_E$)
instantiation ($MyHonda_D$,*HONDA*)
specialization (*HONDA* ,*CAR*)
specialization (*CAR* ,*VEHICLE*)

Derived facts:

member (*MyHonda* ,CAR_E)
member (*MyHonda* ,$VEHICLE_E$)
subclass ($HONDA_E$,$VEHICLE_E$)
instantiation (*MyHonda* ,*CAR*)
instantiation (*MyHonda* ,*VEHICLE*)
specialization (*HONDA* ,*VEHICLE*)

Figure 5.6 ODM Inferences

(9) $gcontains(INT_1, INT_2)$ & $gcontains(INT_2, INT_3) \Rightarrow$
$gcontains(INT_1, INT_3)$

(10) $contains(Ins_1, Ins_2)$ & $contains(Ins_2, Ins_3) \Rightarrow$
$contains(Ins_1, Ins_3)$

In Figure 5.4, if we view the object nodes as intensions, then the links between nodes represent explicit gcontains relationships. By applying theorem (9), we can generate many implicit *gcontains* mappings. In fact, each non-leaf node is "*gcontains*" related to all of its descendents. For example, *gcontains(CAR, x)*, where *x* represents all other nodes in the network, is true. Similarly, *gcontains(ENGINE, y)* is fulfilled by any descendent of *ENGINE*, ie, *PISTON, VALVE, RING, CRANKSHAFT*, and *BEARING*.

By applying theorem (10), analogous inferences are derivable for instances. Notice, however, that no aggregation relationships map intensions to instances. Because intensions are generic or prototypical objects; any subpart of a generic object will itself be a generic object. Likewise, a specific instance only contains instance subparts. Nevertheless, these theorems offer powerful support for managing BOM hierarchies and transitive closure operations over CAD/CAM schemata and data.

The theorems presented so far have not integrated aggregation and generalization. The following two rules combine *gcontains* with *specialization* to generate containment facts. Explanation of theorems (11) and (12) is best presented through the use of examples based on Figure 5.7.

(11) $gcontains\,(INT_1,INT_2)\ \&\ specialization\,(INT_3,INT_1) \Rightarrow gcontains\,(INT_3,INT_2)$

(12) $gcontains\,(INT_1,INT_2)\ \&\ specialization\,(INT_2,INT_3) \Rightarrow gcontains\,(INT_1,INT_3)$

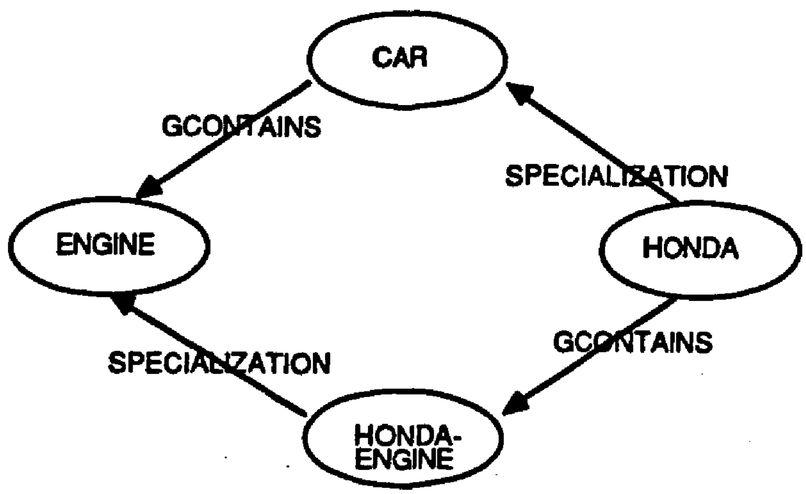


Figure 5.7 Integration of aggregation and generalization

The nodes in Figure 5.7 represent the intensions of four objects: *CAR*, *ENGINE*, *HONDA*, and *HONDA-ENGINE*. Also in Figure 5.7, I have labeled the following four explicit relationships:

gcontains(CAR, ENGINE)
gcontains(HONDA, HONDA-ENGINE)
specialization(HONDA, CAR)
specialization(HONDA-ENGINE, ENGINE)

Looking at the top three nodes of the network: *CAR, ENGINE, and HONDA;* theorem (11) states that if a car contains an engine and Honda is a specialization of a car, then a Honda contains an engine. The dotted link in Figure 5.8 depicts the implied inference. Logically expressed, we have the following:

gcontains(CAR,ENGINE) & specialization(HONDA,CAR) ⇒ *gcontains(HONDA,ENGINE)*

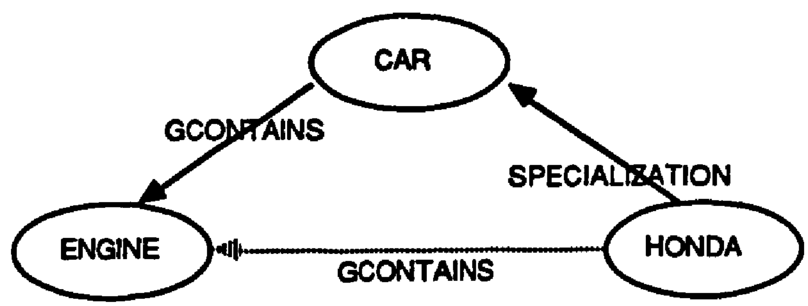


Figure 5.8 Inference derived from theorem (11)

The same fact, namely, *gcontains(HONDA, ENGINE)* can also be proven by focusing on the bottom three nodes of the network and applying theorem (12), where *gcontains(HONDA, HONDA-ENGINE)* and *specialization(HONDA-ENGINE, ENGINE)* are both true. Figure 5.9 illustrates the nodes participating in the implication of theorem (12).

It is not possible, from the network in Figure 5.7, to prove that a car contains a Honda engine. This result is compatible with an intuitive model of the scenario presented in Figure 5.7.

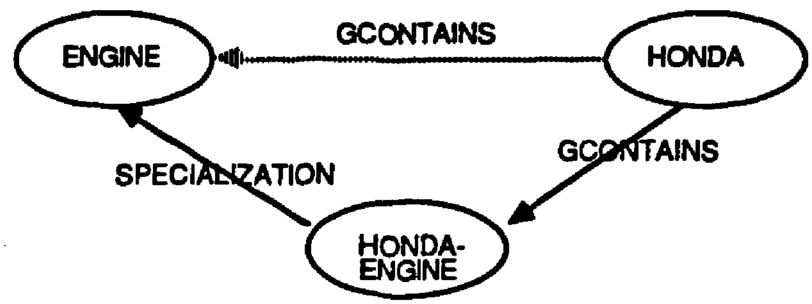


Figure 5.9 Inference derived from theorem (12)

Aggregation and generalization are two independently powerful abstraction techniques. Integrating them within one representation framework supports extensions to typical BOM operations. By combining the six theorems presented above, new information can be derived from previously unrelated data. Future research should address aggregation in a more general fashion to determine if similar logical inferences can be applied without the limitations of physical containment.

Early in this section, I identified three advantages of formal semantics for a representation model. First, ambiguity of definitions and terminology is eliminated in the presence of formal semantics. In ODM, four primitive components and six component relationships are defined in terms of set theory and predicate logic. Second, six theorems are derived from the model's axioms. Inferences generated from the theorems can be proven using the underlying definitions. Finally, ODM can be used as a theoretical modeling tool. The behavior of the model is prescribed by its formalisms and does not depend on a computer implementation. Having fulfilled these theoretical goals, the next task is to demonstrate the practical aspects of ODM for CAD/CAM data management. A prototype implementation, presented in the next chapter, has been developed for analysis and experimentation toward achieving the data management goals presented in Chapters 2 and 3.

CHAPTER 6

ODM PROTOTYPE

In the preceding chapter, I discussed set theoretic foundations of ODM in terms of intensions, instances, descriptions, and extensions. In this chapter, I present the product of this research: an ODM prototype software system. The prototype software I have implemented is a set of integrated computer programs which achieve the functionality of the ODM theoretical model previously presented. The following sections describe how the ODM software system facilitates heterogeneous data types, semantic entities, constraint management, and dynamic schemata. The prototype I describe below is not intended to be a comprehensive data management system. It does not include features and capabilities frequently associated with generalized DBMS, such as sophisticated query languages and techniques for physical organization. Instead, the prototype implementation is meant to provide an operational version of the modeling ideas embodied in the theoretical set-oriented ODM. In addition to describing specific prototype facilities, I compare capabilities in the ODM prototype with analogous DBMS features. For the rest of this chapter I refer to the prototype implementation as "ODM". Therefore, unless otherwise indicated, "ODM" refers to the computer programs realizing the ODM theoretical model. Many sample sessions interacting directly with the ODM computer prototype are presented. In this chapter and Chapter 8, these interactive sessions are identified as "ODM dialogues".

I start with a discussion of modeling capabilities in ODM. This section introduces the construction of intensions and instances; and the use of generalization and aggregation networks for building complex heterogeneous objects. The next section details a data manipulation facility for creating, accessing, and inferring schema and data information. Semantic constraint management is described in section 6.3 followed by a presentation of the methodology underlying ODM's dynamic schema facilities. Section 6.5 concludes with implementation details.

6.1 Modeling facilities

An intension represents a conceptual entity, therefore, creating an intension defines a generic class of objects. When an intension is defined, a corresponding extension is also built, although the extension is empty until instances have been created. For discussion purposes and consistency with other knowledge representation terminology, a *class* of objects refers to an ODM component pair consisting of an *extension* and *intension*.

Once classes are defined, relationships between classes can be specified. In the following discussion, I use a graphical representation for depicting ODM's objects and relationships. Ellipses denote *intensions*, dotted lines between intensions represent *specialization* relationships and solid lines between intensions denote *gcontains* mappings. For example, in Figure 6.1, I show a network modeling ten intensions, five specialization links, and five gcontains links. In this example, a 4-cylinder-engine is a specialization of an engine and is also a subpart of a Honda. Although explicit extensions are not displayed graphically, the extension of an object is generated automatically when the intension is defined. The extension will be empty until instances are entered into the model. For the remaining examples in this chapter, I refer to extensions without displaying their explicit graphical images.

6.1.1 Generalization and aggregation

Although *specialization* relates intensions, and the *subclass* relationship maps extensions; the concept of *generalization* between entities embeds both relationships. In the ODM prototype, if a *specialization* mapping is established between two intensions, a *subclass* relationship is automatically generated. Again, to maintain consistency with other knowledge representation terminology, creation of a *subclass* or *generalization* mapping connotes the establishment of specialization links between intensions *and* subclass links between corresponding extensions. In Figure 6.1, generalization links between vehicle and car, and between car and cadillac imply another generalization relationship between vehicle and cadillac. This result is based on the transitivity of generalization, theorems (7) and (8). By repeated application of theorems (7) and (8), many more generalization links are added. For simplicity and clarity, I show only those links which have been explicitly defined.

Transitivity also holds for aggregation mappings represented as *gcontains* relationships. In Figure 6.1, a piston is part of an engine, and a car contains an engine; therefore, a piston is part of a car. *Gcontains(CAR, CRANKSHAFT)*, although not shown explicitly, is implied by theorem (9). Use of theorem (11) combines specialization and aggregation to generate the facts: *gcontains(4-CYLINDER-ENGINE, PISTON)* and *gcontains(4-CYLINDER-ENGINE, CRANKSHAFT)*.

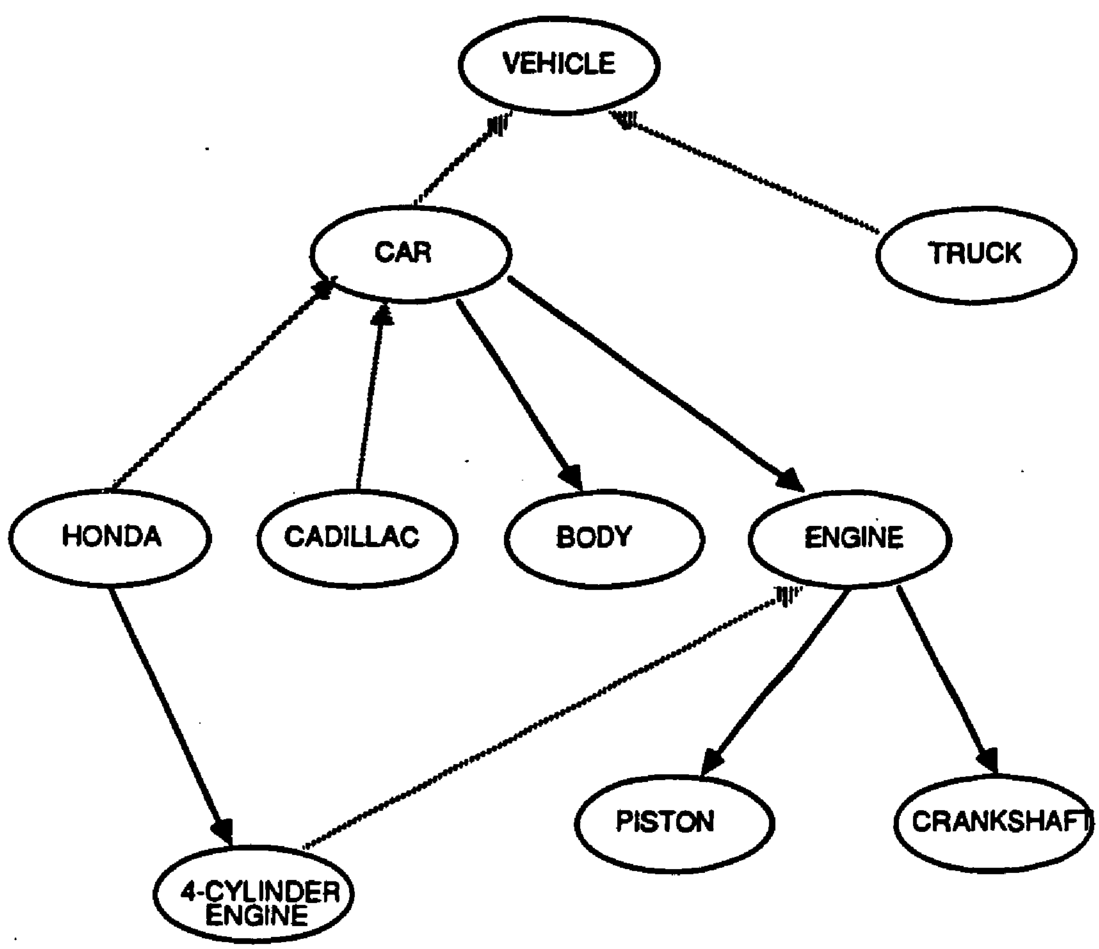


Figure 6.1 ODM network

Primitives for integrating generalization and aggregation hierarchies are not found in any existing DBMS or knowledge representation language. Some projects have addressed the combination of abstraction mechanisms [Myl80, Smi77], however, to date, none have included an axiomatic system for inferencing based on these abstractions. Other CAD/CAM DBMS efforts have recognized the need for built-in aggregation hierarchies and are considering similar mechanisms [Smi84, Bro84]. In ODM, these facilities are the basis for BOM hierarchies and transitive closure operations over CAD/CAM data.

In addition to subpart specification, another critical aspect of BOM data is the *quantity* of a subpart contained in an assembly. Typical BOM schemata in the relational, network, and hierarchical models include a field for subpart quantity. In the following graphical descriptions, subpart quantities are expressed as numbers associated with aggregation links. Figure 6.2 shows the BOM schema of Figure 5.4, with subpart quantities expressed.

I refer to Figure 6.2 as a BOM *schema*, however, in other DBMS models, it best resembles a specification of *data* rather than schema. Compare Figure 6.2 with Figures 3.3 through 3.7. In network, hierarchical, and relational models, references to specific parts, ie. *car*, *engine*, and *body*, exist only in the specification of data instances. In this ODM example, the distinction between schema and data begins to vanish. The intensions shown in Figure 6.2 represent generic objects, not specific instances. Similarly, in a CAD/CAM environment, an engineering drawing models the generic structure of a design. The drawing is instantiated to produce data base instances corresponding to finished products in the specific manufacturing application being modeled.

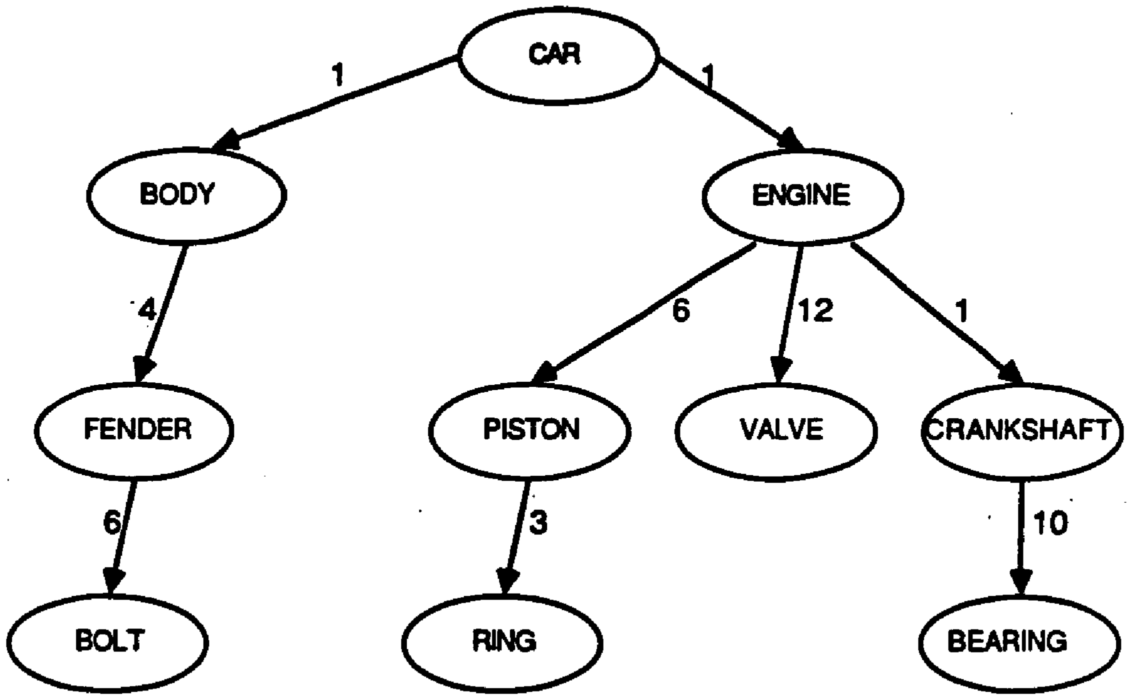


Figure 6.2 ODM BOM hierarchy with subpart quantities

Instances also participate in subpart or *contains* relationships. A BOM schema hierarchy, for example Figure 6.2, may have many corresponding BOM instance hierarchies, such as one partially shown in Figure 6.3. The *contains* relation over instances is also transitive; however, transitivity does not hold over the combination of intension and instance subparts. In the following figures, instances are drawn as bold circles and instantiation links are depicted as bold dotted lines. I have drawn instance subpart links, *contains* relationships, as bold solid lines to differentiate them from intension subparts, denoted as regular solid lines. In Figure 6.3, three intensions: *CAR, BODY*, and *FENDER* are instantiated. The network of regular solid links represents an aggregation hierarchy of intensions; bold solid lines depict the aggregation of instances; and bold dotted lines associate intensions with instances through instantiation links. Nodes on the right hand side of Figure 6.3 are regarded as schema, and the left side represents data instances.

In many knowledge representation systems, instances are created strictly from leaf nodes of a generalization hierarchy. For example, in Figure 6.4, *Clyde* and *Fido* are instances of the the leaf nodes, *ELEPHANT* and *DOG*. In ODM, intensions are neither complete or exclusive, therefore, an object can be an instance of any single intension. If a specific animal has been identified only as a mammal, not an elephant or dog, then the animal should be an instance of the *MAMMAL* intension, a non-leaf node in Figure 6.4. With the same generalization hierarchy, an animal named *Garfield*, also known to be a cat, can only be defined as an instance of *MAMMAL* (or any generalization of mammal). A better alternative, if possible, is to first add a *CAT* intension and then create an instance of *CAT* named *Garfield*. With dynamic schema facilities, discussed in section 6.4, it is possible to dynamically and interactively add new intensions to a data base.

In previous examples, intensions and instances are identified by a conceptual correspondence between the name of an object and its counterpart in the world being modeled. Although the intension representing the concept of a dog is named *DOG*, there is no inherent meaning associated with the intension name *"DOG"*. User selected names of intensions and instances carry no predefined

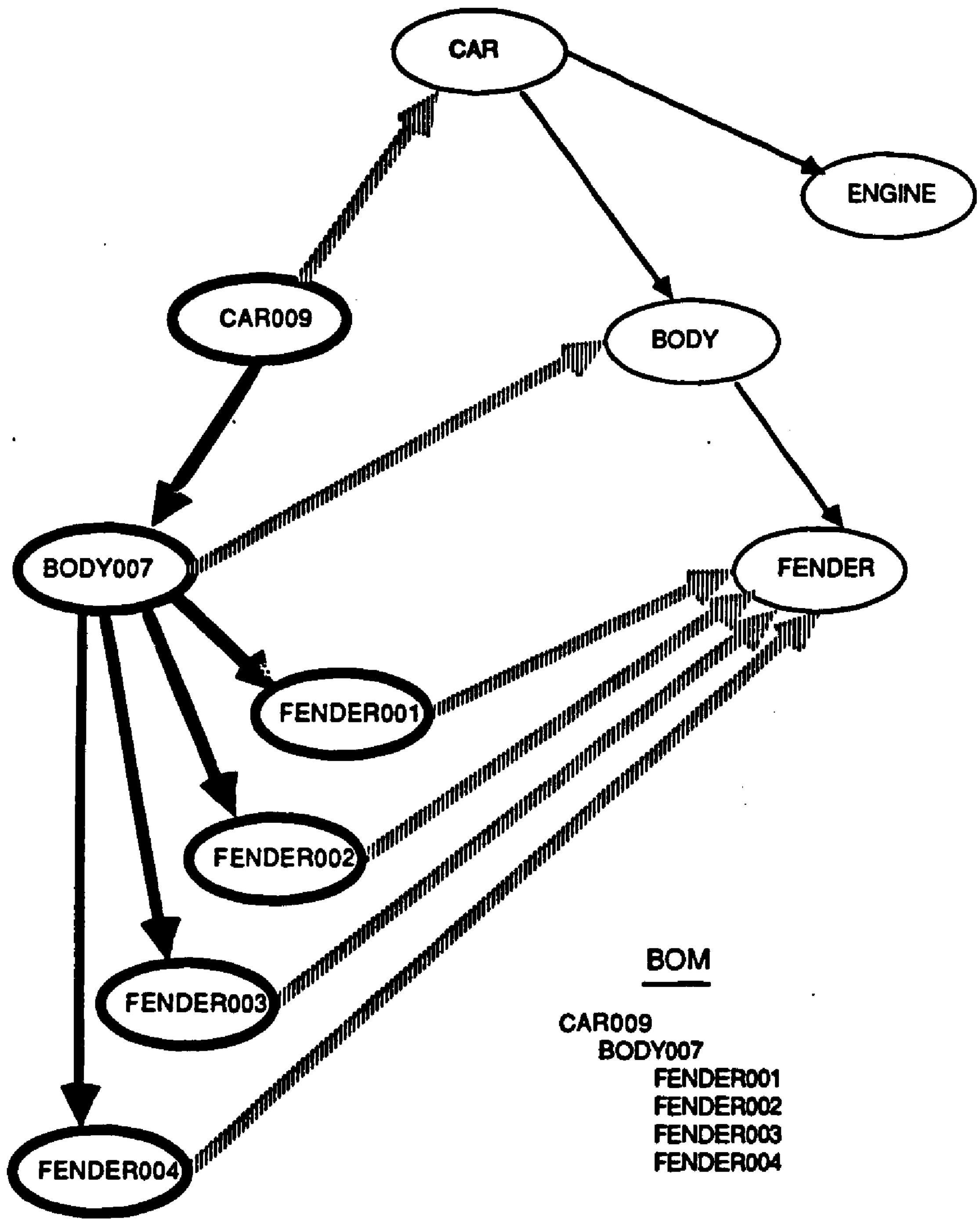


Figure 6.3 BOM instance hierarchy

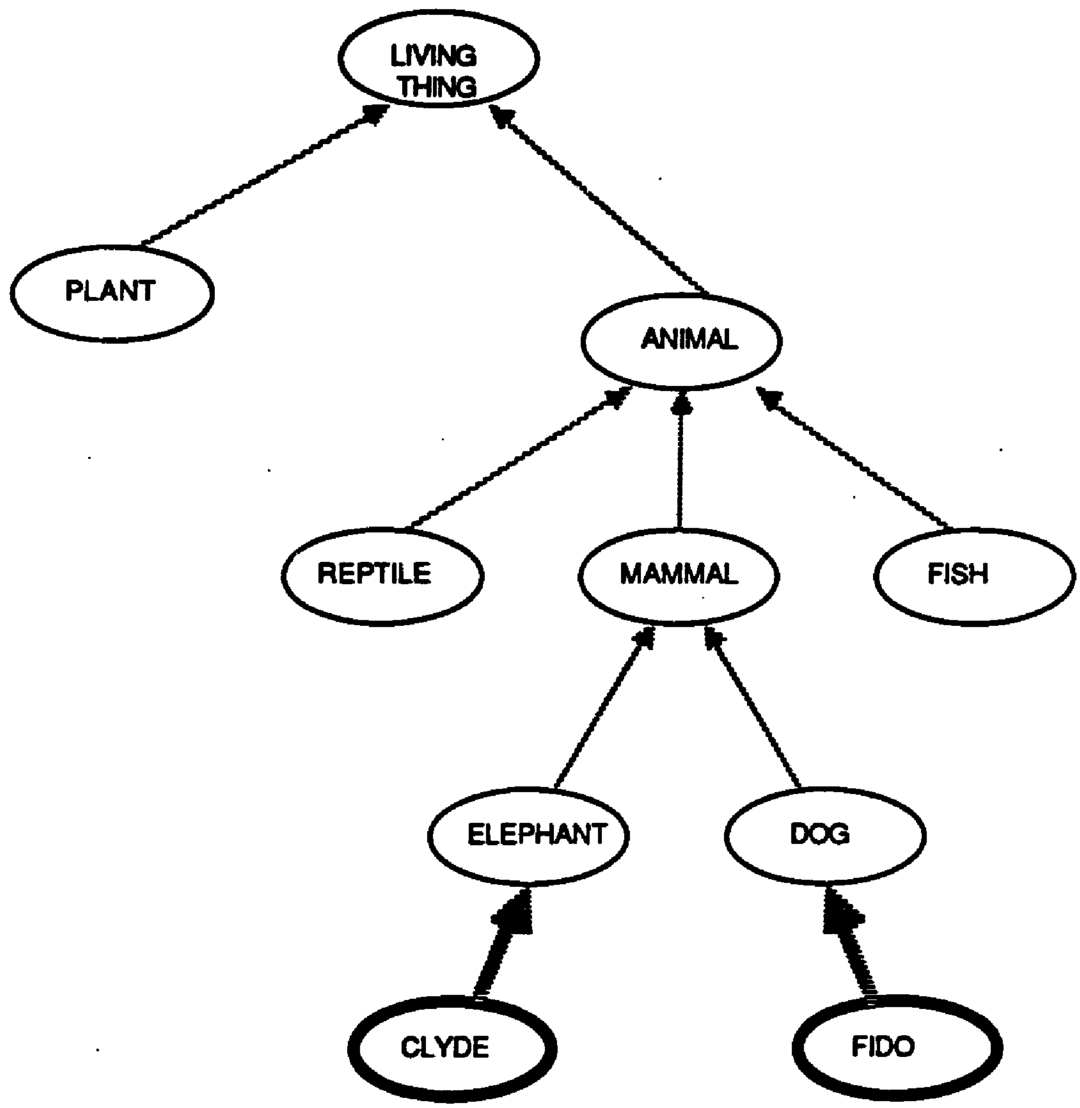


Figure 6.4 ODM generalization network

significance. Furthermore, there are no limitations preventing us from defining an intension named *FOOBAR*, to represent the concept of dog. Data and knowledge base designers, however, try to name entities mnemonically, to reinforce a conceptual correspondence between the symbol *"DOG"* and our inherent notion of a dog.

6.1.2 Properties

Properties are used to construct complex objects and heterogeneous data types in ODM. Properties must be explicitly defined for intensions, but are automatically created for all instances of an intension. Any descriptional attributes of an intension are retained with the intension as properties. Information describing the ***extension*** of a concept, such as the number of instances, is maintained automatically with the extension data structure In ODM, a property is a complex structure. Each property contains seven fixed slots which further describe the property. One of these seven slots is the value associated with the property. Although the slot names are fixed, the values associated with most of the slots are set by the user. Property slots in the current ODM prototype include ***p-name***, ***p-lambda***, ***p-proc***, ***p-units***, ***p-cardinality***, ***p-description***, and ***p-value***. Four of the seven slots (p-name, p-lambda, p-proc, p-value) are required; selection of the other three was based on their relevance to CAD/CAM applications. Additional slots can be added for different domains. Below is an example of the property, ***weight***, defined for the *VEHICLE* intension:

VEHICLE

weight:

p-name: weight
p-lambda: (lambda (x) (lessp x 10000))
p-proc: {procedure 16}
p-units: pounds
p-cardinality:
p-description: "the weight of a vehicle"
p-value:

Only those slots which are applicable for the property, *weight*, are assigned values. A summary of the slots and their use is given below:

p-name: the name of the property

p-lambda: a lambda expression constraining the value of the property

p-proc: a procedure identifier set by the system to verify allowable property values

p-units: a units identifier further describing the property

p-cardinality: an integer indicating "how many" of this property exist

p-description: a textual description of the property

p-value: the value of the property

Two slots, *p-name* and *p-proc*, are automatically set by the system. The slot, *p-lambda*, maintains a default value; however, for most properties, the user will override the system default. Other slots are optionally set by the user. The seven property slots described above apply to properties of an intension. Instances inherit properties through their description component and utilize the slot *p-value* to store their specific property value. Detailed discussion and syntax for setting property slots is presented in section 6.2.

The complex structure of properties enables arbitrary value constraints. In the above example, the weight of a vehicle is constrained to less than 10,000 pounds. If a user tries to set the weight of a specific vehicle to a value greater than 10,000, the transaction is rejected. Slots such as *p-units, p-description,* and *p-cardinality* help improve the richness of the modeling environment by including relevant supplemental information. *P-units* and *p-cardinality* are especially useful in design and manufacturing environments where quantity and units information abounds. Other application domains may benefit from different slot descriptors.

One motivation for constructing generalization hierarchies is the distribution of properties, generally referred to as *inheritance*. For example, if the property *weight*, is defined for the intension *VEHICLE*, and *HONDA* is a specialization of *VEHICLE*, then the *weight* property should also apply to *HONDA*. Most knowledge representation languages and semantic data management systems support basic property inheritance across generalization hierarchies. ODM offers more sophisticated forms of inheritance by allowing selective slots of properties to be inherited or reassigned. In Figure 6.1, the *HONDA* intension inherits the *weight* property and also inherits the slot values of the property. However, since a Honda is a specialization of a vehicle, some of its properties are more specialized or constrained. Although it is true that the weight of a Honda is less than 10,000 pounds, we can be more precise in our specification of a Honda's weight by asserting that it is less than 3,000 pounds. ODM allows cascading of value constraints for more precise property specification. In the previous chapter, I referred to this aspect of specialization as the *specificity* of value-sets. To further restrict the value of a Honda's weight, we merely need to reset the *p-lambda* slot of the Honda's weight property to: *(lambda (x) (lessp x 3000))*. If the weight of any Honda instance is set to a value greater than or equal to 3,000, the transaction is rejected. *P-lambda* can also be used for specifying the data types of property values.

6.1.3 Relations

In addition to representing domain objects and properties, a data modeling environment must also accommodate domain-specific relationships. In object-oriented data models, such as ODM, relational data is less prevalent than entity-oriented data. Nevertheless, supporting user-defined relationships is necessary for subsets of data which are most naturally modeled in a relational representation.

In ODM, a generic relationship can be expressed between classes of objects and corresponds to the *relation schema* in relational DBMS. Instantiations of the relationship generate unique relation instances and are similar to *relational tuples*. A class of objects participating in a relationship fulfills a particular *role* of the relationship. This characterization of relations is derived from knowledge representation formalisms based on case grammars [Bra78, Mil76, Sow84]. In a case grammar, the main predicate of an assertion corresponds to the relation, and nominal expressions within the assertion represent *roles*.[1] ODM adheres to this technique by providing primitives for defining *relation objects* and specifying *role constraints*.

A relation object is a special case of an intension and corresponds to a schema description in a relational DBMS. Similarly, the structure of a role resembles a property. Role slots, like property slots, describe and constrain instantiations of the relationship. An instantiation of a relation assigns specific instances of ODM objects as role values. A relation instance is analogous to a *tuple* in an extended relational DBMS which maintains unique tuple identifiers

[1] In this context, "*predicate*" refers to the subject/predicate construction of assertions.

[Gut82]. For data which is primarily relation-oriented, ODM can emulate a relational data model through its *relation objects*, *roles*, and *role values*.

To demonstrate a CAD/CAM application of the constructs presented above, I have mapped a geometric boundary model from its conceptual structure to an ODM representation. The B-rep model in Figure 6.5 was developed by Lillihagen [Lil78] and is typical of many boundary representation models. Links labeled *consists-of* and *contains* exemplify aggregation relationships. Generalization hierarchies are illustrated by objects graphically enclosed within *surface* and *surface unit* entities. Other B-rep relationships, such as *has boundary curve*, *succeeds*, and *has startpoint* are also expressed. Figure 6.6 shows Lillihagen's B-rep model constructed in terms of ODM intensions and relationships. Relational schema are displayed as rectangles whose links point to role intensions participating in the relationship. For each geometrical entity in Figure 6.5, a corresponding ODM intension has been constructed. This example illustrates the close correspondence between conceptual models, like Figure 6.5, and ODM schema structures realizing the conceptual model.

6.1.4 Complex and heterogeneous data types

Although each geometrical entity in Figure 6.6 is displayed simply as an intension, most entities are themselves complex structures. For example, the definition of *POINT* is composed of x, y, and z coordinates. Two different representations of a point intension are shown in Figures 6.7 and 6.8. In these examples, the intension names, property names, and property value specifications are shown. The property name corresponds to the *p-name* slot of the property and the value specification is an abbreviation of the *p-lambda* slot.

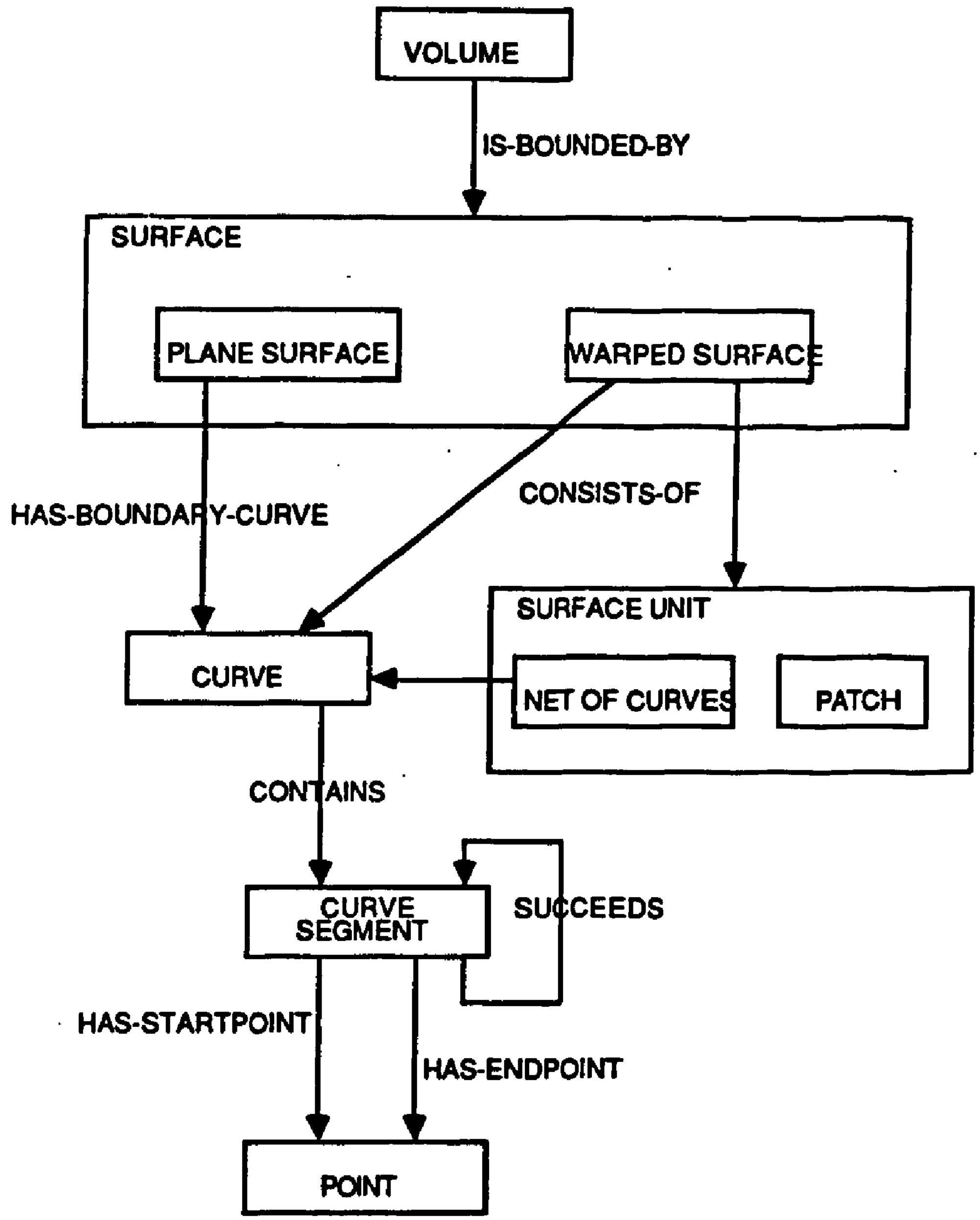


Figure 6.5 B-rep schema for solid volume

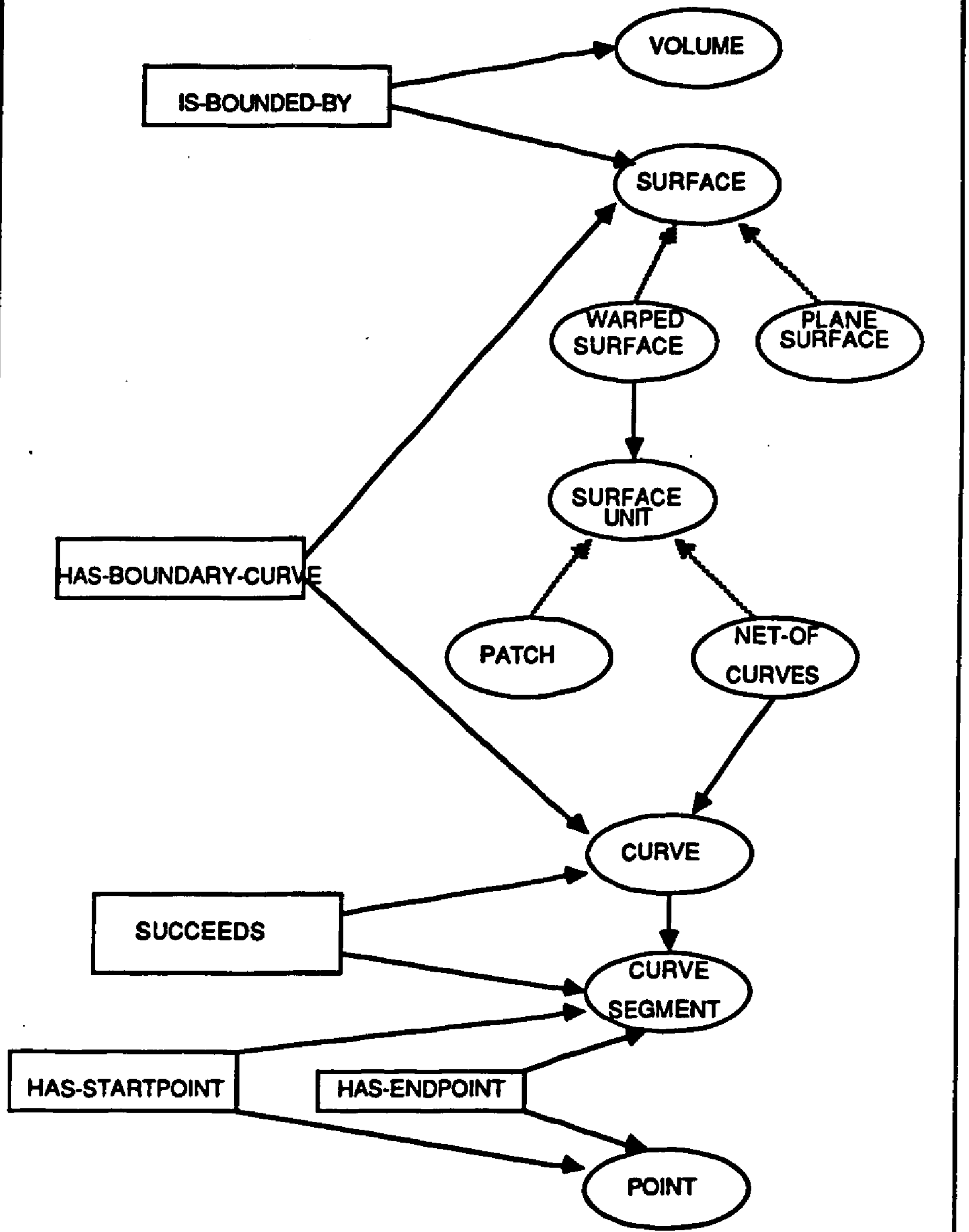


Figure 6.6 ODM network of B-rep model

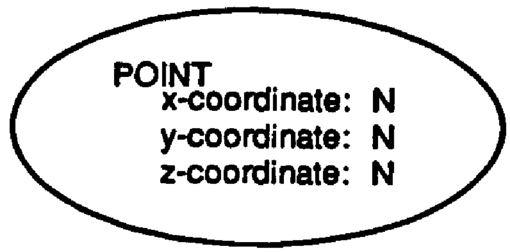


Figure 6.7 POINT intension with 3 properties

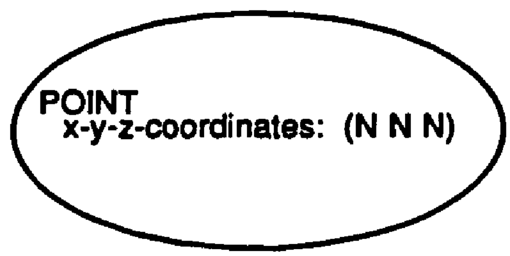


Figure 6.8 POINT intension with one property

Figure 6.7 defines x, y, and z coordinates as individual properties. Each coordinate property is required to be numeric, indicated by the "*N*" value specification, and is accessed and set independently of the others. Figure 6.8 considers the coordinates of a point as a single property accessed by the property name, *x-y-z-coordinates*. This alternate specification represents the coordinates as a list of three numeric elements, ie. "*(6.2 4 8.655)*". Each representation has advantages and disadvantages in terms of overhead for access and modification. The choice of organization is a data base design issue which should consider how the data is accessed interactively and used by applications programs.

The intensions discussed so far have consisted of simple numeric or textual properties. The *LINE-SEGMENT* intension, in Figure 6.9, illustrates a complex data structure whose property value-sets are intensions. A line-segment is defined by three properties: two endpoints and a linear equation. The vertical bar syntax, "|...|", specifies an intension whose instances are the allowable values of the property. For example, the value of an endpoint must itself be an instance of the *POINT* intension. The definition of line-segment is not concerned about *how* the *POINT* intension is defined, ie., Figure 6.7 or Figure 6.8; it only requires that the values assigned for its endpoints are instances of *POINT*. In addition, the value of the property, *equation*, is constrained to be an instance of *LINEAR-EQUATION*. With these definitions, the *LINE-SEGMENT* intension can now be specified as a property value of other intensions, or as a role in a relational schema. By assigning intensions as property values; complex hierarchical structures are constructed.

Sophisticated property specification, in addition to the modeling constructs previously described, support the integration of graphical, geometrical, and manufacturing entities for constructing heterogeneous data types. The generalization hierarchy and property specifications in Figure 6.10 show a schema of fabrication data for three different types of manufacturing jobs. In Figure

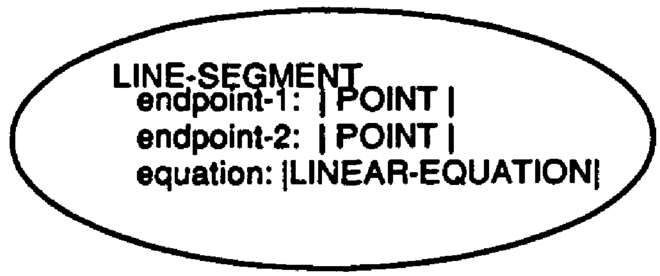


Figure 6.9 LINE-SEGMENT intension

6.10, a literal value is specified as "*L*", and "*S*" denotes any string of characters. The property, ***serial-numbers***, requires a list whose elements are numeric; and, annotations in the properties, ***cylindricality*** and ***concentricity***, indicate units information. In this example, domain specific data types, like intensions ***TOOLING-PROCESS*** and ***GT-SPEC***, are integrated with traditional and extended data types to generate the heterogeneous intension ***FABRICATION-JOB***.

6.2 Data manipulation

In this section I present interactive facilities for constructing and manipulating ODM representations which, until now, have been described graphically. The ODM prototype was implemented to test and evaluate the modeling capabilities of ODM, therefore, data manipulation facilities were not a primary consideration. Nevertheless, it was necessary to develop an ***Object Manipulation Language***, OML, for creating, accessing, modifying, and traversing ODM networks and components. In addition, OML integrates inferencing with its data

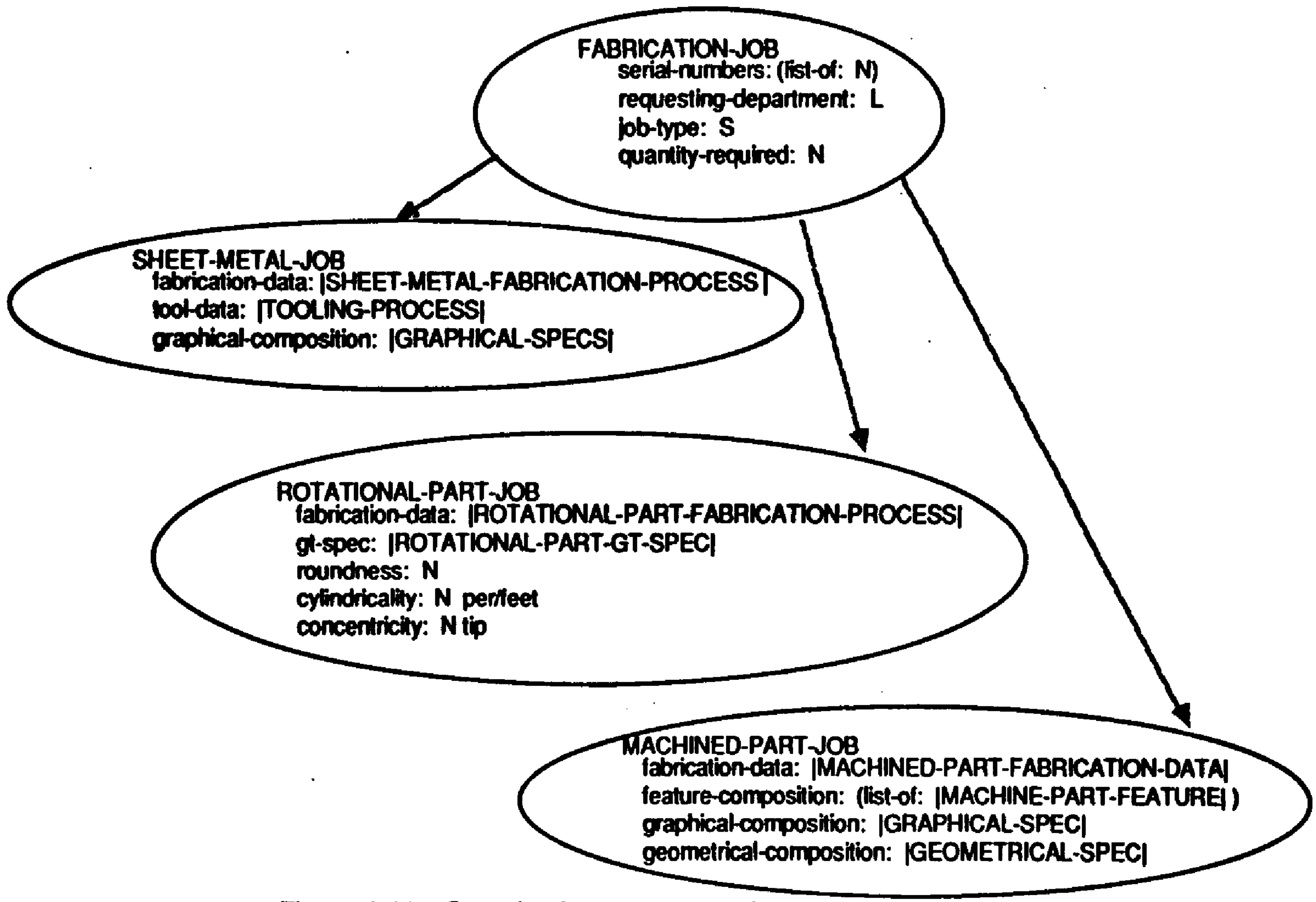


Figure 6.10 Complex heterogeneous intensions

manipulation capabilities.

ODM is implemented in a general-purpose object-oriented programming language. Message-passing is the main technique for procedure invocation; therefore, statements in OML consist of message transmissions, similar to the Smalltalk language. For the initial creation of data base entities, the object-oriented OML is exceedingly verbose. To streamline schema and data input, I developed a simplified *Object Entry Language*, OEL, for entering new intensions, relations, and instances. Schema and data expressed in OEL are parsed, producing equivalent commands in the object-oriented OML, and subsequently loaded into the ODM prototype system. For discussion of object creation and data entry, I use the simplified OEL. Data manipulation, such as setting and retrieving property values, displaying an object, and traversing aggregation and generalization hierarchies is performed in the object-oriented OML. The complete syntax of OML is found in Appendix B.

For pedagogical purposes, I constructed an ODM network displayed graphically in Figure 6.11. Figure 6.12 shows the OEL specification for generating the corresponding ODM intensions and instances. Appendix C contains the object-oriented OML syntax generated by parsing the specification in Figure 6.12. The OEL syntax for creating a new intension is given below:

```
(c <intension> |<optional-superclass>| /<optional-superpart>/
      <property-1>: <value-spec-1> <optional-units-1>
      <property-2>: <value-spec-2> <optional-units-2>
        .
        .
        .)
```

To generate an instance, the following OEL syntax is used:

```
(i <instance> |<intension>| /<optional-superpart>/
     <property-1>: <property-1-value>
     <property-2>: <property-2-value>
      .
      .)
```

The *<value-spec-n>* fields are necessary for entering data type and value constraint information. Figure 6.13 provides a list of predefined atomic and extended data types available in OEL. Any complex data structure not included in OEL can be defined directly in the object manipulation language. A property value specification enclosed in vertical bars restricts the value of that property to some instance of the intension specified.

Data entry, discussed above, is only one aspect of schema and data manipulation facilities. In ODM, data retrieval is primarily object-oriented, that is, the primary access method is through intensions and instances. Each intension and instance is identified by a unique addressable name. In the ODM implementation, this name serves as a symbolic pointer to the internal structure retaining information about the object. User-defined relations are special cases of intensions; therefore, it is also possible to query relations. Navigation through intension and instance networks, such as Figure 6.11 is performed by traversing generalization and aggregation hierarchies. Axioms presented in Chapter 5 support queries related to specialization, generalization, subpart, superpart, and instantiation. Figure 6.14 presents a sample dialogue with the ODM prototype based on the data in Figure 6.11. In the remainder of this document, scripts of direct interaction with the ODM software implementation are identified as *"dialogues"*. In a dialogue, the user's input is preceded by the prompt character ">", and the system's response follows on the next line. The general syntax of OML commands is the following:

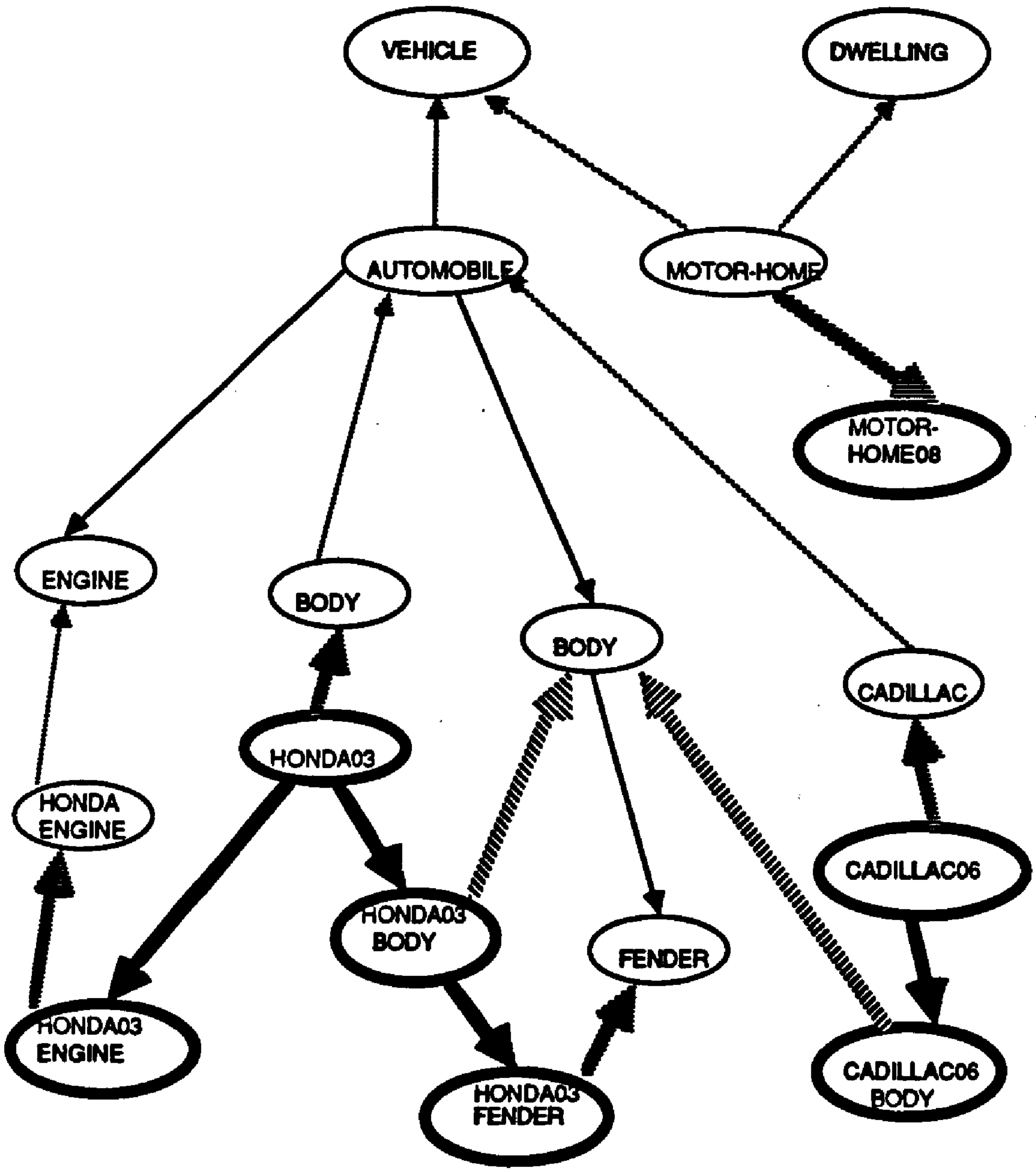


Figure 6.11 ODM network of intensions and instances

(send <object> <OML-message> <message-parameter-1>
<message-parameter-2> ...)

```
(c VEHICLE)

(c DWELLING)

(c MOTOR-HOME |VEHICLE|)
(c MOTOR-HOME |DWELLING|)

(c AUTOMOBILE |VEHICLE|)

(c HONDA |AUTOMOBILE|)
(c CADILLAC |AUTOMOBILE|)

(c ENGINE /AUTOMOBILE/)
(c BODY /AUTOMOBILE/)
(c FENDER /BODY/)

(c HONDA-ENGINE |ENGINE|)

(i Motor-Home08 |MOTOR-HOME|)
(i Cadillac06 |CADILLAC|)
(i Honda03 |HONDA|)

(i Honda03-Engine |HONDA-ENGINE| /Honda03/)
(i Honda03-Body |BODY| /Honda03/)
(i Honda03-Fender |FENDER| /Honda03-Body/)

(i Cadillac06-Body |BODY| /Cadillac06/)
```

Figure 6.12 OEL specification of ODM network

Each statement in OML is surrounded by parentheses, *"(...)"*, and begins with the keyword *send*. The *<object>* field represents the identifier of an ODM component, such as an intension, instance, or relation name. An *<OML-message>* is a string prescribing an ODM action. The *<message-parameter-n>* slots depend on the OML message and provide additional relevant information for

BASIC TYPES	INTERPRETATION	EXAMPLE
I	value must be an integer	age: I
R	value must be a real	temperature: R
N	value must be numeric	distance: N
S	value must be a string	inspection-order: S
L	value must be a literal	department: L
<literal value>	value must be equal to <literal value>	machine-type: "HSS Drill"
COMPLEX TYPES		
|<intension>|	value must be an instance of the named intension	endpoint: |POINT|
(less-than: <n>) (greater-than: <n>)	where <n> is numeric; value must be less or greater than <n>	weight: (less-than: 1000)
(one-of: <element-list>)	where <element-list> is any of the above specs; value must conform to one element in <element-list>	color: (one-of: red blue green white)
(list-of: <element-spec>)	where <element-list> is any of the above specs; value must be a list of any number of elements conforming to <element-spec>	inventory: (list-of: |HONDA|)

Figure 6.13 OEL data types

ODM processing.[1] If the string *''all''* is contained in the *OML-message*, transitivity theorems from Chapter 5 are applied to relevant objects in the data base. New assertions are inferred by repeated application of the theorems. For example, in Figure 6.11, no explicit assertion relates instances, *Honda.03* and *Honda-Fender.03*, however, based on the transitivity of *contains*, ODM infers the subpart relationship between *Honda.03* and *Honda-Fender.03*.

Dialogue 6.1 OML dialogue for traversing ODM networks

```
> (send db is-intension? AUTOMOBILE)
T

> (send db is-intension? VEHICLE)
T

> (send VEHICLE get-specializations)
(AUTOMOBILE MOTOR-HOME)

> (send VEHICLE get-all-specializations)
(AUTOMOBILE MOTOR-HOME CADILLAC HONDA)

> (send HONDA get-specializations)
()

> (send HONDA get-generalizations)
(AUTOMOBILE)

> (send HONDA get-all-generalizations)
(AUTOMOBILE VEHICLE)

> (send MOTOR-HOME get-all-generalizations)
(DWELLING VEHICLE)

> (send AUTOMOBILE get-subparts)
(BODY ENGINE)

> (send AUTOMOBILE get-all-subparts)
(BODY ENGINE FENDER)

> (send FENDER get-superparts)
(BODY)
```

[1]The underlying ODM implementation language is a variant of Lisp, therefore, all objects, messages, and parameters in OML are evaluated during processing. Because ODM's constructs are not global, each field must be *quoted*. For clarity, I have removed the single quotes, " ' ", from the fields of all OML statements.

```
> (send FENDER get-all-superparts)
(BODY AUTOMOBILE)

> (send HONDA get-instantiations)
(HONDA03)

> (send HONDA get-all-instantiations)
(HONDA03)

> (send VEHICLE get-all-instantiations)
(CADILLAC06 HONDA03 MOTOR-HOME08)

> (send db is-instance? Honda03)
T

> (send db is-instance? Cadillac03)
()

> (send Honda03 get-parts)
(HONDA03-BODY HONDA03-ENGINE)

> (send Honda03 get-all-parts)
(HONDA03-BODY HONDA03-ENGINE HONDA03-FENDER)

> (send Honda03-Fender get-assemblies)
(HONDA03-BODY)

> (send Honda03-Fender get-all-assemblies)
(HONDA03-BODY HONDA03)

> send Honda03 get-intension)
HONDA

> (send Motor-Home08 get-intension)
MOTOR-HOME

> (send HOTOR-HOME is-specialization? DWELLING)
SPEC.55

> (send Honda03 is-instantiation? VEHICLE)
SPEC.58

> (send FENDER is-subpart? VEHICLE)
()

> (send FENDER is-subpart? AUTOMOBILE)
SUBPRT.69

> (send BODY is-subpart? VEHICLE)
()

> (send Honda03-Body is-part? Honda03)
PART.86

> (send Honda03-Fender is-part? Honda03-Body)
```

```
PART.89

> (send Honda03-Body is-part? Cadillac06)
()
```

An extension is an ODM component for maintaining instances. When a new intension is defined, a corresponding extension is created by the system. Instances are *members* of an extension and extensions are *subextensions* (subsets) of a corresponding generalization. For example, in Figure 6.11, *Honda.03* is a member of the extension of Hondas, and the extension of Hondas is a subextension of the automobile extension. ODM queries and responses in Dialogue 6.2 show these relationships. In the ODM prototype, extension names are assigned by the system and consist of the intension name and a unique integer separated by a dot, ".".

Dialogue 6.2 OML dialogue querying extensions

```
> (send HONDA get-extension)
HONDA.60

> (send AUTOMOBILE get-extension)
AUTOMOBILE.57

> (send FENDER get-extension)
FENDER.70

> (send db is-extension? HONDA)
()

> (send db is-extension? Honda.60)
T

> (send HONDA.60 get-members)
(HONDA03)

> (send AUTOMOBILE.57 get-members)
()

> (send AUTOMOBILE.57 get-all-members)
(CADILLAC06 HONDA03)

> (send Honda03 get-extension)
```

```
HONDA.60

> (send Honda03 get-all-extensions)
(HONDA.60 AUTOMOBILE.57 VEHICLE.50)

> (send VEHICLE.50 get-subextensions)
(AUTOMOBILE.57 MOTOR-HOME.52)

> (send VEHICLE.50 get-all-subextensions)
(AUTOMOBILE.57 MOTOR-HOME.52 CADILLAC.63 HONDA.60)

> (send Honda.60 get-superextensions)
(AUTOMOBILE.57)

> (send Honda.60 get-all-superextensions)
(AUTOMOBILE.57 VEHICLE.50)

> (send Motor-Home08 is-member? MOTOR-HOME.52)
MEMB.75

> (send Motor-Home08 is-member? VEHICLE.50)
SUBEXTEN.54

> (send Motor-Home08 is-member? AUTOMOBILE.57)
()

> (send HONDA.60 is-subextension? VEHICLE.50)
SUBEXTEN.59
```

Properties and their slots also retain information about intensions and instances. The object entry language, OEL, includes syntax for property specification when an intension is defined. OEL supports value assignments for two of four property slots: *p-lambda* and *p-units*. The other slots, *p-description* and *p-cardinality* must be set using the object-oriented command language, OML. New properties can be added to an intension at any time. OML syntax for defining new properties, and setting and retrieving the value of property slots is given below:

(send <intension> def-property <property-name>)
(send <intension> get-property-slot <property-name> <slot-name>)
(send <intension> set-property-slot <property-name> <slot-name> <slot-value>)

The only relevant slot of an *instance* property is the *p-value* slot, therefore, setting and retrieving *p-value* of an instance is an OML primitive operation.

(send <instance> get-property-value <property-name>)
(send <instance> set-property-value <property-name> <property-value>)

Dialogue 6.3 presents a session with the ODM prototype showing examples of property definitions, queries, and modifications.

Dialogue 6.3 OML dialogue querying properties

```
> (send VEHICLE def-property weight)
WEIGHT.93

> (send VEHICLE get-properties)
(WEIGHT)

> (send MOTOR-HOME get-properties)
()

> (send MOTOR-HOME get-all-properties)
(WEIGHT)

> (send Honda03 get-all-properties)
(WEIGHT)

> (send Honda03 is-property? weight)
WEIGHT.93

> (send Honda03 is-property? color)
()

> (send DWELLING def-property color)
COLOR.94

> (send DWELLING set-property-slot color p-lambda
        (lambda (x) (memq? x (red blue
                              green white brown black))))
(Procedure 20)

> (send DWELLING get-property-slot color p-lambda)
(LAMBDA (X) (MEMQ? X (QUOTE (RED BLUE GREEN
                              WHITE BROWN BLACK))))

> (send Motor-Home08 get-property-slot color p-lambda)
(LAMBDA (X) (MEMQ? X (QUOTE (RED BLUE GREEN WHITE
                              BROWN BLACK))))

> (send Motor-Home08 set-property-value color yellow)
```

```
** Error: YELLOW -- not a legal value

> (send Motor-Home08 set-property-value color red)
RED

> (send Motor-Home08 get-property-value color)
RED

> (send MOTOR-HOME get-all-instances-where color red)
(MOTOR-HOME08)

> (send VEHICLE set-property-slot weight p-units pounds)
POUNDS

> (send VEHICLE set-property-slot weight p-description
        "the weight of a vehicle")
"the weight of a vehicle"

> (send VEHICLE set-property-slot weight p-lambda
        (lambda (x) (< x 10000)))
{Procedure 21}

> (send AUTOMOBILE is-property? weight)
WEIGHT.93

> (send AUTOMOBILE set-property-slot weight p-lambda
        (lambda (x) (< x 5000)))
{Procedure 22}

> (send Motor-Home08 set-property-value weight 5000)
5000

> (send Motor-Home08 set-property-value weight 7000)
7000

> (send Honda03 is-property? weight)
WEIGHT.99

> (send Honda03 set-property-value weight 5000)


** Error: 5000 -- not a legal value

> (send Honda03 set-property-value weight 3000)
3000

> (send Honda03 get-property-value weight)
3000

> (send Motor-Home08 get-property-value weight)
7000
```

Another facility, useful for design and manufacturing data, is retrieval of instances based on the qualification of its property values. For example, a request for all red vehicles, is expressed by the following OML command:

(send vehicle get-all-instances-where color red)

In this example, ODM theorems infer that every motor home, automobile, Honda, and Cadillac is also a vehicle. The qualification and selection of ''*vehicles*'' is therefore based on derivable facts not explicitly represented. An extended version of the above statement permits qualification over any number of properties. The basic selection capability, excluding the inferencing mechanisms, corresponds directly to the *selection* operation in relational algebra. However, extended relational models which support complex hierarchical objects [Plo84, Sto84], cannot recursively perform selections over hierarchically organized relations. Furthermore, with the object-oriented schema representations described in section 6.4, it is possible to qualify over instances of any combination of intensions. In a relational model, this capability corresponds to second order selection over relations. By viewing relational schemata as *meta-data*, these facilities are now being introduced for extending the semantic modeling power of relational models [Sto84].

The following discussion presents facilities for creating and maintaining domain-specific relationships in ODM. Below is an example of the OEL specification for defining the relationship ''*inside*'' between the body and engine of an automobile.

(r inside
inner-component: |AUTOMOBILE-ENGINE|
outer-component: |AUTOMOBILE-BODY|)

ODM relations are n-ary; the number of *roles*, such as *inner-component* and *outer-component*, is not limited. Roles behave similarly to properties of intensions. In the above example, the role specification of *inner-component*, namely, the intension *AUTOMOBILE-ENGINE*; limits the value of that role to an instance of the *AUTOMOBILE-ENGINE* intension. Role specifications, however, are not limited to intensions; any data type specification in Figure 6.13 is applicable for a role value specification in a relationship.

Defining instances of a relation object utilizes the same OEL syntax as the definition of instances of an intension. Below I define an *inside* relationship between *Honda-Engine.03* and *Honda-Body.03*.

(i inside
inner-component: Honda-Engine.03
outer-component: Honda-Body.03)

The ODM system assigns a unique relation identifier to each instance of a relation. This identification *key* is used for accessing specific relational instances. A relation identifier corresponds to a *tuple-id*, a proposed tuple component in extended relational DBMS [Lor82, Gut82]. Examples of ODM's relational facilities are demonstrated in Dialogue 6.4.

Dialogue 6.4 OML dialogue defining ODM relations

```
> (send db def-relation-intension inside
                   inner-component outer-component)
INSIDE

> (send inside set-argument-lambda inner-component
        (lambda (x) (memq? x (send engine
                          get-all-instantiations))))
```

```
{Procedure 24}

> (send inside set-argument-lambda outer-component
        (lambda (x) (memq? x (send body
                               get-all-instantiations))))
{Procedure 25}

> (send inside def-relation-instance)
INSIDE.110

> (send inside.110 set-argument-value inner-component
                                        Honda03-Engine)
HONDA03-ENGINE

> (send inside.110 set-argument-value outer-component
                                            Cadillac06)


** Error: CADILLAC06 -- not a legal value

> (send inside.110 set-argument-value outer-component
                                          Honda03-Body)
HONDA03-BODY

> (send inside get-arguments)
(INNER-COMPONENT OUTER-COMPONENT)

> (send inside get-argument-lambda inner-component)
(LAMBDA (X) (MEMQ? X (SEND (QUOTE ENGINE)
                           (QUOTE GET-ALL-INSTANTIATIONS))))

> (send inside get-instantiations)
(INSIDE.110)

> (send inside.110 get-argument-value inner-component)
HONDA03-ENGINE

> (send inside.110 get-argument-value outer-component)
HONDA03-BODY
```

Dialogue 6.5 presents a final interactive session, based on the network of Figure 6.11 augmented with data defined throughout this section. These additional OML statements are used for *read-only* access, and produce formated output of ODM entity descriptions.

Dialogue 6.5 OML dialogue displaying formated output

```
> (send VEHICLE show-self)
```

```
VEHICLE
    WEIGHT

> (send VEHICLE show-self-in-detail)
VEHICLE
                                      WEIGHT
      P-DESCRIPTION: the weight of a vehicle
      P-UNITS: POUNDS
      P-PROC: {Procedure 21}
      P-LAMBDA: (LAMBDA (X) (< X 10000))
      P-NAME: WEIGHT

> (send Motor-Home08 show-self)
MOTOR-HOME08
    COLOR: RED
    WEIGHT: 7000 POUNDS

> (send CADILLAC show-self)
CADILLAC

> (send MOTOR-HOME show-property color)
MOTOR-HOME
    COLOR
      P-PROC: {Procedure 20}
      P-LAMBDA: (LAMBDA (X) (MEMQ? X (QUOTE (RED BLUE
                              GREEN WHITE BROWN BLACK))))
      P-NAME: COLOR

> (send HONDA show-property weight)
HONDA
    WEIGHT
      P-DESCRIPTION: the weight of a vehicle
      P-UNITS: POUNDS
      P-PROC: {Procedure 22}
      P-LAMBDA: (LAMBDA (X) (< X 5000))
      P-NAME: WEIGHT

> (send Honda03 show-self)
HONDA03
    WEIGHT: 3000 POUNDS

> (send Honda03 show-self-in-detail)
HONDA03
    WEIGHT: 3000 POUNDS

> (send Honda03 show-property-value weight)
3000 POUNDS
```

The syntax of OML is derived from a general purpose object-oriented programming language. As a result, it was not fine-tuned to provide *user-friendly* facilities for manipulating ODM objects. Although OML is a functionally complete language, many commands are exceedingly general and verbose. Additional work on data manipulation languages and user interfaces would improve the interactive language capabilities of the ODM prototype.

A graphical language is another research direction for data manipulation in ODM. Bit-map display facilities would permit data manipulation using windows, menus, and pointing devices. Labeled icons and links, similar to those presented in previous figures, would represent intensions, instances, and relationships. Research efforts by [Wel79, Eco83, Kin86, Nas78, Wel76, Ito] are experimenting with graphical interfaces for data management systems.

With interactive display facilities, users could graphically navigate through aggregation and generalization hierarchies. Graphical operations would correspond to those functional capabilities of OML presented above. ***Panning*** a window would display different portions of a network. An operation like ***zooming*** would enable a user to look *inside* an object node to view property descriptions, values, and other information. I envision graphical displays of modeling domains resembling Figures 6.6 and 6.11. A graphical interface is another step toward closing the gap between a conceptual model of an application, (frequently presented graphically), and its corresponding logical model. Although the implementation of a graphical interface was not pursued for this ODM prototype, I believe it would enhance user interactivity and understandability of schema and data objects modeled in the underlying ODM.

6.3 Semantic constraint management

Facilities for constraint maintenance have been previously introduced under the guise of *property value specification.* In ODM, constraint processing is supported by value restrictions on properties of intensions. Because an intension has no inherent semantics, constraints are not associated with intensions, but rather with its corresponding properties. Semantic constraints keyed on properties and relevant slots, further extend the semantics of an intension by adding more information to its properties.

Before discussing aspects of *semantic* constraints, I show how ODM maintains typical validity and consistency constraints supported by generalized DBMS. In previous sections, examples of validity constraints, such as value ranges and data types were described. Figure 6.13 shows the types of validity specifications permitted in OEL. In addition to the basic types: numeric, integer, real, literal, string; a value can be an element of a fixed set of values; or a list of items, where each item is an element of a set. For numerical constraints, it is also possible to define a range of values, or upper and lower limits for property values. The specifications shown in Figure 6.13 list only those structures and types built into OEL. Complex heterogeneous data types can be constructed using OML.

Maintaining consistency in conventional DBMS usually refers to structural constraints on DBMS relationships. Intension, relation, and instance names are symbolic pointers to data structures; therefore, structural inconsistencies which occur in other data models do not arise in ODM. *Existence* constraints, such as those exemplified by "*child*" data in an employee data base, are implicitly maintained. If an employee resigns, the employee is removed from the data base, and the employee's children should also be deleted. In Figure 6.14, *MarySmith, JohnSmith,* and *DavidSmith* are instances, and therefore refer to auxiliary data structures representing these entities. If *MarySmith* is deleted from her employer's data base, all references to her child, *JohnSmith*, are also expunged. Deleting the data structure representing *JohnSmith*, is an implementation issue; however, as long as there are no other references to *JohnSmith*, he is no longer part of the data base. For practical reasons, if the entity *JohnSmith* cannot be accessed, it should be deleted and the storage reclaimed for other data objects.

```
MarySmith
        children:  (JohnSmith)


JohnSmith
        parents:  (MarySmith DavidSmith)
        year-born:  1975
```

Figure 6.14 ODM parent and child instances

Symbolic pointers for ODM's components are also advantageous for the specification of relation types. M:N relationships are notoriously troublesome in CODASYL network and IMS-like hierarchical models [Dat81, Car79, Enc83]. ODM, like the relational model, represents M:N relationships implicitly. The *TEACHER-STUDENT* relationship in Figure 6.15 shows the relational schema and data representing this M:N relationship. Figure 6.16 presents the corresponding ODM relation and instance definition expressed in OEL. Roles specified as ODM intensions, such as *FACULTY-MEMBER* and *REGISTERED-STUDENT*, are analogous to *domains* of a relational model.

TEACHER-STUDENT (teacher, student)
where domain of "teacher" is "FACULTY-MEMBER"
and domain of "student" is "REGISTERED-STUDENT"

teacher	student
Einstein	McBride
Einstein	Sheldon
Feynman	Sheldon
VonNeuman	McBride
VonNeuman	Lohman

Figure 6.15 M:N relations in the relational model

In Chapter 3, I described some capabilities of a semantic constraint facility unavailable in conventional DBMS. I emphasized that semantic integrity constraints maintain the consistency of the world being modeled, in addition to maintaining the integrity of data instances in a computer representation. In a CAD/CAM environment, maintaining design consistency requires design-specific knowledge. Because data models and corresponding DBMS implementations do not include domain knowledge; they must provide data base designers, the DBA, and data base users with tools for adding relevant knowledge supporting semantic integrity management. In ODM, I did not

```
( r TEACHER-STUDENT
      teacher: |FACULTY-MEMBER|
      student: |REGISTERED-STUDENT|)


( i TEACHER-STUDENT
      teacher: Einstein
      student: McBride)

( i TEACHER-STUDENT
      teacher: Einstein
      student: Sheldon)

( i TEACHER-STUDENT
      teacher: Feynman
      student: Sheldon)

( i TEACHER-STUDENT
      teacher: VonNeumann
      student: McBride)

( i TEACHER-STUDENT
      teacher: VonNeumann
      student: Lohman)
```

Figure 6.16 M:N relations in ODM

develop a high-level language or interface for expressing domain knowledge in the form of constraints; rather, I relied on OML routines augmented by procedures expressed in the underlying implementation language. These general purpose facilities permit experimentation without limitations imposed by a particular constraint language. Through repeated experimentation and analysis of semantic constraints in a specific domain, such as CAD/CAM, patterns of use will emerge. Design of a user-oriented constraint language is then appropriate. The examples discussed below, are intended to demonstrate the power of the facility, not the simplicity or ease of expressing constraints. An example of a semantic constraint introduced earlier is the following equality:

feed-rate = 2 *(spindle-speed) (feed)*

This equation relates three properties of the intension, *SHEET-METAL-FABRICATION-PROCESS*, defined below:

(c SHEET-METAL-FABRICATION-PROCESS
apt-program: S
tool: S
tool-diameter: N
feed: N ipt
cutting-speed: N rpm
spindle-speed: N rpm
feed-rate: N ipm)

The following OML code assigns the *p-lambda* slot of the property *feed-rate* to adhere to the above constraint:

```
(send SHEET-METAL-FABRICATION-PROCESS set-property-slot feed-rate
    p-lambda
    (lambda (x self)
      (if (and (send self get-property-value spindle-speed)
          (send self get-property-value feed))
               then (equal? x
                         (times 2
                              (send self get-property-value spindle-speed)
                              (send self get-property-value feed))))))
```

This lambda expression first determines if values for *spindle-speed* and *feed* have been assigned, and if so, the constraint equation is verified. In OML commands, "*self*", refers to the instance whose property is being assigned. Dialogue 6.6 presents a session with the ODM system illustrating constraint enforcement. In this example, *self* is bound to *Bracket-Sheet-Metal-Fabrication-Process*, an instance of *SHEET-METAL-FABRICATION-PROCESS*. In Dialogue 6.6, the first value assigned to *feed-rate* is rejected because it does not fulfill the equality; the second value, 2.292, is accepted.

Dialogue 6.6 OML dialogue checking semantic constraints

```
> (send Bracket-Sheet-Metal-Fabrication-Process set-property-value
        spindle-speed 573)
573


> (send Bracket-Sheet-Metal-Fabrication-Process set-property-value
        feed .002)
0.002


> (send Bracket-Sheet-Metal-Fabrication-Process show-self)
BRACKET-SHEET-METAL-FABRICATION-PROCESS
    SPINDLE-SPEED: 573 RPM
    FEED: 0.002 IPT


> (send Bracket-Sheet-Metal-Fabrication-Process set-property-value
        feed-rate 4.32)

** Error: 4.32 -- not a legal value

>(send Bracket-Sheet-Metal-Fabrication-Process set-property-value
        feed-rate 2.292)
2.292


> (send Bracket-Sheet-Metal-Fabrication-Process show-self)
BRACKET-SHEET-METAL-FABRICATION-PROCESS
    SPINDLE-SPEED: 573 RPM
    FEED: 0.002 IPT
    FEED-RATE: 2.292 IPM
```

Verifying a semantic relationship such as, *is-orthogonal-to*, requires a procedural definition of *orthogonal*. This definition would be the basis for *p-lambda* slots of relevant properties. Although there is an initial cost for generating procedural constraint code; over time, libraries of validation procedures could greatly benefit design and manufacturing processes.

Incremental consistency checking, one of two maintenance options discussed in section 3.3, is supported in the ODM prototype. If a constraint has been defined and an unacceptable value is subsequently entered; the new value is rejected. However, if the constraint illustrated in Dialogue 6.6, is changed to *feed-rate = 3 (spindle-speed) (feed)*, the current value (now invalid) does not trigger a constraint violation. Only new values of *feed-rate* must conform to the new constraint in effect.

Retroactive consistency checking, although computationally expensive, is beneficial when a design is tentatively complete. If retroactive checking is enabled, old property values, invalidated by a new constraint, trigger violation conditions. One technique for reducing the overhead of retroactive checking is to allow the user to control the amount of checking by specifying portions of a generalization or aggregation network to be verified.

The types of constraints which can be expressed in the ODM prototype, surpass those in existing DBMS and CAD/CAM data management systems. In Chapter 8, I show how these constraint capabilities are utilized to encode CAD/CAM domain knowledge.

6.4 Dynamic schema facilities

Current data management methodologies force data base designers, DBAs, and users, to maintain a genuine separation between *schema* and *data*. In many design environments, especially mechanical design, it is sometimes difficult to identify whether a design represents a schema structure or data instance. For example, the representation of a *leading edge assembly* of an aircraft wing is an *instance* structure in conventional BOM data bases. The same structure, however, is a *schema* for different models and variations of aircraft wing designs. Modeling researchers have begun to question if this separation is warranted. In many knowledge representation and knowledge base management systems, the distinction between schema and data is starting to fade.

ODM's dynamic schema creates and maintains schema structures in the same way that data instances are managed. As discussed earlier, useful applications for dynamic schemata are those where the structure of the representation is defined as the data is generated. *Active* schema [Mai84, Bro84], which can be queried but not modified, benefit domains where the structure of the data is not uniform across data instances and many different structural representations are required. Access and retrieval facilities for schema are necessary to help locate, define, and control data instances. Although ODM differentiates between intensions and instances, the model provides capabilities for intension manipulation analogous to those available for instance processing. In Appendix B, most OML commands apply to both intensions and instances.

In ODM, new intensions and relationships can be added to the data base at any time. Intensions are added independently, or as a leaf node in a generalization or aggregation hierarchy. If a new intension is added to a generalization network, it inherits those properties of its generalizations. If an intension is added to an aggregation network, it becomes a subpart of any ancestors in its aggregation hierarchy. New properties can also be added for existing intensions, however, only subsequently created instances will recognize the new properties; other instances assume a null value for the new property. Similarly, if property slots are modified, only subsequent instances will conform to the new slot values. In the rest of this section, I present an object-oriented methodology underlying these dynamic schema capabilities.

Supporting a dynamic schema is analogous to adding an extended data type to a large programming system. In this task, a programmer must gather static information such as interrelationships between the new structure and existing data types. The programmer must also analyze program code to determine where and how instances of the new data type should be created and referenced. These tasks are error prone because (1) data structures are not always described properly and consistently and (2) all occurrences and interrelationships with other portions of the code are not always recognized. Many factors cause these deficiencies including: the size of the programming system, lack of documentation, the complex nature of structures and relationships, and a potentially large amount of program code which will be effected. This scenario closely resembles the modification of a conventional DBMS data dictionary. Both environments lack a critical tool: a system for managing information about data structures. Instead of requiring the user, ie., programmer or DBA, to manually maintain data structure representations, we should instead supply the system with knowledge about its representations, and let the system use this knowledge to construct and manage the representations. For ODM dynamic schema, a *meta-data* management system utilizes this information for adding new schema entities such as: *intensions, relation schemata*, and *properties*. Notice that operations on meta-data closely correspond to DML (data manipulation language) facilities available for processing instance data.

ODM is implemented in an object-oriented programming language; therefore, I constructed objects representing ODM's primitive entities, such as entities named ''*intension*'', ''*relation*'', and ''*instance*''. Figure 6.17 shows the generalization hierarchy representing these generic objects. Two main entities are *object* and *relation*. A relation includes both the builtin relationships such as subpart and specialization; and user defined relationships. Knowledge incorporated in this network takes the form of messages and corresponding methods. Upon receipt of a message, an entity responds according to the method prescribed by the transmitted message. I have incorporated into ODM primitives, knowledge about how they should respond to messages sent by an ODM user. Therefore, defining a new intension in ODM, such as *VEHICLE*, corresponds to adding a new instance, named *VEHICLE* to the generic entity, *intension*. Each ODM entity in Figure 6.17 has methods for maintaining the structure of its instances. For example, the *VEHICLE* entity, retains its own data base consisting of information such as: the relationships it has with other entities; a list of its own ODM instances, ie. *Vehicle.01, Vehicle.02*; the properties which are associated with it; and bookkeeping information. If a user adds a new property to an intension, such as adding *interior-size* to the *VEHICLE* intension; the underlying management system knows which intensions are specializations of *VEHICLE* and therefore are affected. Relevant methods modify the instances of *VEHICLE* accordingly. OML commands which query an intension object, do so by retrieving its corresponding ODM instance and recalling its attributes. This *meta-level* data management system maintains the organization of those structures normally regarded as schema or meta-data representations.

Three instances of related work in the fields of operating systems, expert systems, and DBMS implementation utilize an object-oriented representation for maintaining meta-data about a computational task. In [Sno83], Snodgrass describes *Cola*, an object-oriented command language for a capability-based

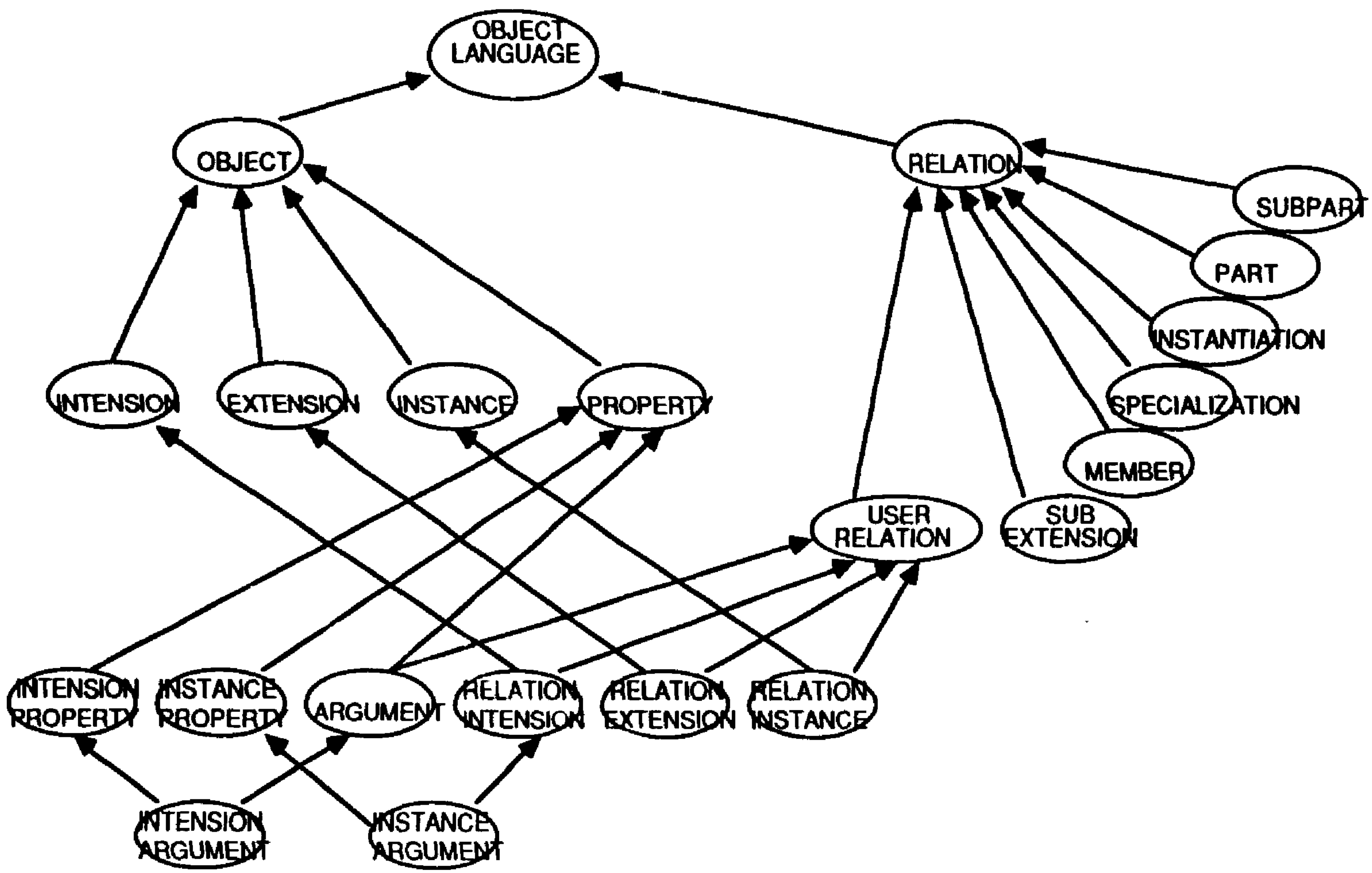


Figure 6.17 Generalization network supporting dynamic schemata

operating system. Cola was designed to effect a correspondence between capabilities in the operating system, and objects supported by the command language. Cola, based on Smalltalk, uses standard message-passing as a control mechanism and its objects are arranged hierarchically for responding to operating systems commands.

Davis [Dav78] adopts a similar approach in his work on knowledge acquisition in rule-based systems. He uses a taxonomic organization to maintain knowledge about representations for expert system construction and maintenance. Davis cites two major contributions of a generalization hierarchy for meta-data management. First, the hierarchy presents a global organization of representations in the system and offers a convenient overview of them. Second, the system uses this information as a tool, allowing an expert to teach an expert system about new instances of conceptual primitives.

An object-oriented approach to database system implementation is addressed by Baroody and DeWitt [DeW81]. Their object-oriented representation encapsulates correspondences between data base entities and relationships. They have demonstrated that the object-oriented approach has advantages of data independence, run-time efficiency, and support for low-level views. Each of these systems, including ODM, benefits from an object-oriented architecture by embedding knowledge about representations and relationships for automatically maintaining meta-level structural information.

6.5 ODM prototype implementation

ODM is implemented in the T programming language [Ree82]. T is a lexically scoped dialect of Lisp, developed at Yale University and used by the Yale Cognitive Science research group. The ODM software system currently operates on two hardware configurations in UCLA's Computer Science Department: the CECS (Center for Experimental Computer Science) Locus network of Vax hardware and a network of Apollo workstations. In addition, the prototype software has been ported to other hardware supporting the T language.

I adopted a layer approach for the ODM prototype implementation. Figure 6.18 shows the hierarchical nature of the software subsystems. Teebert, the bottom layer, is a general-purpose object-oriented programming language which I implemented in T. Teebert is a subset of Ross and Bert [McA85], object-oriented simulation languages developed at The Rand Corporation. Teebert's message-passing form of procedure invocation resembles facilities in Flavors [Obj84], Strobe [Smi84], and Smalltalk [God82]. ODM's processing routines are implemented in Teebert and classes in Teebert correspond to ODM primitives. These classes form the basis of the schema management facilities discussed in the preceding section. Using Teebert, I constructed the higher-level language, OML. As I have shown in previous examples, OML commands manipulate domain-specific objects and relationships in ODM. The complete OML syntax is found in Appendix B.

OEL, an object entry specification language, is an independent software module whose input is a list of new intensions, relations, and instances. Parsing OEL input produces equivalent OML language statements which are subse-

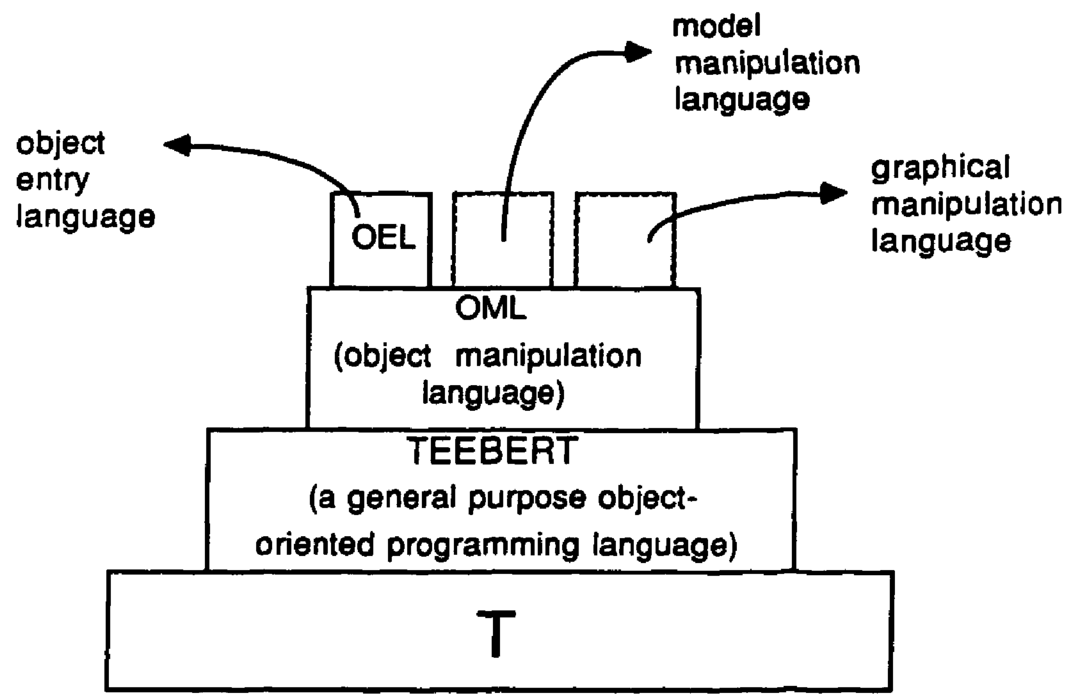


Figure 6.18 ODM software architecture

quently entered as OML commands. Examples of OEL input were presented in Figure 6.12. OEL is used strictly for data entry, including creation of intensions, relations, and instances. As an extension to the current ODM prototype, I propose two additional user interfaces. First, a *model manipulation language* specifically suited for manipulating ODM entities. Before such a language is designed, however, research should be conducted to determine which interactive language facilities are most beneficial. I also recommend a two-dimensional user interface for graphically interacting with data entities. As I discussed earlier, a graphical language and two-dimensional displays, correspond most closely to the data base design process and promote a better understanding of objects and relationships being represented.

Data and program abstraction were the main motivations for utilizing a layer approach. Each level in the hierarchy of Figure 6.18 hides lower level details through its independent language for communication with higher layers. For example, the implementation of Teebert uses vectors for storing properties of objects. Converting to a different data structuring mechanism, such as association lists, only requires modification to Teebert's creation and access functions to manipulate association lists instead of vectors. Because ODM functions are written in Teebert, no changes to the ODM system code are required for shifting from vectors to association lists. Similarly, if a Lisp implementation is desired, it is only necessary to replace T syntax with Lisp syntax in the Teebert language. None of the higher layers use T directly; instead, they communicate in Teebert.

The ODM software system operationalizes four desirable data management functions presented in Chapter 3. Although this implementation is neither fast enough nor robust enough to be considered a true prototype, both of these problems could be overcome if the system were reimplemented. In Chapter 8, I evaluate ODM's performance for achieving the objectives of integrated CAD/CAM data management.

CHAPTER 7

REVIEW OF CAD/CAM DBMS PROJECTS

Many efforts are underway for developing DBMS better suited to the management of CAD/CAM data. The focus of these projects depends heavily on whether the work is sponsored by corporate or research funds. In this chapter, I identify successful projects in each sector which have the greatest potential for industry acceptance.

7.1 Corporate CAD/CAM DBMS projects

Corporate endeavors are mainly directed toward one aspect of conceptual centralization: *the integration of application data and subsystems*. Many progressive industries are already using CAD/CAM tools for design, manufacturing, and assembly. They are recognizing the detrimental effects of many self-contained, independent data bases requiring specialized data input and output. Other corporations are seeing a multitude of data files being generated and experiencing a loss of control over the data. Major industrial CAD/CAM DBMS efforts are generally long-term projects, estimated to require between 10 and 15 years. The mandate for most of these projects is to develop an operational integrated DBMS system and adhere to a plan for converting to the new system. Because of the duration of these projects and the expensive conversion efforts involved, most systems being designed are extensions or variations of conventional DBMS.

An integrated information system at Ingersol Milling Machine [Hes83] is hailed as a great success from manufacturers within and outside the corporation. Their information systems were rewritten to support the installation of a company-wide integrated management and business information system, MIS/BIS, based on IDMS [IDM]. They have cited a reduction in design staff maintenance effort from 57% to 18% of their time. Although their MIS/BIS system contains data for master scheduling, inventory control, purchasing and accounts payable, it does not include engineering design and parts manufacture data, which are generated and maintained by the graphical subsystems. Alphanumeric output from graphics systems is fed indirectly into the MIS/BIS system.

At Boeing, a major effort in progress aims to produce the Boeing Computing Support System (BCSS) [BCS83]. Streamlining CAD/CAM product definition and fabrication processing is the main corporate objective of the project. Data management goals are (1) to provide a common data management and networking facility for all Boeing applications and (2) to integrate the graphics workstation environment and large-scale company database. This program has been in the planning stages since 1980, and it is projected that implementation and conversion will be completed in 1995. BCSS will integrate product definition data, such as two-dimensional and three-dimensional geometry; product properties; bill of material information; job and process specifications; tool definition; and inspection and testing sub-systems.

Tornado [Ulf82a] is a DBMS developed in Norway at the Central Institute for Industrial Research. The first version of Tornado was developed in 1978 to fulfill application requirements of Autokon, the world's most popular ship design system. Tornado, installed at about 20 sites in Europe and the United States, is a CODASYL-like network system especially suitable for complex network data structures. Current work is focused on integrating Tornado with GPM (Geometric Product Model), a CAD project developing a solid modeling system for sculptured surfaces.

Because corporate manufacturing centers cannot interrupt normal activities to spend years researching and experimenting; their efforts, naturally, are more conservative. Their goal is to make effective use of existing data management tools, and focus on the integration of data and applications as a key to increased productivity. I devote the rest of this chapter to *research* efforts in the area of CAD/CAM data management systems and generalized DBMS. The projects I discuss below are not burdened by the totality of design, development, implementation, and conversion efforts required in private industry. Therefore, these projects are dedicated to a number of interesting DBMS challenges.

7.2 CAD/CAM DBMS research efforts

The entity or object-oriented model of data organization has gained general acceptance for CAD, CAM, and engineering data base applications [Bro84]. Unfortunately, this organization is orthogonal to the relational model, popular in recent years due to its simple *table* structures and data independence. Accordingly, major efforts at Berkeley and IBM San Jose have addressed the deficiencies of the relational model for representing object-oriented data. Both groups are developing extensions to their respective systems, Ingres and System R, to accommodate object-oriented data.

At IBM, Plouffe et al. [Plo84], have proposed two extensions to System R supporting object-oriented engineering and design: complex objects and long fields. *Complex objects* represent object hierarchies in a relational format. A complex object is a hierarchical cluster of tuples that comprise a single root tuple defining an object, and one or more dependent tuples describing the object. This extension entails the use of two reserved column types, *IDENTIFIER*, for uniquely identifying tuples; and *COMPONENT-OF*, to indicate which tuples are related. Although the hierarchical nature of objects is captured in this fashion, complex objects are limited to strict hierarchies; networks of tuples are not allowed. In practice, this restriction severly affects inventory and BOM applications where a detailed part is a component of many assemblies. System R's *long fields* are a special kind of heterogeneous data type. This extended feature supports physical storage and retrieval of long unformated items such as raster images or large matrices, but does not specifically address graphical or geometrical data.

Ingres extensions [Sto84] also fulfill the need for hierarchies of complex objects. The approach taken by Stonebraker et al. is to consider a complex object as a collection of tuples which is materialized during query processing. This approach supports commands in the query language as a data type in the DBMS. Another Ingres extension includes a *transitive closure* operator which can be appended to specific query operators. This operator concatenation indicates that the operation should be continued as long as new tuples are generated; thereby, simulating a transitive closure generator. Although the functionality of new Ingres and System R features is desirable, these techniques only partially camouflage the underlying relational structure. They widen, rather than reduce, the gap between logical and conceptual models of CAD/CAM applications.

CAD/CAM DBMS researchers at CCA (Computer Corporation of America) cite aspects of conceptual centralization as their main goal [Bro84]. Components of their CAD/CAM DBMS (CCDBMS) architecture contributing to conceptual centralization are (1) a user interface to provide uniform access to all CCDBMS facilities, (2) a global data manager to handle distributed processing, and (3) a global view of all data needed for queries, distributed processing, and configuration mangement. CCDBMS uses the functional data model Daplex [Shi81] which provides high-level set-oriented operations, permits modeling of complex objects, and supports *is-a* hierarchies. The conceptual model under development consists of information about parts and related documents, such as drawings, specifications, and change notices. Extensions to this model are also being investigated to include manufacturing data, for instance, group technology and process planning data; and analysis data such as finite element models. They have considered adding special facilities for transitive closure operations, currently a complicated Daplex procedure. Additional extensions may include *parts* hierarchies for robust BOM processing, and long term plans address the definition, update, and browsing of local and global schemata.

Development of the Semantic Association Model, SAM*, is in progress at the University of Florida [Su86]. SAM* focuses on CAD/CAM applications and has identified some of the same weaknesses and proposed similar functionality as my research on ODM. However, Su has achieved these objectives using different strategies. SAM* is based on a semantic network model and recognizes seven distinguished relationships or *associations* between objects or nodes in a network. Below I outline five of the associations which are relevant to modeling CAD/CAM data. Although Su references *nodes* and *node clusters* as objects and entities, SAM* is not object-oriented. Few facilities identify or access clusters of nodes comprising an object. Emphasis is placed on the following associations. *Membership* denotes the set theoretic relationship *is-element-of* discussed in Chapter 5. A second SAM* relationship, *aggregation* is based on [Smi77] and is used to construct entities by aggregating sets of attributes. In object-oriented terminology, this interpretation of aggregation refers to the object/property structure of entities. *Generalization* relationships in SAM* allow nodes to be grouped together to form a more general concept node, facilitating attribute inheritance. Although the association called *composition* theoretically reflects the "*contains*" relationship; the semantics of composition associations does not include BOM composition hierarchies. Instead, Su uses this association for version control and to relate multiple data files comprising an entire data base. *Interaction* associations relate to domain relationships and are viewed as relationship sets similar to Chen's E-R model. Facilities for representing and validating semantic constraints is one obvious omission is the SAM* model.

Many efforts have addressed data base management in other design domains, such as electronics and architecture. In general, methodologies for VLSI (Very Large Scale Integration), PCB (Printed Circuit Board), and PWB (Printed Wiring Board) design are better defined than mechanical engineering and manufacturing methodologies. Building blocks for electronics products and corresponding composition rules are more uniform and fixed than features of a manufactured part or mechanical assembly. However, Katz [Kat85] still describes electronics design as a "tentative and iterative ... process" requiring hierarchical object organizations and dynamic schemata. At the University of Southern California, an object-oriented approach for VLSI/CAD, 3DIS, focuses directly on VLSI design methodology [Afs85]. Afsarmanesh et al. have extended a VLSI design environment to capture the underlying semantics of circuit structure and behavior. This methodology and the accompanying environment supports the view that design engineers, who are normally not data base experts, nevertheless become designers, manipulators, and evolvers of their data bases. 3DIS incorporates a geometric model and supports entities, events, operations, and descriptions of meta-data as objects. The VHSIC (Very High Speed Integrated Circuits) program supported by the US Department of Defense is outlining specifications for a VHDL (VHSIC hardware description language). These efforts are also trying to promote integration of electronic design and data management. Eastman [Eas78] discusses data base capabilities in general design activities but notes that manufacturing applications in the areas of aircraft, spacecraft, and shipbuilding differ from electronics design in the customization of a major assembly. In architecture applications, he emphasizes the need for many levels of consistency constraints. Eastman proposes an entity-oriented organization characterized by *spatial* and *composition* hierarchies. These hierarchies combined with aggregation abstractions aid in sophisticated semantic integrity maintenance.

The systems and projects discussed so far, focus directly on the management of CAD, CAM, or engineering data. Many of the CAD/CAM DBMS goals, similar to those presented in Chapter 2, were formulated by an analysis of the application domain. However, generalized DBMS and data management models are also being influenced by Artificial Intelligence (AI), specifically knowledge representation. AI researchers are discovering that DBMS based on existing data models, do not have sufficient functionality for maintaining AI applications data. Work on Knowledge Base Management Systems (KBMS) is beginning to address some of these limitations. The dynamic and semantic nature of CAD/CAM data requires capabilities very similar to those of KBMS. Below I discuss KBMS work related to semantic representations, dynamic schemata, and semantic constraint management.

Smalltalk is the basis of a set theoretic data model developed by [Cop84]. This work demonstrates how features of Smalltalk, such as operational semantics, type hierarchies, and entity identity, solve many problems which arise when using commercial DBMS for managing AI application data. Sembase, derived from a semantic model [Kin86], has shown that semantic modeling can be transformed from an abstract design tool into an effective data management tool. King cites three advantages of semantic models over hierarchical, network, and relational models. First, a data base can be viewed as a collection of abstract objects, instead of a set of flat tables or files. Second, aggregation and generalization can be built into the model, and third, a semantic schema more easily captures integrity constraints. Once the schema dictionary is constructed, Sembase's dictionary facility provides operators for perusing a schema but not modifying it.

Research on active and dynamic schema facilities is addressed by those working on data dictionary systems. Although there has been great promise in the data dictionary as a tool for managing information resources; in practice, data dictionaries have failed to achieve that promise. Curtice [Cur81] predicts that data dictionaries will be undergoing major change during the years to come.

He expects that eventually there will be no distinction between the DBMS and the data dictionary. The Database Directions III Workshop report [Gof82] recommends that future data dictionaries offer facilities to (1) make meta-data more accessible to users, and (2) allow meta-data to be queried and manipulated in the same manner as application data. Some relational systems treat meta-data and data equally, and relational operations produce meta-data as well as data. But meta-data in network and hierarchical systems is quite limited. With the notable exception of SPIRES [Sch75], most other systems that support a rich variety of meta-data do so with separate and less flexible facilities. McCarthy [McC82] has found that scientific and statistical data bases share the need for integrated meta-data management. He has proposed four general goals of meta-data management: *integration, standardization, simplicity,* and *extensibility.* Data base designers, administrators, and users should be able to add new types and structures of meta-data; and add and revise meta-data values quickly and easily, without needing to reload or redefine existing structures.

Accurate modeling of an application often involves constraints beyond those captured by conventional schemata. Semantic constraints define consistency by capturing the behavior of the application. The approach taken by Morgenstern [Mor86] is based on *constraint equations.* A declarative language expresses invariant relationships which must hold among specified data objects. Declarative constraint equations have an executable interpretation; they can be compiled directly into routines for automatic maintenance of the constraints. This approach contrasts with writing procedural code for maintaining the constraints. Shepard and Kerschberg [She84] have developed a knowledge base management system, PRISM, for semantic integrity specification and enforcement in data base systems. PRISM employs a rule-based constraint language, CL. A constraint specified in CL is a collection of rules where each rule consists of a precondition, action, and postcondition sequence. Within each precondition and postcondition, predicates are combined with logical operators *AND, OR, NOT* and parentheses. To determine whether a constraint is satisfied, its logical value is computed to *TRUE, FALSE, UNKNOWN*, or *EXCEPTION*. In both of these systems, constraints are viewed as independent entities of the data management system instead of being associated with particular objects or attributes. Other projects offer constraint primitives within the data model representation [Ham81] or utilize semantic nets and attached procedures [Myl80] for semantic constraint management.

In Figure 7.1, I present a summary of the projects discussed in this chapter. In this summary I consider systems focusing primarily on CAD/CAM DBMS facilities. For instance, electronic CAD DBMS, such as work by Katz and McLeod, are not included. Also, generalized DBMS or KBMS, such as Shepard's PRISM system, are not listed. For any specific system, the capabilities indicated are those currently in design or development phases. Although CCA cites future plans to add aggregation hierarchies, that feature is not a primary goal. Also, in some systems, only specialized versions of a capability are supported. For example, System R's extensions support long data items, however, a general facility for heterogeneous data types is not available.

goals/capabilities of the system	ODM	MIS/BIS Ingersol	BCSS Boeing	CCDBMS CCA	Tornado CIIR	INGRES Stonebrakr	SYSTEM R Lorie	SAM* Su	GLIDE Eastman
integration of graphical, geometric, design, and mfg. data	X	X	X	X	X				X
heterogeneous data types	X			X		X		X	X
complex, hierarchical objects	X			X	X	X	X	X	X
generalization hierarchies	X			X				X	
aggregation hierarchies	X							X	X
transitive closure operations	X					X	X		
semantic constraints	X								X
dynamic schema	X								
conceptual model	O-O with relations	network	entity-relation	object-oriented	network	relational	relational	semantic network	object-oriented
logical model	object-oriented	network	hierarch-ical	functional	network	relational	relational	network	network

Figure 7.1 Summary of CAD/CAM DBMS projects

CHAPTER 8

EVALUATION AND VALIDATION

In this chapter I evaluate the results of this research and demonstrate the advantages offered by ODM. The analyses I present below are based on two application data bases from Hughes Electro-Optical and Data Systems Group. One data base supports the *PWA* (Printed Wiring Assembly) application at Hughes; the other data base contains *part definition data*, utilized for testing a Hughes expert system generating *Producibility Feedback* (PF) [Zuc86]. For each application, I first present the content and organization of the Hughes data bases,[1] followed by a discussion of the design of corresponding ODM data bases. Examples extracted from the ODM data bases, and dialogues interacting with the ODM software system, illustrate the use of ODM features for achieving the goals of integrated CAD/CAM data bases presented in Chapter 2.

Validation is a certification process assuring that the stated goals of the research have been achieved. The final section of this chapter discusses the methods I adopted for validating the ODM prototype software.

---

[1]The values of data items in the Hughes data bases have been altered to preserve the confidentiality and proprietary nature of this information.

8.1 Hughes PWA application

At Hughes, PWA manufacturing is one of the most automated applications. PWAs are designed on Computervision CAD systems, and process plans for assembling the components are computer generated. The HICLASS (Hughes Integrated Classification) system, an AI expert system shell developed in-house, supports many PWA manufacturing processes [Liu]. The manual assembly of a PWA is guided by a sophisticated color graphics system. Personnel manipulate and assemble boards and components with hand-held tools and devices; therefore, their hands are not available for keyboard or mouse input. Instead, a user interacts with the graphical assembly instructions by foot-controlled pedals located underneath the graphics workstation. The integration of many PWA subsystems has eliminated manual translation and transfer of documents, thereby, minimizing production time. Hughes officials claim that the flow of paperwork has been reduced by nearly three-fourths, from an average of 160 hours to 40-70 hours [DMD86].

8.1.1 PWA data bases and file systems

PWAs are referenced by their assembly number. The data for a specific PWA resides in two sets of files: *transfer* data, and *IGES* (International Graphics Exchange Specification) graphical data. Transfer data refers to six independent files describing the components contained in the finished PWA, including hardware, fasteners, and wires. These files include bill of material data, physical characteristics of components, electrical characteristics of components, characteristics of components subject to automated testing, reference information, and general notes. The files are designated as *transfer* files because they follow the development of a PWA through its manufacturing cycle; thus, they are *transferred* from design through manufacturing. Appendices D through I contain the six transfer files for PWA M87706172, displayed in Figure 8.1. Examples discussed in the rest of this section will refer to data for PWA M87706172.

IGES data consists of four or more files representing the graphical characteristics of a PWA. One file contains graphical data for the outline of a bare PWA board, and three files represent graphical data corresponding to three orthographic views. The four required files represent *geometry* entities; optional IGES files include *annotation* and *structure* entities [Ini83]. Appendix J presents data for the board outline of PWA M87706172, and Appendix K illustrates a segment of the file representing the top view of PWA M87706172.

The ten files outlined above comprise the data base for a single PWA and are generated whenever a new PWA is designed. In addition to these PWA-specific files, four *MCL* (Master Component Library) files are a vital part of the Hughes PWA application system. MCL data contains basic physical, electrical, and structural properties of all components and assemblies. Data is extracted from these master files and utilized for constructing new PWA transfer files. Appendices L through O show portions of the four MCL files.

8.1.2 PWA conversion to ODM

ODM evaluation entails two independent investigations. First, I demonstrate below that ODM is comparable in power to Hughes DBMS facilities for maintaining PWA data. The second analysis, presented in the following sections, exhibits improvements in PWA data management by adopting an ODM

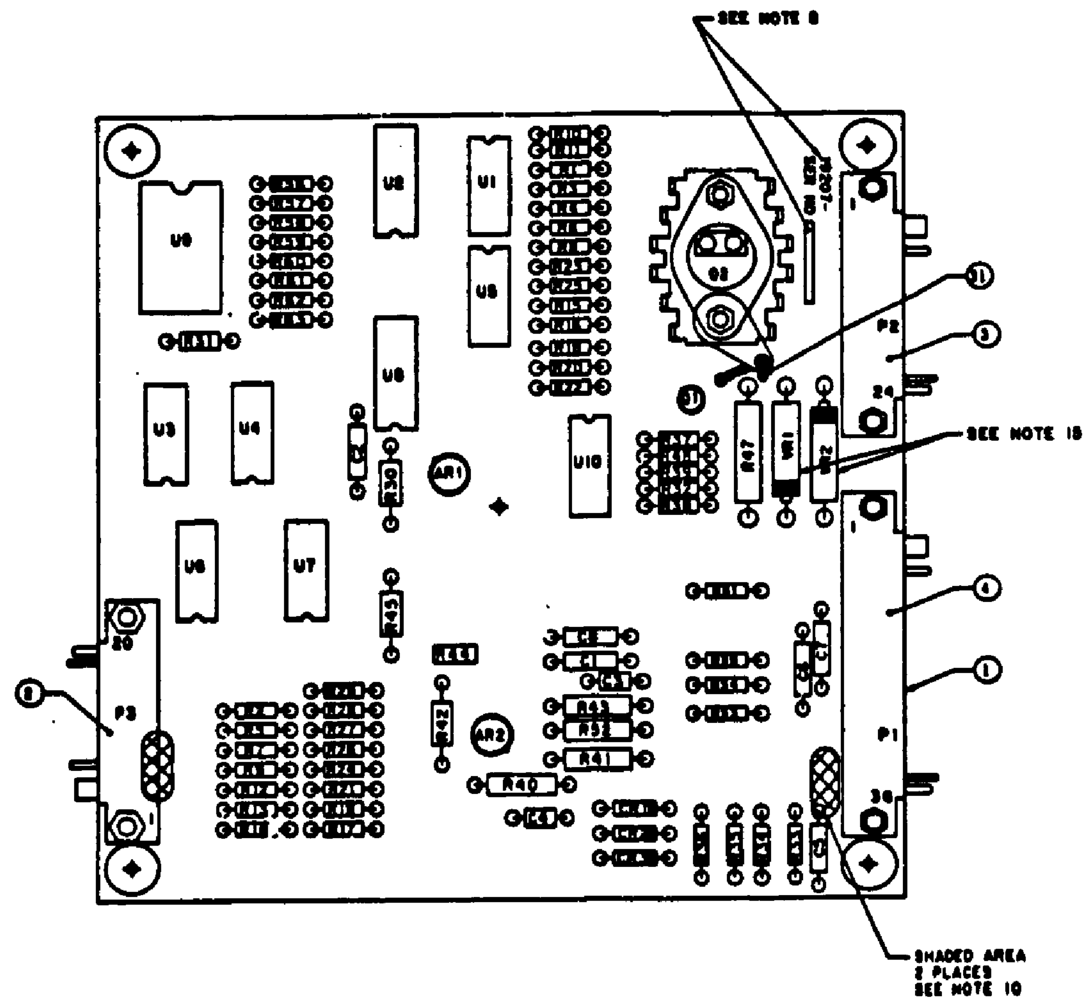


Figure 8.1 PWA M87706172

model and exercising the unique capabilities ODM provides.

Figure 8.2 shows the PWA data sets described above. IGES data files are generated by Computervision CAD systems and are only utilized for graphical display. No data management system is associated with IGES files. MCL and transfer data are managed by the relational DBMS, Oracle.[1] To compare the functionality of Oracle [Ora79] to ODM's facilities, I constructed ODM data bases corresponding to Oracle relations. One technique for modeling a relation in ODM is to create analogous intensions with properties. Therefore, I generated an ODM intension for each Oracle relation; and attributes of the relation were converted to ODM properties. This conversion reflects a simple one-to-one correspondence between Oracle relations and ODM intensions. The original relation name was retained as the intension name, and tuples of the relation became instances of the intension. I generated one piece of additional structure, the instance name, which is constructed from key attribute values of a tuple. Data type information and domain requirements are encoded as constraints on ODM property values. Figure 8.3 shows the Oracle schema of four MCL relations. In Figure 8.4, the corresponding ODM intensions are illustrated as OEL (Object Entry Language) specifications.

The purpose of this exercise is to demonstrate that ODM structures are equivalent in expressive power to existing PWA relations. The majority of data manipulation in PWA data bases demands query processing; therefore, I measure expressive power in terms of query facilities. Because a one-to-one

[1]When these analyses were conducted at Hughes, DBMS conversion to Oracle was underway. Since then, Hughes has discontinued its use of Oracle due to unreliable performance.

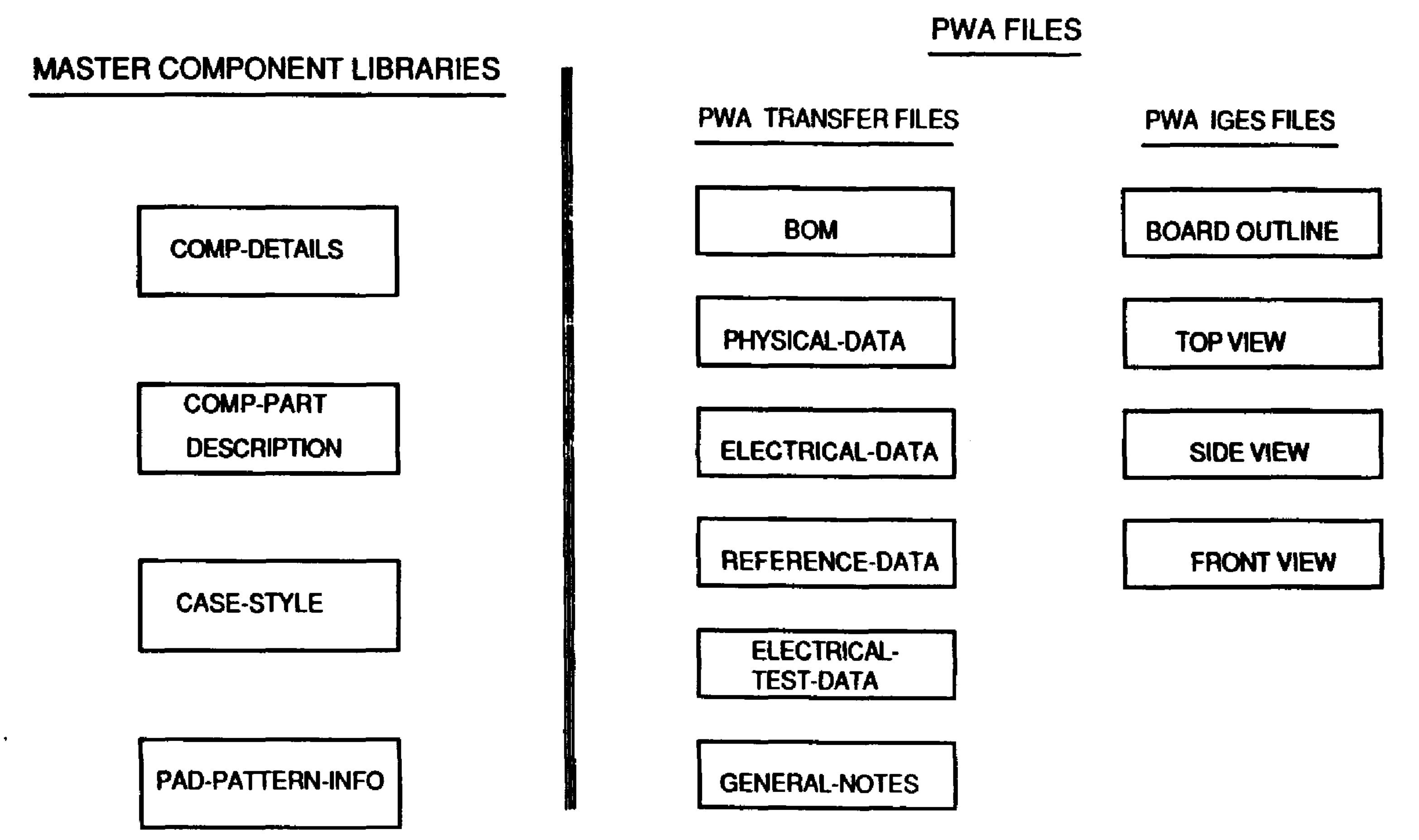


Figure 8.2 PWA file organization

COMP_DETAILS:

NAME	TYPE	WIDTH
COMP_PART_NUM *	char	50
STYLE_CODE	char	20
COMP_WEIGHT	numeric	
COMP_WEIGHT_UOM	char	20
LEAD_MATERIAL	char	50
MAX_NON_OPRING_TEMP	numeric	
COMP_TEST_ID	char	20
POLARITY	numeric	
COMP_VALUE	char	20
VALUE_UOM	char	20
TOLERANCE_PLUS	numeric	
TOLERANCE_MINUS	numeric	
POWER_RATING	numeric	
SEQUENCE_ID	numeric	
PAD_PATTERN_NUM	numeric	
STATIC_SENSITIVE	char	20

COMP_PART_DESC:

NAME	TYPE	WIDTH
COMP_PART_NUM *	char	50
COMP_STATUS	char	20
COMP_GROUP	char	20
COMP_TYPE	char	20
COMP_SPEC	char	50
COMP_DESC	char	50

CASE_STYLE:

NAME	TYPE	WIDTH
STYLE_CODE *	char	20
X_OFFSET	numeric	
Y_OFFSET	numeric	
COMP_MIN_LENGTH	numeric	
COMP_NOM_LENGTH	numeric	
COMP_MAX_LENGTH	numeric	
COMP_MIN_WIDTH	numeric	
COMP_NOM_WIDTH	numeric	
COMP_MAX_WIDTH	numeric	
COMP_MIN_HEIGHT	numeric	
COMP_NOM_HEIGHT	numeric	
COMP_MAX_HEIGHT	numeric	
NOM_LEAD_DIAMETER	numeric	
SHAPE	char	50
NO_OF_PINS	numeric	

PAD_PATTERN_INFO:

NAME	TYPE	WIDTH
PAD_PATTERN_NUM *	numeric	
PAD_SIZE	numeric	
PAD_SPAN	numeric	
DELTA_X	numeric	
DELTA_Y	numeric	

Figure 8.3 Oracle MCL schmata

```
(c component-detail-rec
   comp-part-num: L
   style-code: L
   comp-weight: N
   comp-weight-uom: L
   lead-material: L
   max-non-opring-temp: N
   comp-test-id: L
   polarity: N
   comp-value: L
   value-uom: L
   tolerance-plus: N
   tolerance-minus: N
   power-rating: N
   sequence-id: N
   pad-pattern-num: N
   static-sensitive: L)

(c component-description-rec
   comp-part-num: L
   comp-status: L
   comp-group: L
   comp-type: L
   comp-spec: L
   comp-desc: L)

(c case-style-rec
   style-code: L
   x-offset: N
   y-offset: N
   comp-min-length: N
   comp-nom-length: N
   comp-max-length: N
   comp-min-width: N
   comp-nom-width: N
   comp-max-width: N
   comp-min-height: N
   comp-nom-height: N
   comp-max-height: N
   nom-lead-diam: N
   shape: L
   no-of-pins: N)

(c pad-pattern-info-rec
   pad-pattern-info: N
   pad-size: N
   pad-span: N
   delta-x: N
   delta-y: N)
```

Figure 8.4 Intensions representing MCL schemata correspondence exists between the structure of Oracle relations and ODM intensions, I claim that any data accessed by a relational query can also be retrieved by an OML (Object Manipulation Language) command. Similarly, the creation of new relations and tuples parallels OEL commands to add new intensions and instances. Although this analysis has demonstrated comparable representation models, none of the unique ODM features are shown; ODM is merely imitating a relational model.

The organization of PWA data at Hughes is non-optimal. Data is unnaturally distributed among MCL and transfer files; furthermore, an inordinate amount of data duplication is evidenced. For the studies discussed below, I restructured PWA data to promote more effective data management practices. With these redesigned data bases, I illustrate the benefits of ODM by reviewing CAD/CAM DBMS goals, highlighting ODM features which support the goal, and current deficiencies which have been overcome.

8.1.2.1 Conceptually centralized PWA files

In Chapter 2, I presented a primary motivation for this work: the need for *integrated* CAD/CAM DBMS. A major obstacle toward integration is the distribution of data across many independent files and data bases. With the advent of powerful microprocessors, these self-contained data bases which, until recently, were retained on a single computer, are now physically and geographically distributed among numerous machines. So far, local area networks have widened, instead of reduced, the conceptual gap between multiple data sources.

ODM helps overcome these gaps in three ways. First, ODM networks promote the construction of *directory* structures to identify and access application data files. The functionality of a directory data base resembles capabilities provided by an operating system for file management. Second, ODM supports heterogeneous complex data types permitting file names, hardware devices, access procedures, and network protocols to be entered into the directory as data. Finally, ODM directories allow *gradual* conversion to a totally integrated DBMS. Developing a totally integrated system is a five to fifteen year effort; therefore, application systems and data bases cannot simply be taken off-line for redesign. With incremental conversion, the directory remains in tact while specific data bases and files are converted and reformated. DBAs at Hughes and Rockwell recommend directory data bases for streamlining data retrieval by initially locating data repositories.

To illustrate these advantages, I constructed a *directory* schema supporting Hughes PWA application data. Figure 8.5 shows the graphical ODM format of the schema; the corresponding OEL specification is given in Figure 8.6. Each intension defined in Figure 8.6 represents a file. *PWA-FILE* is the root intension and includes relevant file attributes such as *file-name, machine,* and *operating-system*. All other files (intensions) of the data base *inherit* these attributes. Figure 8.7 presents the directory schema instantiated with specific PWA files. In this example, the string *"M87706172"* is used to construct the names of specific files for PWA M87706172. File names for other PWAs are also instances of the intension *TRANSFER-FILE*. A directory organization for PWA application data supports queries such as:

What are the names of all transfer files for PWA <n>?

Who has authorized access to IGES files for PWA <n>?

What is the login-sequence for access to component-electrical-data of PWA <n>?

In each of these queries the main reference key is a PWA number. Hughes employees emphasized that 80% of all data retrieval is keyed on component or PWA number. A file-oriented directory *conceptually centralizes* data files so a user can determine where and how to access physically distributed files and data bases. The last query illustrated above begins to show the potential for incorporating procedural access to distributed data bases. In addition to providing data like ***login-sequence***, the directory could also provide procedures for querying specific data instances.

Figure 8.8 presents an ODM directory organization for managing IGES files and records. Many CAD/CAM industries and CAD/CAM system suppliers are being encouraged to provide IGES support for their graphical systems. IGES standards allow graphical data to be transported between different CAD systems. An IGES data set consists of five sections, each containing one or more records. To aid IGES data management, I generated an ODM directory schema such that each section is represented as an intension, and fields of different sections are denoted by attributes of the intensions. In ODM format IGES data is more comprehensible to users. Contrast the format of a standard IGES file (Appendices J and K) with ODM instances in Figure 8.9. For transferring IGES data from one graphics system to another, IGES standard format is required; therefore, I built automatic procedures to convert in both directions between ODM instances and IGES files.

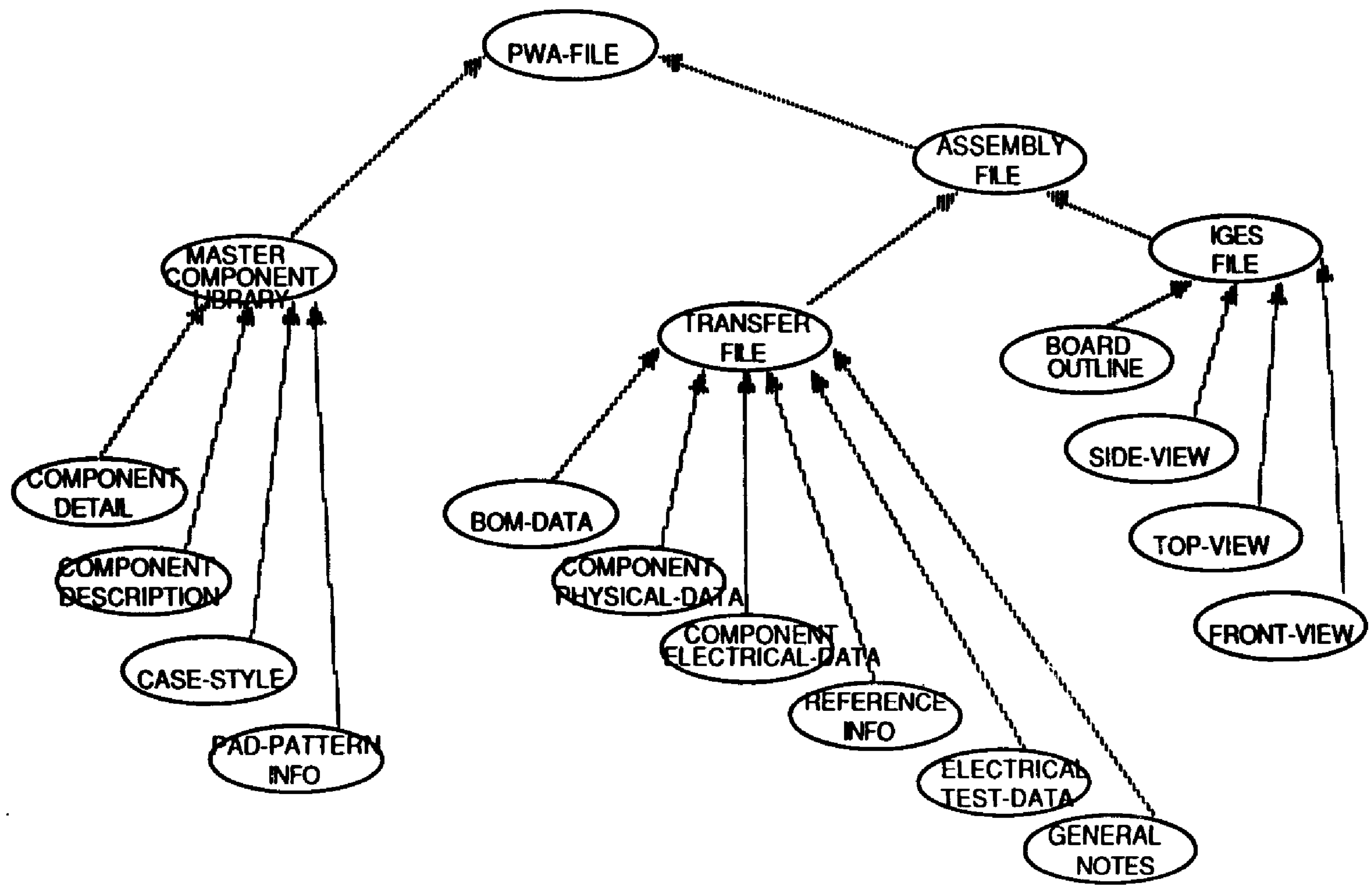


Figure 8.5 PWA directory hierarchy

```
(c pwa-file
   file-name: L
   machine: S
   operating-system: S
   system-account-id: L
   password: L
   access-code: L
   authorized-users: (list-of: L)
   access-procedures: T)

(c master-component-library |pwa-file|)

(c component-detail |master-component-library|)
(c component-description |master-component-library|)
(c case-style |master-component-library|)
(c pad-pattern-info |master-component-library|)

(c assembly-file |pwa-file|)

(c transfer-file |assembly-file|)

(c bom-data |transfer-file|)
(c component-physical-data |transfer-file|)
(c component-electrical-data |transfer-file|)
(c reference-info |transfer-file|)
(c electrical-test-data-info |transfer-file|)
(c general-notes |transfer-file|)

(c iges-file |assembly-file|)

(c board-outline |iges-file|)
(c orthographic-view |iges-file|)
(c mfg-process-view |iges-file|)

(c bareboard-data |assembly-file|)
```

Figure 8.6 Intensions representing PWA directory

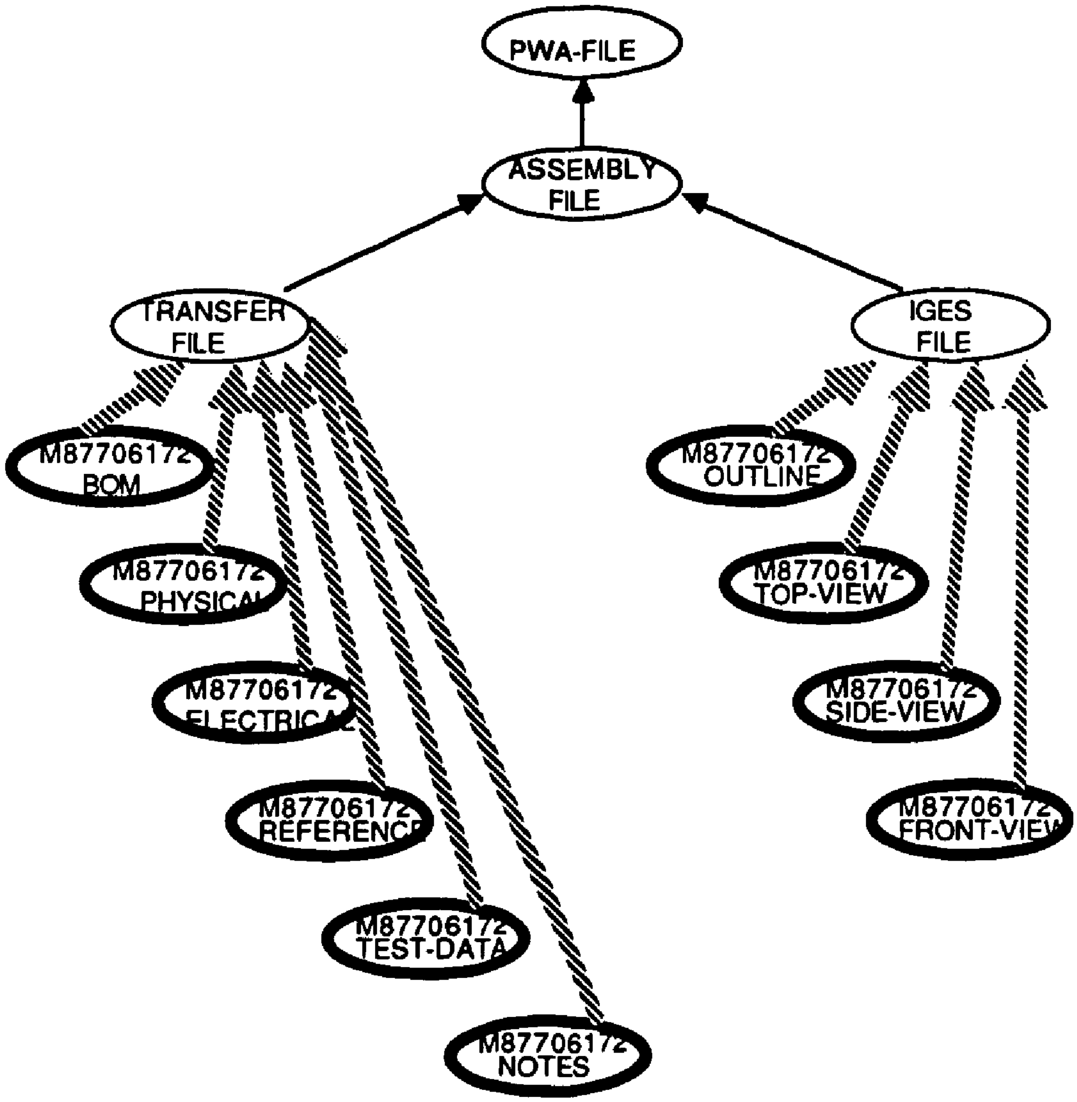


Figure 8.7 PWA directory instances

```
(c iges-file
   start-section: |start-section|
   global-section: |global-section|
   directory-section: |directory-section|
   parameter-section: |parameter-section|
   terminate-section: |terminate-section|)

(c global-section
   field-delimiter: S
   end-delimiter: S
   sending-system-product-id: S
   file-name: S
   system-id: S
   iges-translator-version: S
   integer-bits: I
   receiving-system-product-id: S
   definition-space-scale: R
   unit-flag: I
   maximum-line-weight: R
   size-of-maximum-line-width: R
   file-generation-date-time: S
   minimum-resolution: I
   definition-space-size: I
   organization: S )

(c directory-record-id /directory-section/
   parameter-record-id: L
   entity-type: I
   version: N
   line-font-pattern: N
   level: N
   view: |parameter-record-id|
   defining-matrix: |parameter-record-id|
   label-display: |parameter-record-id|
   line-weight: N
   pen-number: N
   parameter-record-count: I
   form-number: N
   entity-label: S
   entity-subscript: I)

(c parameter-record-id /parameter-section/
   directory-record-id: |directory-record-id|
   parameter-data: (list-of: T))
```

Figure 8.8 IGES intensions

```
(i iges-file-M87706172 |iges-file|
   start-section: start-section-M87706172
   global-section: global-section-M87706172
   directory-section: directory-section-M87706172
   parameter-section: parameter-section-M87706172
   terminate-section: terminate-section-M87706172)

(i start-section-M87706172 |start-section|
   textual-description: "board outline")

(i global-section-M87706172 |global-section|
   field-delimiter: ","
   end-delimiter: ";"
   file-name: "mfvs.3827.iges.outline"
   system-id: "computervision.rev 11.00.cadds"
   iges-translator-version: "iges rev 01.00"
   integer-bits: 16
   definition-space-scale: 201.8000
   unit-flag: 1
   file-generation-date-time: "831207, 94609"
   organization: "72-24-33")

(i directory-section-M87706172 |directory-section|)

(i directory-record-1 |directory-record-id|
                     /directory-section-M87706172/
   parameter-record-id: parameter-record-1
   entity-type: 124
   version: 1
   status: 0
   parameter-record-count: 9)

(i parameter-section-M87706172 |parameter-section|)

(i parameter-record-1  |parameter-record-id|
                     /parameter-section-M87706172/
   directory-record-id: directory-record-1
   parameter-data: (1.0 0.0 1.0 0.0 0.0 1.0 0.0))

(i terminate-section-M87706172 |terminate-section|
   number-directory-records: 10
   number-parameter-records: 10)
```

Figure 8.9 IGES instances

8.1.2.2 Component-oriented BOM hierarchies

During my site visits at Lockheed, Rockwell, and Hughes, I observed that a BOM hierarchy is the primary conceptual organization of design and manufacturing data. Unfortunately, existing DBMS do not directly support this organization. The second goal of CAD/CAM DBMS, exemplified below, is a data model facilitating natural BOM data management. Three ODM capabilities support this goal: an object-oriented representation paradigm, class/subclass generalizations, and part/subpart aggregations.

A BOM architecture allows the conceptual view of CAD/CAM data to be equated with the logical view represented by schemata descriptions. For example, Figure 8.10 illustrates the conceptual view of PWA data at Hughes. This conceptual hierarchy enables the physical structure of a PWA to be traced from assembly through components and hardware. Other DBMS efforts offering data abstraction hierarchies have resulted in modified relational models with tuple identifiers and repeating groups. Nevertheless, the underlying relational model remains inappropriate for an inherently entity-oriented application. An object-oriented representation allows direct access from PWAs to components and from components to corresponding assemblies. Each PWA is constructed from the following items: a bare board; components, such as capacitors and transistors; connectors, ie., cables and relays; and fasteners, like screws and nuts. Unfortunately, current PWA data base organization at Hughes distributes properties of assemblies, components, fasteners, and connectors throughout six transfer files and four MCL files, (see Figure 8.2). Retrieving data relevant to a particular component, for example, capacitor M60985/94-7380, necessitates access keyed on component M60985/94-7380 from seven different sources.

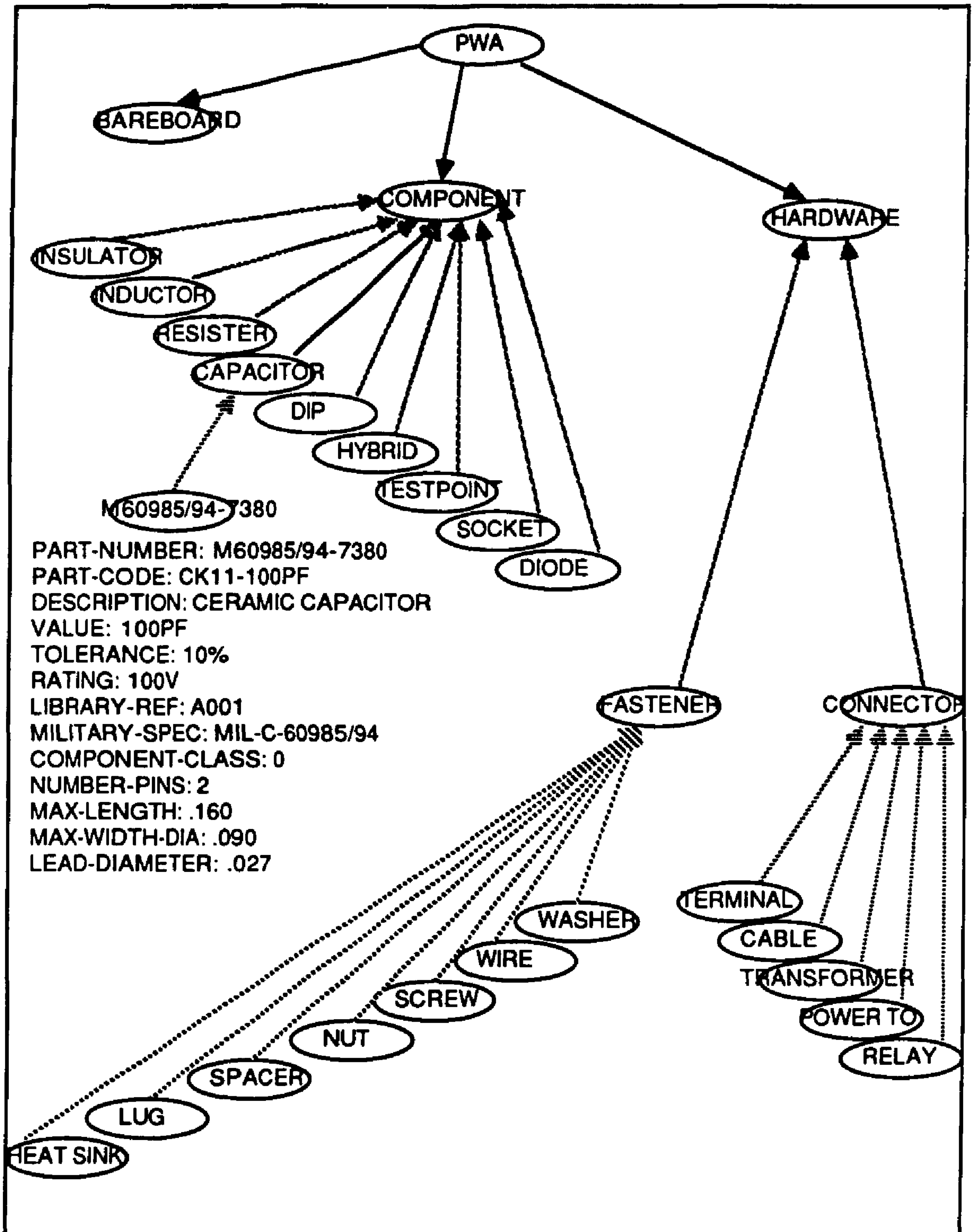


Figure 8.10 PWA conceptual schema

In ODM, I constructed aggregation and generalization networks directly reflecting the conceptual organization of Figure 8.10. This organization minimizes file and data base cross-references; any or all data referring to a given component, such as capacitor M60985/94-7380, may be retrieved through a single access to the intension representing capacitor M60985/94-7380. By querying the intension named *COMPONENT*, ODM generalization networks allow access to properties which *all* components share. Properties which are common to *one* type of component, such as capacitors, are accessed through the *CAPACITOR* intension. Properties relevant to a *specific* capacitor are associated with its intension object, as shown in Figure 8.10. Finally, properties pertaining to M60986/94-7380 as it relates to PWA M87706172, such as *x-offset* and *x-origin*, are retained with an instance object. Figure 8.11 shows the OEL specification of the ODM conceptual schema in Figure 8.10.

Another deficiency of Hughes PWA data is the overwhelming amount of data duplication. Figure 8.12 shows the relational attributes contained in four of the six PWA relations maintaining transfer data. Out of a total of 20 attributes, only two, *quantity* and *maximum-thickness*, are found in a single relation. Three of the attributes (excluding the key attribute *part number*) are duplicated in all four relations. Other cases of replicated data occur within each of the individual relations whenever a PWA contains more than one instance of a specific component. For example, PWA M87706172 contains three M60985/94-7380 capacitors. In the physical-data, electrical-data, and electrical-test-data relations, three instances of M60985/94-7380 are stored, however, only six of the 19 attributes differ across the three instances. Property values for *x-offset, y-offset, x-origin* and *y-origin* represent data related not only to the capacitor but also the

```
(c pwa)

 (c bareboard /pwa/)

 (c hardware /pwa/)

 (c component /pwa/
   part-number: L
   reference-desig: L
   x-origin: N
   y-origin: N
   x-offset: N
   y-offset: N
   orientation: N
   component-class: N
   number-pins: N
   library-ref: L
   max-length: N
   max-width-dia: N
   max-thick: N
   lead-diameter: N
   military-spec: L
   part-code: L
   description: S
   value: L
   tolerance: L
   rating: L)

 (c fastener |hardware|)

   (c heat-sink |fastener|)
   (c lug |fastener|)
   (c spacer |fastener|)
   (c nut |fastener|)
   (c screw |fastener|)
   (c wire |fastener|)
   (c washer |fastener|)

 (c connector |hardware|)

   (c relay |connector|)
   (c power-to |connector|)
   (c transformer |connector|)
   (c cable |connector|)
   (c terminal |connector|)

  (c insulator |component|)
  (c inductor |component|)
  (c resistor |component|)
  (c dip |component|)
  (c hybrid |component|)
  (c test-point |component|)
  (c socket |component|)
  (c diode |component|)
  (c capacitor |component| pins:
              (greater-than: 0))

  (c M60985/94-7380 |capacitor|
      part-number: M60985/94-7380
      part-code: ck11-100pf
      description: "ceramic"
      value: 100pf
      tolerance: 10%
      rating: 100v
      library-ref: a001
      military-spec: c-60985/94
      component-class: 0
      number-pins: 2
      max-length: .160
      max-width-dia: .090
      lead-diameter: .027)
```

Figure 8.11 PWA component intensions

complete PWA. Physical and structural attributes of the capacitor, such as *length* and *diameter*, apply to all M60985/94-7380 capacitors, and therefore, have identical values across all instances.

A primary problem associated with duplicated data (aside from the overhead of extra storage facilities) is maintaining consistency. In an extreme case, modifying the *lib reference* value of component M60985/94-7380 requires modifications to the four transfer relations for PWA M87706172. Furthermore, within each of three of those relations: *physical-data, electrical-data,* and *electrical-test-data*, three entries must be modified accordingly because PWA M87706172 contains three M60985/94-7380 capacitors. Similar modifications are also required for other PWAs containing capacitor M60985/94-7380.

Data abstraction facilities offered by generalization and aggregation networks, discussed in Chapters 5 and 6, minimize data duplication. Attribute values which are common to all instances of an intension are retained with the intension. Aggregation networks allow a component contained in many PWAs to be represented once and referenced by its instance name. Figure 8.13 presents an instance of PWA M87706172, and one of its components. The left side of Figure 8.13 is identical to one branch of the ODM hierarchy in Figure 8.10. Component attributes, such as *part-code* and *number-pins*, are only specified once for capacitor M60985/94-7380; however, they are distributed through *instantiation* to the three instances contained in PWA M87706172. In Figure 8.13, only component attributes related to PWA M87706172 are retained with instance M60985/94-7380-1. Modifications to intension attributes implicitly effect those attributes of instances. The modifications are also applied to all PWAs which contain the modified component. Figure 8.14 shows instance data

PWA relations	BOM	physical data	electrical data	electrical test data
component attribute				
qty	*			
+ part number	*	*	*	*
ref designator	*	*	*	*
x-org	*	*		
y-org	*	*		
x-offset	*	*		
y-offset	*	*		
orientation	*	*		
+ component cls	*	*		
+ lib reference	*	*	*	*
+ max-length	*	*		
+ width-dia	*	*		
+ lead-dia	*	*		
+ military spec	*	*	*	*
+ max-thickness		*		
+ part-code			*	*
+ description			*	*
+ value			*	*
+ tolerance			*	*
+ rating			*	*

* ==> attributes in the indicated relation
+ ==> attributes which are identical for the same components

Figure 8.12 Replication in PWA data

for the three M60985/94-7380 capacitors contained in an instance of PWA M87706172.

In this section I have described how the BOM organization, supported by ODM, improves *conceptual accessibility*. I redesigned the logical view of the PWA data bases to reflect the hierarchical conceptual schema. All component data related through the *contains* relationship is accessed directly from the PWA in which it is contained. Dialogue 8.1 shows a session using OML (Object Manipulation Language) commands based on the ODM schemata in Figures 8.10 and 8.13.

Dialogue 8.1 OML dialogue traversing PWA networks

```
> (send pwa get-subparts)
(COMPONENT HARDWARE BAREBOARD)

> (send hardware get-specializations)
(CONNECTOR FASTENER)

> (send connector get-specializations)
(TERMINAL CABLE TRANSFORMER POWER-TO RELAY)

> (send fastener get-specializations)
(WASHER WIRE SCREW NUT SPACER LUG HEAT-SINK)

> (send component get-specializations)
(CAPACITOR DIODE SOCKET TEST-POINT HYBRID DIP
                      RESISTOR INDUCTOR INSULATOR)

> (send component show-self)
COMPONENT
    PART-NUMBER
    REFERENCE-DESIG
    X-ORIGIN
    Y-ORIGIN
    X-OFFSET
    Y-OFFSET
    ORIENTATION
    COMPONENT-CLASS
    NUMBER-PINS
    LIBRARY-REF
    MAX-LENGTH
    MAX-WIDTH-DIA
```

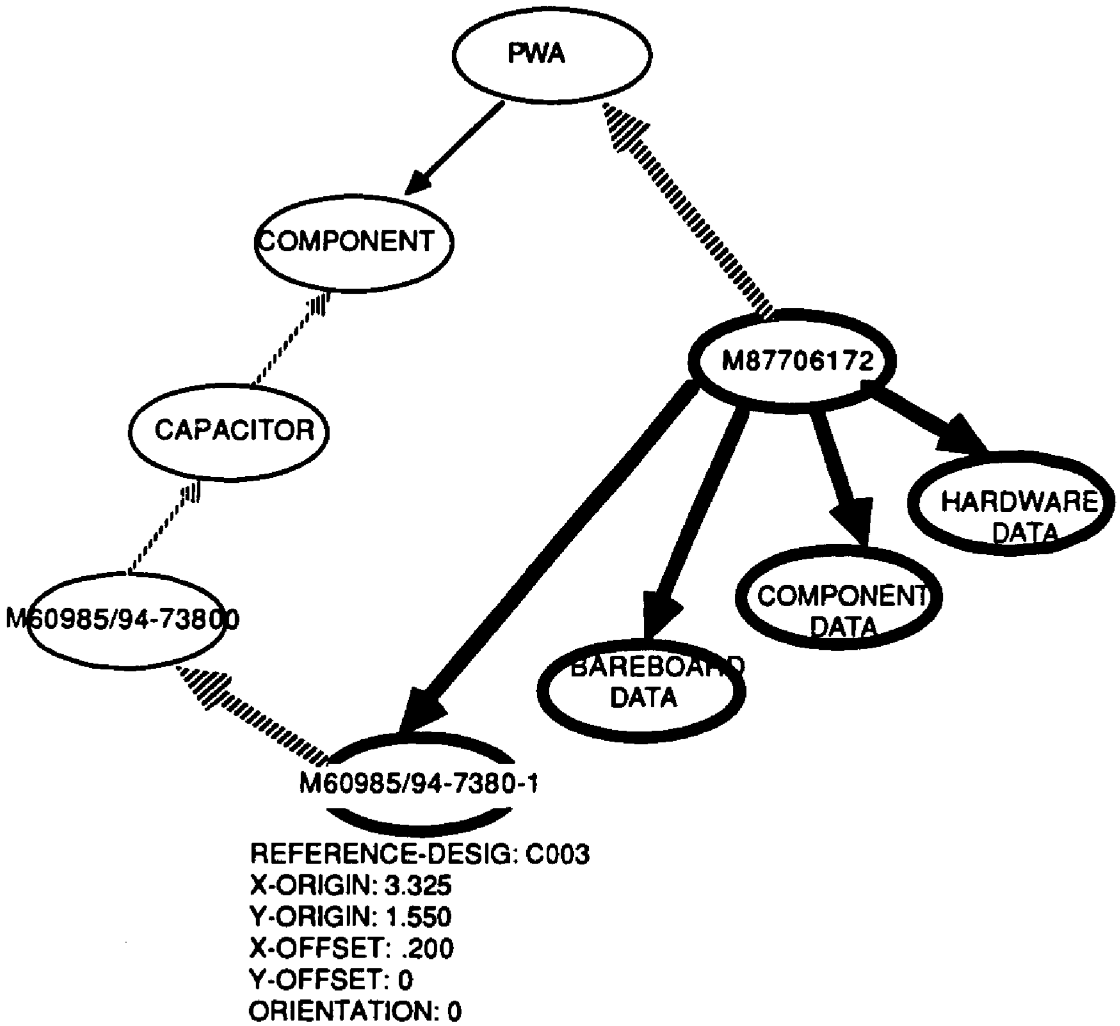


Figure 8.13 PWA component instances

```
(i M87706172 |pwa|)

(i M60985/94-7380-1 |M60985/94-7380| /M87706172/
   reference-desig: c003
   x-origin: 3.325
   y-origin: 1.550
   x-offset: .200
   y-offset: 0
   orientation: 0)

(i M60985/94-7380-2 |M60985/94-7380| /M87706172/
   reference-desig: c004
   x-origin: 2.750
   y-origin: .450
   x-offset: .250
   y-offset: 0
   orientation: 0)

(i M60985/94-7380-3 |M60985/94-7380| /M87706172/
   reference-desig: c008
   x-origin: 3.125
   y-origin: 1.900
   x-offset: .350
   y-offset: 0
   orientation: 0)
```

Figure 8.14 OEL specification of PWA instances

```
    MAX-THICK
    LEAD-DIAMETER
    MILITARY-SPEC
    PART-CODE
    DESCRIPTION
    VALUE
    TOLERANCE
    RATING

> (send capacitor show-self)
CAPACITOR
    NUMBER-PINS

> (send capacitor get-specializations)
(M60985/94-7380)

> (send M60985/94-7380 show-self)
M60985/94-7380
    PART-NUMBER
```

```
    PART-CODE
    DESCRIPTION
    VALUE
    TOLERANCE
    RATING
    LIBRARY-REF
    MILITARY-SPEC
    COMPONENT-CLASS
    NUMBER-PINS
    MAX-LENGTH
    MAX-WIDTH-DIA
    LEAD-DIAMETER

> (send M60985/94-7380 get-property-slot description p-value)
"ceramic capacitor"

> (send M60985/94-7380 get-property-slot number-pins p-value)
2

> (send M60985/94-7380 get-property-slot military-spec p-value)
MIL-C-60985/94

> (send pwa get-instantiations)
(M87706172)

> (send M87706172 get-parts)
(M60985/94-7380-3
 M60985/94-7380-2
 M60985/94-7380-1)

> (send M60985/94-7380-3 show-self)
M60985/94-7380-3
    REFERENCE-DESIG: C008
    X-ORIGIN: 3.125
    Y-ORIGIN: 1.9
    X-OFFSET: 0.35
    Y-OFFSET: 0
    ORIENTATION: 0

> (send M60985/94-7380-3 show-self-in-detail)
M60985/94-7380-3
    PART-NUMBER: M60985/94-7380
    REFERENCE-DESIG: C008
    X-ORIGIN: 3.125
    Y-ORIGIN: 1.9
    X-OFFSET: 0.35
    Y-OFFSET: 0
    ORIENTATION: 0
    COMPONENT-CLASS: 0
    NUMBER-PINS: 2
    LIBRARY-REF: A001
    MAX-LENGTH: 0.16
    MAX-WIDTH-DIA: 0.09
    MAX-THICK: ()
    LEAD-DIAMETER: 0.027
```

```
    MILITARY-SPEC: MIL-C-60985/94
    PART-CODE: CK11-100PF
    DESCRIPTION: ceramic capacitor
    VALUE: 100PF
    TOLERANCE: 10%
    RATING: 100V

> (send M60985/94-7380-2 show-self)
M60985/94-7380-2
    REFERENCE-DESIG: C004
    X-ORIGIN: 2.75
    Y-ORIGIN: 0.45
    X-OFFSET: 0.25
    Y-OFFSET: 0
    ORIENTATION: 0
```

8.1.2.3 Customized components and assemblies

DBMS schema facilities describing CAD/CAM data cannot represent semantic features, structure, or relationships. Semantic features such as *holes*, *flanges*, and *cutouts* are only represented graphically by entities like lines and circles. Structural relationships, for example, *orthogonal-to*, *on-top-of*, and *inside*, are not explicitly represented, although they are implicitly present in an engineering drawing, and are relationships which effect design and manufacturing processes. One reason existing DBMS cannot model these entities is because the extent of semantic data cannot be enumerated; semantic entities are not fixed across all parts and assemblies. Current schema definitions can only capture characteristics of parts and assemblies which are common to all instances being modeled. This limitation severely restricts the types of information which can be represented.

ODM's dynamic schema and hierarchical constraint management improves the flexibility and robustness of schema facilities. The goal of *customized representations* enables a designer to specify many individual semantic features of a product during the design stage. In most cases, designers know the relevant features and relationships necessary for future processing. Entering semantic information during product definition realizes three benefits. First, data generation for a new part is optimized. Extracting relevant data from the engineering drawing is a time consuming task usually requiring numerous iterations. In many CAD/CAM environments, new specialized data bases are created for each separate process, although the content of the data bases is similar. Data should be entered once and retained for use throughout part fabrication. A second benefit is the consistency which is promoted by interleaving design with data entry. If the data is generated at the same time the part is designed, the same information is maintained and referenced throughout the manufacturing cycle, in the same way that an engineering drawing is referenced. Currently, data used in different facets of production may be incompatible or contradictory. Modifications to semantic design data should require the same control which is enforced for changes to engineering drawings. Finally, with a dynamic schema, data base design efforts are reduced. Schema structures and constraints are added and modified dynamically, instead of incurring expensive reconfiguration costs for reformating a data base.

In the context of Hughes PWA application, customizing a PWA representation means that data bases of new PWAs can easily be generated by designers from existing component data bases. Let's assume a new PWA, say M9999, is being designed and contains capacitor M60985/94-7380 (which is also contained in PWA M87706172), Currently it is necessary to construct entries for PWA M9999 in four transfer relations, where the majority of data is identical to the values for instances of capacitor M60985/94-7380 in PWA M87706172. Data replication persists because capacitor M80985/94-7380 is defined as an instance of an MCL component relation rather than a schema definition. Instead, if capacitor M80985/94-7308 is regarded as an intension, as in Figure 8.10, then specific instances are created as components of the new PWA M9999. All generic properties of capacitor M80985/94-7308 are retained with the intension. Only seven properties (those without a "+" in Figure 8.12) pertain to instances and therefore only seven new pieces of data are entered for each capacitor contained in the new PWA M9999.

In Figure 8.10, the intension, *CAPACITOR*, maintains data shared by all capacitors. If a new capacitor is designed, a new specialization of *CAPACITOR* is created dynamically which inherits those properties and values common to all capacitors. Figure 8.15 depicts an ODM network with a new PWA, M9999, containing one capacitor M60985/94-7308-5, and one new capacitor, M99/99-99-1. OML commands creating the new PWA and capacitor are presented in Dialogue 8.2. Data underlying Dialogue 8.2 is based on the schema in Figure 8.10.

Dialogue 8.2 OML dialogue creating new PWA components

```
> (send pwa def-instance M9999)
M9999

> (send M60985/94-7380 def-instance M60985/94-7380-5)
M60985/94-7380-5

> (send M60985/94-7380-5 show-self-in-detail)
M60985/94-7380-5
    PART-NUMBER: M60985/94-7380
    REFERENCE-DESIG: ()
    X-ORIGIN: ()
    Y-ORIGIN: ()
    X-OFFSET: ()
    Y-OFFSET: ()
    ORIENTATION: ()
    COMPONENT-CLASS: 0
    NUMBER-PINS: 2
    LIBRARY-REF: A001
    MAX-LENGTH: 0.16
```

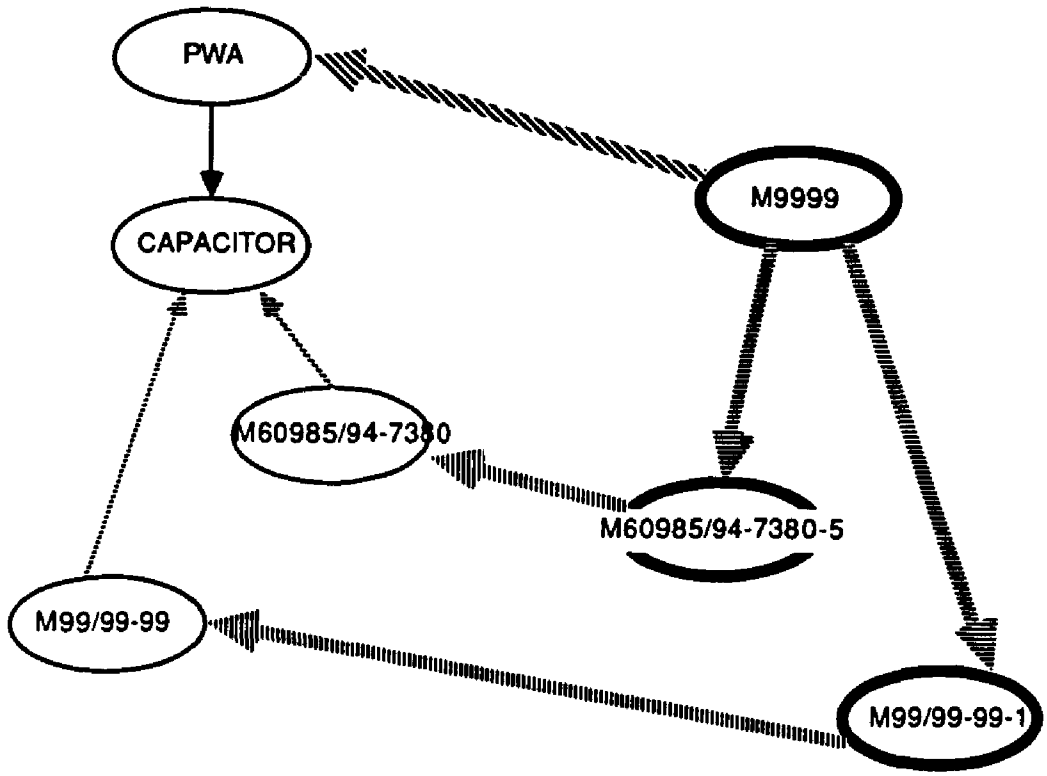


Figure 8.15 New PWA instances

```
    MAX-THICK: ()
    LEAD-DIAMETER: 0.027
    MILITARY-SPEC: MIL-C-60985/94
    PART-CODE: CK11-100PF
    DESCRIPTION: ceramic capacitor
    VALUE: 100PF
    TOLERANCE: 10%
    RATING: 100V

> (send M9999 def-subpart M60985/94-7380-5)
M60985/94-7380-5

> (send M9999 get-parts)
(M60985/94-7380-5)

> (send M60985/94-7380-5 set-property-value x-offset 3.15)
3.15

> (send M60985/94-7380-5 show-self)
M60985/94-7380-5
    X-OFFSET: 3.15
```

```
> (send capacitor def-subclass M99/99-99)
M99/99-99

> (send M99/99-99 show-self)
M99/99-99

> (send M99/99-99 def-instance M99/99-99-1)
M99/99-99-1

> (send M99/99-99-1 show-self-in-detail)
M99/99-99-1
    PART-NUMBER: ()
    REFERENCE-DESIG: ()
    X-ORIGIN: ()
    Y-ORIGIN: ()
    X-OFFSET: ()
    Y-OFFSET: ()
    ORIENTATION: ()
    COMPONENT-CLASS: ()
    NUMBER-PINS: ()
    LIBRARY-REF: ()
    MAX-LENGTH: ()
    MAX-WIDTH-DIA: ()
    MAX-THICK: ()
    LEAD-DIAMETER: ()
    MILITARY-SPEC: ()
    PART-CODE: ()
    DESCRIPTION: ()
    VALUE: ()
    TOLERANCE: ()
    RATING: ()

> (send M9999 def-subpart M99/99-99-1)
M99/99-99-1

> (send M9999 get-parts)
(M99/99-99-1 M60985/94-7380-5)
```

Hierarchical constraint management also contributes to customized representations. ODM's semantic constraint facilities permit constraint *cascading* along a generalization hierarchy. For example, in Figure 8.16, *CAPACITOR* is a specialization of *COMPONENT*, and capacitor *M60985/94-7308* is a specialization of *CAPACITOR*. Therefore, the value constraint of a property, such as *number-pins*, may be more specialized for a specific capacitor than for a component in general. In Figure 8.16, the value constraints on ''*number-pins*'' range from *numeric*; to a specific value, namely, 2, for capacitor M60985/94-7308.

ODM Dialogue 8.3 is based on the scenario presented in Figure 8.16. These hierarchical constraint facilities further improve the robustness of ODM's dynamic schema facilities.

Dialogue 8.3 OML dialogue checking component constraints

```
> (send component def-instance component-500)
COMPONENT-500

> (send component-500 set-property-value number-pins none)

** Error: NONE -- not a legal value

> (send component-500 set-property-value number-pins 0)
0

> (send capacitor def-instance capacitor-600)
CAPACITOR-600

> (send capacitor-600 set-property-value number-pins 0)

** Error: 0 -- not a legal value

> (send capacitor-600 set-property-value number-pins 8)
8

> (send M60985/94-7380 def-instance M60985/94-7380-9)
M60985/94-7380-9

> (send M60985/94-7380-9 set-property-value number-pins 8)

** Error: 8 -- not a legal value

> (send M60985/94-7380-9 get-property-value number-pins)
2
```

The examples described above begin to reduce the distinction between conventional DBMS schema and data. As I previously discussed, data management practices promoted by dynamic schemata are desirable, especially in

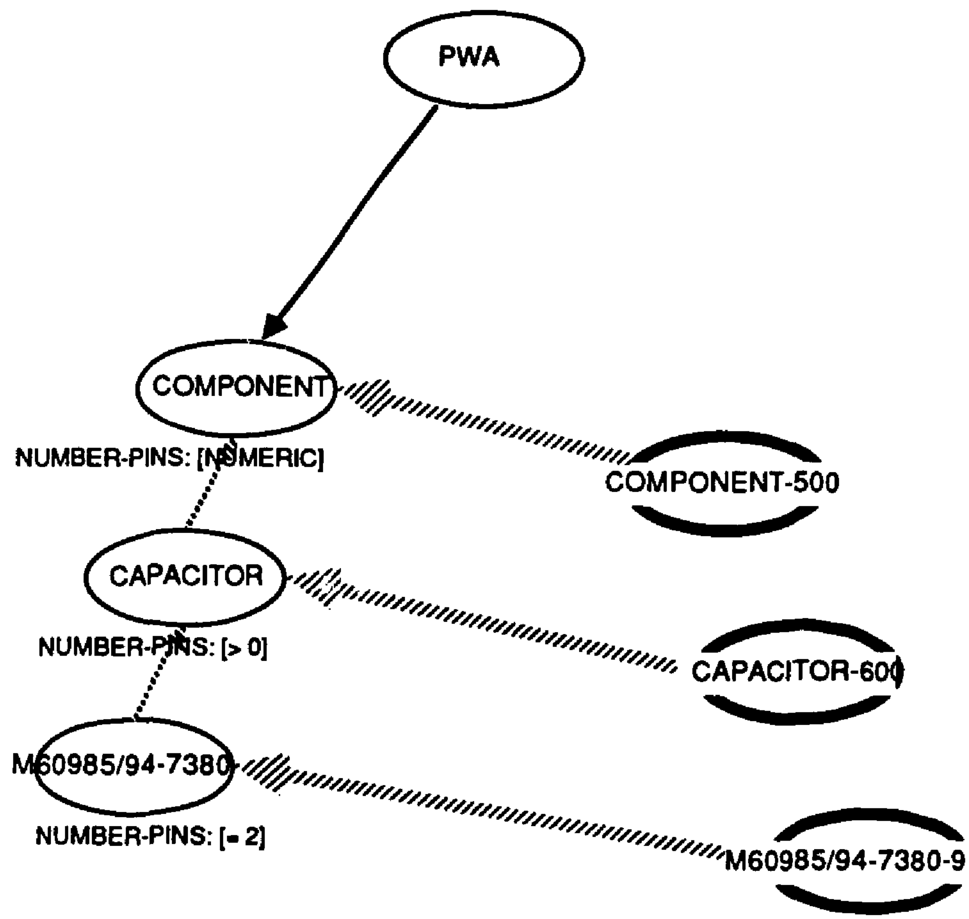


Figure 8.16 Hierarchical constraints in ODM

CAD/CAM applications where the structure of a product should be reflected in the structure of its data.

8.2 Hughes PF system

A project currently under development at Hughes is applying expert system technology for analyzing *producibility* data in mechanical design. Producibility analysis considers the physical and structural properties of a machined part during the design phase, and determines how these properties affect fabrication. For example, if two holes are drilled too close to one another, a weakened structure results. Currently, process planners and manufacturing planners review engineering drawings and accompanying notes and instructions. They must determine if a machined part can be manufactured according to the designer's specifications. Hughes Producibility Feedback (PF) system aims to automate these tasks.

In the rest of this chapter, I discuss the use of ODM features for producibility analysis currently performed by expert system rules. One machined part design utilized at Hughes for testing their PF system is presented in Figure 8.17. The data base for this drawing contains geometry data; draw form and datum specifications; and feature data for holes and surfaces. The ODM version of these data bases is used for the analysis presented in the following sections.

8.2.1 Expressing standards as constraints

In all design environments, numerous constraints must be considered and enforced. Many constraints reflect common sense; or, they are part of the knowledge retained by a designer. For example, mechanical designers know

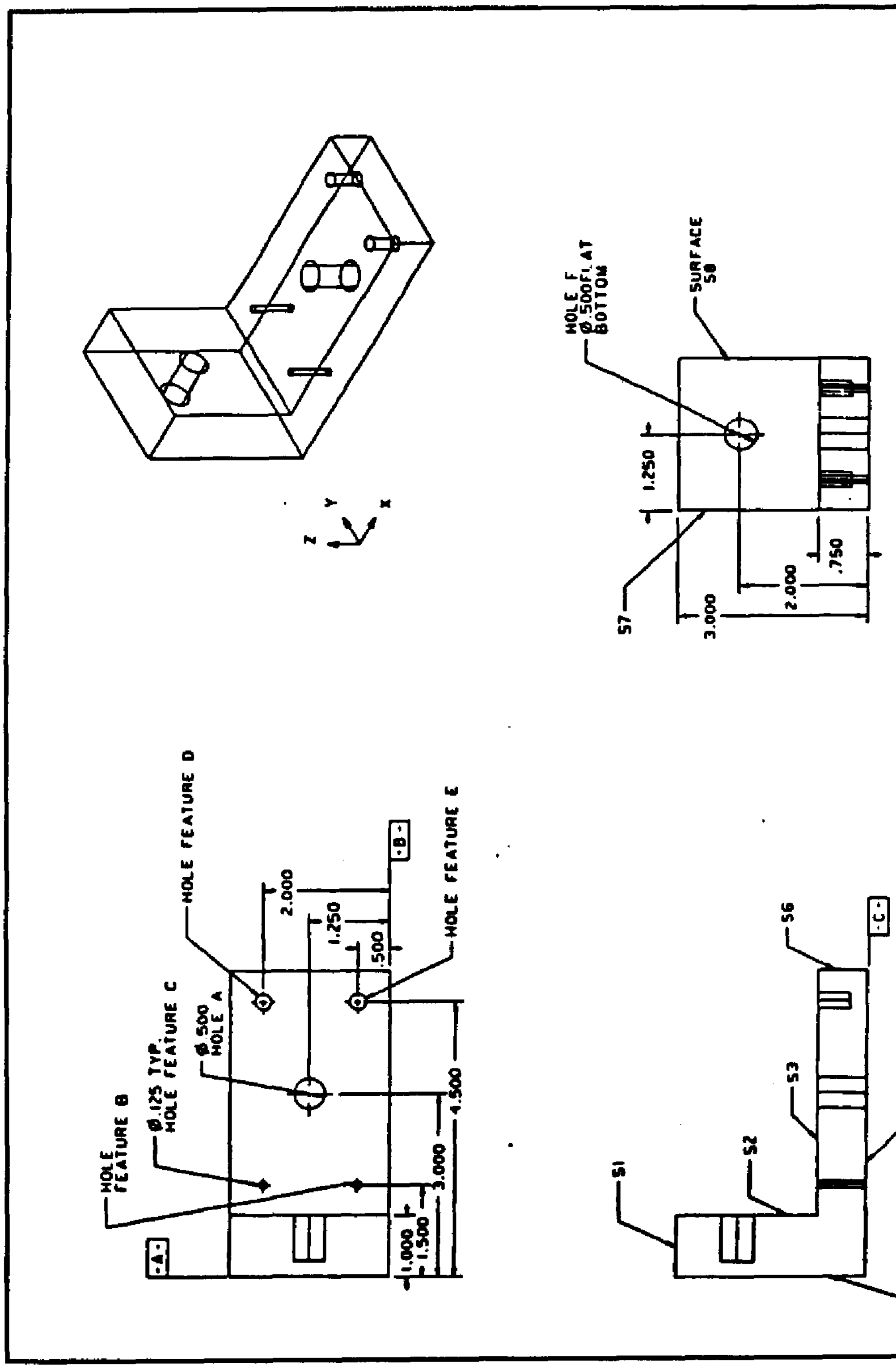


Figure 8.17 PF sample part

principles of structural and stress analysis, and they confine their designs to conform to these principles. In addition to constraints imposed by the application domain, industries also enforce their own constraints on properties of their products.

Throughout a design and manufacturing cycle, constraints are continuously checked, validated, and amended. If a design flaw goes unnoticed until the part is on the production line, vast corporation losses in terms of time and money are incurred. Corporations continually search for techniques to automatically verify design data. Standard DBMS fall short in terms of this goal. Value and structural integrity constraints are usually aimed at limitations on the *data*. These constraints are imposed by computational components of the systems such as: DBMS software, DBMS hardware, and secondary storage. For example, if the *name* field in a data base record is limited to 32 characters, this restriction doesn't imply that in the real world no one is assigned a name with more than 32 characters. Similarly, if a *parent/child* data base enforces existence constraints disallowing *orphans*, you cannot infer that there aren't any parent-less children! Some constraints do help maintain consistency with the world being modeled, for instance, verification of calendar dates. However, in general, DBMS constraints maintain the integrity of the data being managed; they do not maintain the integrity of the world being modeled.

ODM facilities for representing and verifying semantic constraints permit many domain *standards* to be incorporated into a data base and maintained by a data management system. Lockheed cites the following advantages of coupling standards verification with data management processes. First, interactive verification enables designers to reenter erroneous data during the design process. Current batch verification loops through all data and produces error reports. Designers review the error reports and make appropriate corrections. The data is then resubmitted for another iteration of batch verification. Alternatively, interactive checking produces a tight loop of iteration over single data values; designers reenter data for a single property until a value is accepted. Another advantage is knowledge centralization. Standards, operationalized as constraints, centralize information within an assembly or part representation. This localization of knowledge reduces the number of different information sources, like manuals and handbooks, which are consulted. Also, centralized knowledge is easy to access, view, and modify. Maintaining knowledge as data base constraints contrasts with the use of an expert system where knowledge is contained within procedural rules or other knowledge representation.

In the examples described below, I show how expert system knowledge is incorporated in an ODM data base through semantic entity representation and constraint specification. These examples refer to the machined part in Figure 8.17. I also present examples of rules implemented in Hughes PF expert system and illustrate how the knowledge embedded in these rules is verified by ODM constraint maintenance.

8.2.2 PF knowledge in ODM networks

Hughes PF system analyzes *hole* and *surface* features. Therefore, a practical representation of a machined part requires properties relevant to holes and surfaces. Figure 8.18 shows an ODM network with intensions, specialization links, and aggregation relationships necessary for modeling producibility data. The corresponding OEL input including property specifications is given below.

ODM instance data for hole and surface features of Figure 8.8.17 is presented in Appendix P.

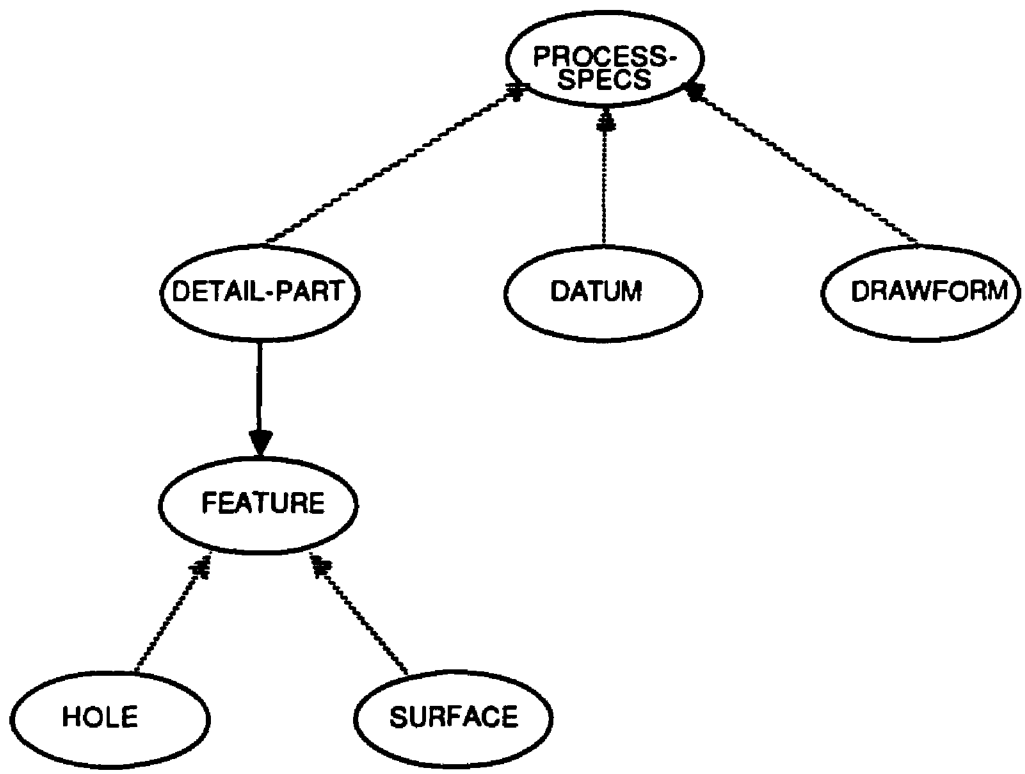


Figure 8.18 ODM representation of machined part

OEL specification of machined part

```
(c detail_part
   draw_form: |draw_form|
   datum: |datum|
   number_of_holes: I
   holes: (list-of: |hole|)
   number_of_surfaces: I
   surfaces: (list-of: |surface|)
```

```
   size_x_axis: R
   size_y_axis: R
   size_z_axis: R
*  part_volume: (less-than: 400.0)
*  material: (one-of: aluminum steel)
*  original_form: (one-of: casting forging barstock plate)
*  original_form_x_axis: (less-than: 20.0)
*  original_form_y_axis: (less-than: 20.0)
*  original_form_z_axis: (less-than: 18.0))

(c draw_form
   detail_part: |detail_part|
*  designer: (one-of: smith jones clark)
   revisions: T
*  block_tolerance: .001
   project: T
   program: T)

(c datum
   detail_part: |detail_part|
   primary_datum: T
   secondary_datum: T
   tertiary_datum: T
   ref_datum_a: T
   ref_datum_b: T
   ref_datum_c: T)

(c feature /detail_part/)

(c hole |feature|
   detail_part: |detail_part|
   ent_surface: T
   exit_surface: T
   int_x_geo: T
*  diameter: (one-of: 0.0625 0.1250 0.1875 0.2500 0.3750
                                            0.5000 0.6250)
   dia_tol: T
   bottom_cond: L
   surface_cond: R
*  tap_size: (one-of: 3-48 3-56 4-48 6-32 8-32 10-24 12-28)
   pos_tol: R)

(c hole_ref
   detail_part: |detail_part|
   x_start_loc: T
   x_start_ref_surface: T
   x_end_loc: T
   x_end_ref_surface: T
   y_start_loc: T
   y_start_ref_surface: T
   y_end_loc: T
   y_end_ref_surface: T
   z_start_loc: T
   z_start_ref_surface: T
   z_end_loc: T
```

```
   z_end_ref_surface: T)

 (c surface |feature|
    detail_part: |detail_part|
    resident_plane: T
    x_bounding_plane_xy: (list-of: |surface|)
    y_bounding_plane_xy: (list-of: |surface|)
    x_bounding_plane_xz: (list-of: |surface|)
    z_bounding_plane_xz: (list-of: |surface|)
    y_bounding_plane_yz: (list-of: |surface|)
    z_bounding_plane_yz: (list-of: |surface|)
    datum_plane: T
*   fillet_radius: (greater-than: .015)
*   corner_radius: (greater-than: .015)
    type_of_surface: T
    surface_finish: T
    number_of_intersecting_holes: T)
```

For the following demonstrations, I selected eight PF rules which examine producibility data. Condensed versions of these rules are presented below. In most cases, the rules reflect industry or corporation standards. In the PF system, if data is determined to be non-standard, an appropriate error condition is generated. However, the PF system is a passive analysis tool; therefore, no attempt is made to flag or reject unacceptable values. The information verified by these rules is expressed in ODM by those properties listed above which are prefaced by an "*".[1] Below I discuss three PF rules in detail and describe how ODM actively rejects nonstandard values when they are entered into the data base.

Rule (1) determines whether the part under consideration conforms to the requirements for *standard* processing. If any of the three conditions expressed in Rule (1) are violated, a *"Process type is nonstandard"* message is reported. This PF rule combines three conditions into one rule, but supplies little

[1]In the OEL specification of a machined part, an "*" is not part of the OEL syntax; it is only included for discussion purposes.

information, if the rule fails, about erroneous values. The knowledge expressed in this rule corresponds to value constraints associated with three properties of the intension, *DETAIL-PART*: *part-volume, material,* and *original-form,*. An ODM value constraint on *part-volume* restricts the volume to a value less than 400.0. The *material* property is limited to either aluminum or steel. Similarly, the value of *original-form* must be one of four possible values. In ODM, an unacceptable value for any of the relevant properties is rejected immediately.

PF expert system rules

```
Rule (1) IF original form IS CONTAINED IN
                    {casting forging barstock plate}
         AND
            material IS CONTAINED IN {aluminum steel}
         AND
            part volume IS LESS THAN 400.0
         THEN
            EXECUTE print ("Process type is standard")
         ELSE
            EXECUTE print ("Process type is nonstandard")


Rule (2) IF (original form x axis IS LESS THAN 20.0)
         AND
            (original form y axis IS LESS THAN 20.0)
         AND
            (original form z axis IS LESS THAN 18.0)
         THEN
            EXECUTE print
                ("Process type equals standard mill size")
         ELSE
            EXECUTE print
                ("Process type equals nonstandard mill size")


Rule (3) IF designer IS CONTAINED IN {smith jones clark}
         THEN
            EXECUTE print ("Designer has been cleared")
         ELSE
            EXECUTE print ("Designer has not been certified")


Rule (4) IF block tolerance IS EQUAL TO .001
         THEN
            EXECUTE print ("Block tolerance is acceptable")
         ELSE
```

```
            EXECUTE print ("Block tolerance is unacceptable")


Rule (5) IF diameter IS CONTAINED IN { 0.0625 0.1250 0.1875
                                0.2500 0.3750 0.5000 0.6250}
         THEN
            EXECUTE print
                  ("Hole diameter is standard size hole")
         ELSE
            EXECUTE print
                  ("Hole diameter is not a standard size hole")


Rule (6) IF tap size IS CONTAINED IN { 3-48 3-56 4-48 6-32
                                          8-32 10-24 12-28 }
         THEN
            EXECUTE print ("Called out tap size is ok")
         ELSE
            EXECUTE print ("Called out tap size is nonstandard")


Rule (7) IF fillet radius IS GREATER THAN .015
         THEN
            EXECUTE print
                      ("Fillet radius is permitted")
         ELSE
            EXECUTE print ("Fillet radius is less than permitted")


Rule (8) IF corner radius IS GREATER THAN .015
         THEN
            EXECUTE print
                      ("Corner radius is acceptable")
         ELSE
            EXECUTE print ("Corner radius is unacceptable")
```

The *block-tolerance* of an engineering drawing is verified by Rule (4). Block-tolerance, a property of *DRAW-FORM*, is restricted to a specific value, namely, .001. Any other value produces an *''Unacceptable block tolerance''* message in the PF system and, likewise, is rejected by ODM.

In Rule (7) the attribute of a surface feature is examined. *Fillet radius* is a property of *SURFACE* and is limited to a value greater than .015. The corresponding ODM value constraint limits the property accordingly. Dialogue 8.4 presents an ODM session setting and retrieving properties validated by these eight PF rules. The data base underlying this ODM session contains the OEL specification schema presented above and input data listed in Appendix P corresponding to Hughes PF test data represented in Figure 8.17.

Dialogue 8.4 OML dialogue checking producibility constraints

```
> (send new_part show-self)
NEW_PART
    SIZE_X_AXIS: 5.0
    SIZE_Y_AXIS: 2.5
    SIZE_Z_AXIS: 3.0
    PART_VOLUME: 20.5
    MATERIAL: ALUMINUM
    ORIGINAL_FORM: CASTING
    ORIGINAL_FORM_X_AXIS: 5.165
    ORIGINAL_FORM_Y_AXIS: 2.625
    ORIGINAL_FORM_Z_AXIS: 3.165
    NUMBER_OF_HOLES: 6
    NUMBER_OF_SURFACES: 8

> (send new_part set-property-value material plastic)


** Error: PLASTIC -- not a legal value

> (send new_part set-property-value material steel)
STEEL

> (send new_part set-property-value original_form block)

** Error: BLOCK -- not a legal value


> (send new_part set-property-value original_form barstock)
BARSTOCK

> (send new_part set-property-value part_volume 550.0)

** Error: 550.0 -- not a legal value


>(send new_part set-property-value part_volume 350.0)
350.0

> (send new_part set-property-value original_form_z_axis 20.0)

** Error: 20.0 -- not a legal value


> (send new_part set-property-value original_form_z_axis 14.0)
```

```
14.0

> (send new_part show-self)
NEW_PART
    SIZE_X_AXIS: 5.0
    SIZE_Y_AXIS: 2.5
    SIZE_Z_AXIS: 3.0
    ORIGINAL_FORM_X_AXIS: 5.165
    ORIGINAL_FORM_Y_AXIS: 2.625
    NUMBER_OF_HOLES: 6
    NUMBER_OF_SURFACES: 8
    MATERIAL: STEEL
    ORIGINAL_FORM: BARSTOCK
    PART_VOLUME: 350.0
    ORIGINAL_FORM_Z_AXIS: 14.0

> (send new_part_draw_form show-self)
NEW_PART_DRAW_FORM
    DETAIL_PART: NEW_PART
    DESIGNER: CLARK
    REVISIONS: REV_A
    BLOCK_TOLERANCE: 0.001
    PROJECT: DEMO
    PROGRAM: UCLA

> (send new_part_draw_form set-property-value
                              designer johnson)

** Error: JOHNSON -- not a legal value


> (send new_part_draw_form set-property-value
                                designer smith)
SMITH

> (send new_part_draw_form set-property-value
                             block_tolerance .02)

** Error: 0.02 -- not a legal value


> (send new_part_draw_form set-property-value
                            block_tolerance .001)
0.001

> (send new_part_draw_form show-self)
NEW_PART_DRAW_FORM
    DETAIL_PART: NEW_PART
    REVISIONS: REV_A
    PROJECT: DEMO
    PROGRAM: UCLA
    DESIGNER: SMITH
    BLOCK_TOLERANCE: 0.001

> (send hole_b_data show-self)
```

```
HOLE_B_DATA
    DETAIL_PART: NEW_PART
    ENT_SURFACE: S5
    EXIT_SURFACE: S3
    INT_X_GEO: S1
    DIAMETER: 0.125
    DIA_TOL: 0.001
    BOTTOM_COND: THRU
    SURFACE_COND: 0.001
    TAP_SIZE: 3-56
    POS_TOL: 0.001

> (send hole_b_data set-property-value diameter .7500)

** Error: 0.75 -- not a legal value


> (send hole_b_data set-property-value diameter .6250)
0.625

> (send hole_b_data set-property-value tap_size 4-32)

** Error: 4-32 -- not a legal value


> (send hole_b_data set-property-value tap_size 4-48)
4-48

> (send hole_b_data show-self)
HOLE_B_DATA
    DETAIL_PART: NEW_PART
    ENT_SURFACE: S5
    EXIT_SURFACE: S3
    INT_X_GEO: S1
    DIA_TOL: 0.001
    BOTTOM_COND: THRU
    SURFACE_COND: 0.001
    POS_TOL: 0.001
    DIAMETER: 0.625
    TAP_SIZE: 4-48

> (send s3 show-self)
S3
    DETAIL_PART: NEW_PART
    RESIDENT_PLANE: (X Y)
    X_BOUNDING_PLANE_XY: ()
    Y_BOUNDING_PLANE_XY: ()
    X_BOUNDING_PLANE_XZ: ()
    Z_BOUNDING_PLANE_XZ: ()
    Y_BOUNDING_PLANE_YZ: ()
    Z_BOUNDING_PLANE_YZ: ()
    DATUM_PLANE: NO
    FILLET_RADIUS: 0.02
    CORNER_RADIUS: 0.028
    TYPE_OF_SURFACE: MACH
```

```
    SURFACE_FINISH: 0.001
    NUMBER_OF_INTERSECTING_HOLES: 5

> (send s3 set-property-value fillet_radius .015)

** Error: 0.015 -- not a legal value


> (send s3 get-property-value fillet_radius)
0.02

> (send s3 set-property-value fillet_radius .024)
0.024

> (send s3 get-property-value corner_radius)
0.028

> (send s3 show-self)
S3
    DETAIL_PART: NEW_PART
    RESIDENT_PLANE: (X Y)
    X_BOUNDING_PLANE_XY: ()
    Y_BOUNDING_PLANE_XY: ()
    X_BOUNDING_PLANE_XZ: ()
    Z_BOUNDING_PLANE_XZ: ()
    Y_BOUNDING_PLANE_YZ: ()
    Z_BOUNDING_PLANE_YZ: ()
    DATUM_PLANE: NO
    CORNER_RADIUS: 0.028
    TYPE_OF_SURFACE: MACH
    SURFACE_FINISH: 0.001
    NUMBER_OF_INTERSECTING_HOLES: 5
    FILLET_RADIUS: 0.024
```

The three rules previously discussed consider properties independently. That is, a nonstandard condition is determined by examining the value of a single property in isolation. In the PF rule given below, a nonstandard condition depends on *two* properties of a hole, *diameter* and *diameter-tolerance.*

```
IF ((diameter > 0.125 AND diameter < 0.750)  AND
    (diameter-tolerance < 0.0005))
OR
   ((diameter > 0.750 AND diameter < 2.0)  AND
    (diameter-tolerance < 0.0008))
OR
   ((diameter > 2.0 AND
    (diameter-tolerance < 0.0015))
THEN
   EXECUTE print("Tolerance callout is too tight")
```

ODM permits analogous constraints, although, the constraint specification is more procedural in nature. An ODM value constraint for the property, *diameter-tolerance*, of the *HOLE* intension is the following:

```
(send HOLE set-property-slot diameter-tolerance p-lambda
  (lambda (x self)
    (let* ((diameter (ask self get-property-value diameter)))
          (if diameter
                then
                  (or (and (> diameter .125)
                           (< diameter .750)
                           (> x .0005))
                      (and (> diameter .750)
                           (< diameter 2.0)
                           (> x .0008))
                      (and (> diameter 2.0)
                           (> x .0015)))
                else (number? x)))))
```

In the above constraint, if the value of *diameter* has not been entered, then any numeric value is allowed for the value of *diameter-tolerance*. However, if the value of *diameter* has already been set, then *diameter-tolerance* is constrained accordingly.

Combining data entry with producibility analysis benefits design operations in four ways. First, immediate feedback is produced when invalid data is entered. Second, domain knowledge is associated with semantic schema definitions; therefore, it is easier to locate, view, and modify. Integrating design and analysis tasks is another benefit contributing to production line efficiency.

Finally, a higher degree of consistency is afforded during design phases. The resulting CAD environment helps maintain and control the integrity of product designs.

8.3 ODM validation

Validation of research results affords impartial confirmation that the goals of the research have been met. For this dissertation work, I relied on Hughes personnel to independently certify the CAD/CAM DBMS improvements which I claimed to have achieved. Hughes employees critically reviewed each phase of the evaluation process described earlier in this chapter. They supported and approved my evaluation methodology using both PWA and PF application data.

Initial conversion of existing Hughes PWA data to the corresponding ODM organization was uncomplicated and direct. Hughes personnel agreed that the capabilities of the resulting ODM data bases were at least as powerful as their existing data management facilities.

For the second phase of evaluation, CAM department members at Hughes supplied notes and diagrams documenting their conceptual view of PWA data [Nig85]. These documents formed the kernel of new PWA data bases which I designed using the ODM prototype software. Hughes staff members examined the restructured data organization including schemata, data instances, and constraints. In addition, they reviewed the dialogues presented in the preceding sections demonstrating interactions with the ODM computer software. Their analysis confirmed those benefits which I highlighted in the sample sessions [Liu85, Zuc85]. They also emphasized the following advantages over their conventional DBMS practices:

- *Conceptual view of application data equates with logical DBMS view.* Currently, a wide gap exists between the conceptual representation of PWAs and the logical organization of existing data bases. Bridging this gap enables designers and manufacturers to interact with the data bases in a fashion which is most natural for them.
- *Interactive browsing.* Using ODM generalization and aggregation networks, users can inspect the properties, subparts, and classifications of PWAs, components, hardware, and fasteners. They can directly access the content (properties and constraints) of a data base object, or they can view an object as a node in a network and traverse connecting links to access related objects.
- *Built-in "contains" relationship with transitive closure operations.* Bill of Materials data can be processed more effectively if it is organized hierarchically and users can view and query the data in a hierarchical manner.
- *Modifiable schema supporting PWA changes.* Decisions concerning the structure of PWAs and components are sometimes deferred by the designers. With modifiable schema structures, the data bases for these entities can be generated as they are designed, instead of waiting until all design decisions have been made.
- *Reduction of duplicated data.* This improvement eases the task of maintaining consistency across many duplicate data items.
- *Centralization of component data.* By using the ODM architec-

ture for PWA data, it is no longer necessary to access four Master Component Library files to retrieve all data for an existing component. Furthermore, when constructing a new PWA, data is entered into a single data base rather than four transfer files.

- *More meaningful presentation of IGES graphical data.* IGES data organized as ODM objects is much more comprehensible to users. Existing IGES formats are only efficient for graphical CAD systems supporting IGES standards.

The demonstrations presented in this chapter, along with the described validation process, affirm the utility of ODM in operational CAD/CAM applications. I have shown that ODM is comparable to relational models for maintaining Hughes PWA data. More importantly, PWA and PF applications served as authentic testbeds exemplifying significant improvements in CAD/CAM data management. The corresponding ODM data bases exhibit the qualities and functionality advocated by this research.

CHAPTER 9

CONCLUSIONS

The objectives of this dissertation were to analyze CAD/CAM data management practices, identify deficiencies, and develop improved methods for maintaining integrated CAD/CAM data. This research produced an object-oriented data model and software prototype system, ODM, with sophisticated DBMS capabilities addressing the limitations of existing facilities. In this concluding chapter, I first review evidence supporting the need for this research. The next section itemizes the contributions of the research from a CAD/CAM application perspective, and also from the viewpoint of semantic data modeling. I conclude with a discussion of ODM's limitations, its potential for future research and development efforts, and its applicability to other domains.

9.1 Factors necessitating improved CAD/CAM data management

The proliferation of CAD/CAM application systems indicates that automation in all phases of design, engineering, and manufacturing is booming. My interactions with Lockheed, Rockwell, and Hughes employees emphasized the need for improved CAD/CAM data management facilities supporting diverse application systems. The requirements analysis phase of this research revealed inefficiencies due to the following factors:

- Each CAD/CAM application requires specialized input data and generates system-specific output.

- Multiple independent data bases cause data flow gaps, and hamper automatic translation mechanisms.
- Manual preparation and transfer of data between applications reduces efficiency and increases the chance of errors.
- Enormous amounts of redundant data and duplicate data processing hinder consistency maintenance and data retrieval.
- A wide gap exists between an engineer's view of product design and production, and the organization of corresponding data in today's DBMS.

9.2 Contributions

Existing data management systems are inadequate for overcoming the resulting inefficiencies. The research presented in this dissertation recommends solutions for achieving effective CAD/CAM data management. Specifically, the accomplishments of this work are the following:

- *An information management environment for interleaving mechanical design, data entry, and design validation tasks.* Currently, initial data entry for a new part occurs after a design is complete, and design validation follows data entry as an off-line task. Design inconsistencies are not recognized until a design and its data have been committed, at which time, a second iterative pass through design, data entry, and validation is required for corrections. Integrating these activities, first, bridges a gap between these tasks; and second, allows design experts to select and control the type and structure of relevant information stored in the data base.

- *A data model supporting a logical schema which equates with the conceptual schema maintained by manufacturing experts.* Most data models do not directly support the conceptual view of an enterprise generated during *data base design* phases. In ODM, complex conceptual entities and relationships can be mapped onto data base objects, thereby, retaining the conceptual organization for future access and manipulation.
- *Data manipulation capabilities which directly support BOM processing.* The BOM organization of assemblies and parts is ubiquitous in the manufacturing industry. ODM directly supports composition hierarchies and provides primitive operations for retrieving BOM data.
- *Extended data types for maintaining heterogeneous data.* Complex combinations of graphical, geometrical, manufacturing, and administrative data can be represented as ODM objects. Domain object types, like extended data types, can be customized to fit any application requiring the data as input.
- *Semantic constraint facilities for maintaining the consistency of mechanical designs.* By representing semantic features and relationships, consistency checking of design criteria can be included in the schematic description of entities. Unacceptable design decisions can be rejected and redesigned early in the manufacturing cycle before subsequent activities, like tool design, are initiated.
- *A methodology for partial or total conversion to integrated CAD/CAM data management.* A directory approach for maintaining data sources permits the existence of multiple data bases,

yet, helps to *conceptually* centralize distributed data repositories. This organization provides users with a starting point to begin searching for required data bases and files.

Proof of these concepts was demonstrated by the implementation of an ODM prototype software system. An OEL (Object Entry Language), and OML (Object Manipulation Language) were developed for interacting with the ODM prototype. Hierarchical and heterogeneous data types, semantic constraint specification, and transitive closure operations are supported in the operational ODM prototype. Interactive sessions with the prototype exemplify the CAD/CAM DBMS goals which were achieved.

To illustrate the practical benefits of this research in a manufacturing setting, I coordinated with Hughes data management and manufacturing personnel. To validate the utility and application of this research, I demonstrated the following capabilities of the ODM software system using Hughes PWA and producibility data:

- directory data bases integrating MCL, IGES, and PWA transfer data
- a BOM schema reflecting the conceptual PWA and component organization
- dynamic creation of new PWA and capacitor schema and data
- hierarchical constraints for PWA components
- interactive producibility checking by converting expert system rules to ODM constraints

During the course of this research, I also investigated theoretical aspects of object-oriented models in programming languages, data management and knowledge representation. The data model I developed contributes to the field of semantic data modeling with the advancements outlined below:

- An object-oriented model with explicit intensional and extensional semantics based on set theory and predicate logic.
- Formalisms which relate aggregation and generalization principles, and inferencing theorems derived from their integration.
- The application of meta-knowledge in DBMS schemata enabling dynamic schema structures.
- A computer software system achieving the functionality of the ODM theoretical model.

9.3 Limitations and future work

Throughout this document, I have suggested aspects of ODM which would profit by additional research. To summarize, efforts focused on the following topics would extend the utility of ODM as a viable CAD/CAM DBMS:

- *improved user interface (graphical and textual).* An object-oriented message-passing language is generally too verbose for efficient interactivity. Graphical manipulation of icons representing domain intensions and instances is desirable.
- *constraint specification languages.* The underlying implementation language is currently used for representing procedural constraints. Instead, a language specifically suited for expressing relationships and conditions over domain objects should be investigated.

- *generalized aggregation principles.* In ODM, aggregation applies only to the composition of physical parts. A natural extension of this work would consider more generalized forms of aggregation.
- *secondary storage facilities.* One hallmark of generalized DBMS is their ability to maintain large data sets efficiently in secondary storage. Efforts in this direction must also be pursued for object-oriented data models.

The extensions described above apply to domain independent aspects of ODM. Additional investigation, however, can be pursued toward a better understanding of mechanical design and engineering, and the data management tasks entailed by these disciplines. Insights gained by analyzing domain tasks encourage developments tailored specifically to CAD/CAM needs. One such task, not addressed by this research, focuses on version control and configuration management. These capabilities are clearly necessary in the types of manufacturing environments I analyzed, namely, aerospace and electronics. Also, the developed model encourages the incorporation of domain knowledge within the data base schema. Identifying and incorporating the following kinds of information will further benefit designers and engineers utilizing integrated CAD/CAM DBMS:

- static and dynamic properties of manufactured parts
- semantic representations of part features
- assembly and part taxonomies
- semantic models for graphical and geometrical representations
- libraries of design validation procedures

My review of related work indicates that corporate projects focused on integrated CAD/CAM DBMS are inadequate. Therefore, collaborative research and industry efforts in this direction must be encouraged. As a follow-up project, discussions with Hughes data management personnel are continuing toward the goal of applying ODM facilities in their production environment.

Although the main emphasis of this work is on mechanical design and manufacturing; the developed methodology and tools for CAD/CAM data management also apply to other domains. Disciplines involving the construction or synthesis of physical entities can benefit from facilities for modeling *BOM-like* data exhibiting the *contains* relationship. Domains whose data items and structure are dynamic over time require more robust and dynamic schemata, such as those developed by this research. Finally, integration of heterogeneous data types is a goal in many DBMS applications. Most enterprises must maintain multiple data repositories because facilities for integrating heterogeneous data types are limited.

References

[Afs85] Afsarmanesh, H., McLeod, D., Knapp, D., and Parker, A., "An extensible object-oriented approach to databases for VLSI/CAD," pp. 13-24 in *Proceedings of 11th conference on very large data bases,* Stockholm (1985).

[Ahr84] Ahrens, L., February 1984. Personal communication.

[Ast80] Astrahan, M.M., "A history and evaluation of System R," RJ2843(36129), IBM Research Laboratory, San Jose, CA (1980).

[BCS83] *BCSS design document,* 1983. Boeing internal correspondence.

[Bar81] Barr, A. and Feigenbaum, E.A., *The handbook of artificial intelligence,* William Kauffman, Inc., Los Altos, CA (1981).

[Bir73] Birtwistle, G., Dahl, O., Myhrhaug, B., and Nygaard, K., *Simula begin,* Van Nostrand Reinhold, New York (1973).

[Bor79] Borning, A., "THINGLAB: A constraint simulation laboratory," CS-79-746, Stanford University, Palo Alto, CA (1979).

[Bor80] Borkin, S.A., *Data models,* The MIT Press, Cambridge, MA (1980).

[Bra78] Brachman, R.J., "A structural paradigm for representing knowledge," Report No. 3605, Bolt, Beranek and Newman, Inc., Cambridge, MA (May 1978).

[Bra83] Brachman, R.J., "What IS-A is and isn't: An analysis of taxonomic links in semantic networks," *Computer* 16(10), pp.30-36 (October 1983).

[Bra85] Brachman, R. and Schmolze, J., "An overview of the KL-ONE knowledge representation system," *Cognitive Science* 9(2), pp.171-216 (1985).

[Bro84] Brodie, M.L., Blaustein, B., Dayal, Y., Manola, F., and Rosenthal, A., "CAD/CAM database management," *Database Engineering* 7(2), pp.12-20 (June 1984).

[Can83] Cannon, D., November 1983. Personal communication.

[Car79] Cardenas, A.F., *Data base management systems*, Allyn and Bacon, Inc., Boston, MA (1979).

[Cha81] Chang, T.C. and Wysk, R.A., "An integrated CAD/automated process planning system," *AIIE Transactions* 13(3), pp.223-233 (September 1981).

[Che76] Chen, P.P., "The entity-relationship model: Toward a unified view of data," *ACM Transactions on Database Systems* 1(1), pp.9-36 (March 1976).

[Cod71] *Codasyl data base task group report, 1971*, Association for Computing Machinery, New York (April 1971).

[Cod79] Codd, E.F., "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems* 4(4), pp.397-434 (December 1979).

[Cop84] Copeland, G. and Maier, D., "Making Smalltalk a database system," *ACM Sigmod Record* 14(2), pp.316-325 (June 1984).

[Cur81] Curtice, R.M., "Data dictionaries: An assessment of current practice and problems," pp. 564-570 in *Proceedings of 7th conference on very large data bases*, Cannes, France (September 1981).

[DMD86] *AI Trends* 2(5), DM Data, Inc. (February 1986).

[Dat81] Date, C.J., *An introduction to database systems*, Addison-Wesley, Menlo Park, CA (1981).

[Dav78] Davis, R., "Knowledge acquisition in role-based systems," pp. 99-134 in *Patern directed inference systems*, ed. D.A. Waterman, Academic Press, New York (1978).

[DeW81] Baroody, A.J. Jr., "An object-oriented approach to database system implementation," *ACM Transactions on Database Systems* 6(4), pp.576-601 (December 1981).

[Eas78] Eastman, C., "The representation of design problems and maintenance of their structure," pp. 335-357 in *Artificial intelligence and pattern recognition in computer-aided design*, North Holland Publishing Company, Amsterdam (1978).

[Eas86] Eastman, C.M. and Lafue, G.M.E., "Semantic integrity transactions in design databases," pp. 39-54 in *File structures and data bases for CAD*, ed. J. Encarnacao, North Holland Publishing Company, Amsterdam (1986).

[Eco83] Economopoulos, P. and Lochovsky, F. H., "A system for managing image data," in *Proceedings of 9th IFIP World Computer Congress*, Paris (1983).

[Enc83] Encarnacao, J. and Schlechtendahr, E.G., *Computer aided design*, Springer-Verlag, Heidelberg, FRG (1983).

[Eps77] Epstein, R., "Creating and maintaining a database using Ingres," Memorandum No. ERL-M77-71, University of California at Berkeley, Berkeley, CA (December 1977).

[Fah79] Fahlman, S., *NETL: A system for representing and using real-world knowledge*, The MIT Press, Cambridge, MA (1979).

[Fen85] Fenves, S.J., "Representation and processing of engineering design constraints in a relational database," pp. 343-347 in *Proceedings of COMPINT 85 computer aided technologies*, Quebec (September 1985).

[Fin79] Findler, N., *Associative networks*, Academic Press, New York (1979).

[God82] Goldberg, A. and Robson, D., *Smalltalk-80: The language and its implementation*, Addison-Wesley Publishing Company, Menlo Park, CA (1982).

[Gof82] Goldfine, N., "Information resource management: Strategies and tools," National Bureau of Standards Special Publication 500-92, Database Directions III Workshop (1982).

[Gra85] "Graphics kernel system: Functional description," ANSI X3,124-1885, American National Standard for Information System (1985).

[Gut82] Guttman, A. and Stonebraker, M., "Using a relational database management system for computer-aided design data," *Database Engineering* 5(2), pp.21-28 (June 1982).

[Ham81] Hammer, M. and D.McLeod,, *Database description with SDM: A semantic database model*, ACM Transactions on Database Systems (September 1981).

[Han78] Hanson, A.R. and Riseman, E.M., *Computer vision systems*, Academia Press, Inc., New York (1978).

[Hes83] Hess, G.J., *Computer integrated manufacturing*, December 1983. Presentation at winter meeting of American Society of Agricultural Engineers.

[Hoo85] Hooper, R.P., "An application of knowledge-based systems to electronic computer-aided engineering, design and manufacturing data base transport," Doctoral Dissertation, UCLA (1985).

[IDM] *IDMS: Concepts and facilities,* Cullinane Corporation, Wellesley, MA.

[Ini83] "Initial graphics exchange specification," NBSIR 82-2631 (AF), National Bureau of Standards (February 1983).

[Ito] Itoh, S. and Iisaka, J., *An image oriented database system,* IBM Scientific Center, Tokyo.

[Kam83] Kamvar, E., "Recognition of graphical data - A Bridge between CAD and CAM," Doctoral dissertation, UCLA (1983).

[Kin86] King, R., "A database management system based on an object-oriented model," pp. 443-468 in *Expert database systems,* ed. L. Kerschberg, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

[Kla82] Klahr, P., McArthur, D., Narain, S., and Best, E., "SWIRL: Simulating warfare in the ROSS language," N-1885-AF, The Rand Corporation, Santa Monica, CA (September 1982).

[Kat85] Katz, R.H., *Information management for engineering design,* Springer-Verlag, Berlin (1985).

[Led83] Ledbetter, E.B., February 1983. Lockheed internal correspondence.

[Lew83] Lewis, J., November 1983. Personal communication.

[Lil78] Lillehagen, F.M., *Modeling in CAD systems,* 1978. CAD Tutorial, SIGGRAPH conference.

[Liu] Liu, D., *Utilization of artificial intelligence in manufacturing,* Electro-Optical and Data Systems Group, Hughes Aircraft Company, El Segundo, CA.

[Liu85] Liu, D., June 1985. Personal communication.

[Lor82] Lorie, R.A., "Issues in database for design applications," pp. 213-222 in *File structures and data bases for CAD,* ed. J. Encarnacao, North Holland Publishing Company, Amsterdam (1982).

[Mac80] Machaver, C. and Blauth, R.E., *The CAD/CAM handbook,* Computervision Corporation, Bedford, MA (1980).

[Mai84] Maier, D. and Price, D., "Data model requirements for engineering applications," in *Proceedings of first international workshop on expert database systems*, ed. L. Kerschberg (1984).

[McA85] McArthur, D., Klahr, P., and Narain, S., "The ROSS language manual," N-1854-1-AF, The Rand Corporation, Santa Monica, CA (September 1985).

[McC82] McCarthy, J.L., "Metadata management for large statistical databases," in *Proceedings of 8th conference on very large databases* (September 1982).

[Mel84] Melkanoff, M.A., "The CIMS database: Goals, problems, case studies and proposed approaches outlined," pp. 78-93 in *Industrial engineering* (November 1984).

[Mil76] Miller, G.A. and Johnson-Laird, P.N., *Language and perception*, Harvard University Press, Cambridge, MA (1976).

[Min74] Minsky, M., "A framework for representing knowledge," MIT AI Memo 306, Cambridge, MA (June 1974).

[Mor86] Morgenstern, M., "The role of constraints in databases, expert systems, and knowledge representation," pp. 351-368 in *Expert database systems*, ed. L. Kerschberg, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

[Myl80] Mylopoulos, J., Bernstein, P.A., and Wong, H.K.T., "A language facility for designing database-intensive applications," *ACM Transactions on Database Systems* 5(2), pp.185-207 (June 1980).

[Nas78] Nash, J.H., "Graphic interaction with database systems," pp. 107-118 in *3rd International conference on computers in engineering and building design*, Brighton Metropole, Sussex, UK (March 1978).

[Nas83] Nashenberg, L., November 1983. Personal communication.

[New78] Newman, N.M. and Dam, A. Van, "Recent efforts toward graphics standardization," *ACM Computing Surveys* 10(4), pp.365-380 (December 1978).

[Nig85] Nightingale, C., June 1985, November 1985. Personal communication.

[Nil80] Nilsson, N.J., *Principles of artificial intelligence*, Tioga Publishing Company, Palo Alto, CA (1980).

[Noc84] Nocket, M., February 1984. Personal communication.

[Obj84] *Objects, message passing and flavors,* Lisp Language Documentation, Symbolics, Inc. (1984).

[Ora79] *Oracle introduction, Version 1.3,* Relational Software, Inc., Menlo Park, CA (1979).

[PDD83] *PDDS Overview,* Integrated Database Team, Lockheed-California Company, Burbank, CA (1983).

[Plo84] Ploufte, W., Kim, W., Lorie, R., and McNabb, D., "A database system for engineering design," *Database Engineering* 7(2), pp.48-55 (June 1984).

[Pro81] *Proceedings of IFIP WG5.2 working conference on file structures and data bases for CAD,* September 1981. Closing discussion.

[Qui68] Quillian, M.R., "Semantic Memory," pp. 27-70 in *Semantic information processing,* ed. M. Minsky, MIT Pressson, Cambridge, MA (1968).

[Ree82] Rees, J. and Adams, N., "A dialect of Lisp or, Lambda: The ultimate software tool," in *Proceedings ACM symposium on Lisp and functional programming* (1982).

[Req] Requicha, A.G., "Representations of rigid solid objects," pp. 2-78 in *CAD/CAM Computer science lecture notes,* Springer-Verlag.

[Sch75] Schroeder, J.R., Kiefir, W.C., and Guertim, R.L., "Stanford's generalized database system," pp. 120-143 in *Proceedings of 2nd conference on very large data bases,* Framingham, MA (1975).

[She84] Shephard, A. and Kerschberg, L., "PRISM: A knowledge based system for semantic integrity specification and enforcement in data base systems," *ACM Sigmod Record* 14(2), pp.307-315 (June 1984).

[Shi81] Shipman, D.W., "The functional data model and the data language DAPLEX," *ACM Transactions on Database Systems* 6(1), pp.140-173 (March 1981).

[Smi77] Smith, J.M. and Smith, D.C.P., "Data base abstractions: Aggregation and generalization," *Communications of the ACM* 20(6), pp.405-413 (June 1977).

[Smi84] Smith, R.G., *Structured object programming in Strobe,* Schlemberger Technology Corporation, Ridgefield, CT (1984).

[Sno83] Snodgrass, R., "An object-oriented command language," *IEEE Transactions on Software Engineering* SE-9(1) (January 1983).

[Sow84] Sowa, J.F., *Conceptual Structures: Information processing in mind and machine*, Addison-Wesley Publishing Co., New York (1984).

[Ste78] Stefik, M., "An examination of a frame-structured representation system," HPP-78-13, Heuristic Programming Project, Stanford University (1978).

[Ste80] Stefik, M.J., "Planning with constraints," Report No. 80-784, Computer Science Dept, Stanford University (1980).

[Sto76] Stonebraker, M., Wong, E., Kreps, P., and Held, G., "The design and implementation of Ingres," *ACM Transactions on Database Systems* 1(3), pp.189-222 (September 1976).

[Sto84] Stonebraker, M. and Guttman, A., "Using a relational database management system for computer-aided design data - An Update," *Database Engineering* 7(2), pp.56-60 (June 1984).

[Su86] Su, S.Y.W., "Modeling integrated manufacturing data with SAM*," *Computer* 19(1), pp.34-49 (January 1986).

[Sut65] Sutherland, I.E., "SKETCHPAD: A man-machine graphical communication system," TR 269, MIT Lincoln Laboratory, Cambridge, MA (May 1965).

[Tei85] Teicholz, E., *CAD/CAM Handbook*, McGraw-Hill, Inc., New York (1985).

[Tsi82] Tsichritzis, D.C. and Lochovsky, F.H., *Data Models*, Prentice-Hall, Inc., Edgewood Cliffs, NJ (1982).

[Ulf82a] Ulfsby, S., Meen, S., and Oian, J., "Tornado: A DBMS for CAD/CAM systems," pp. 335-346 in *File structures and data bases for CAD*, ed. J. Encarnacao, North Holland Publishing Company, Amsterdam (1982).

[Ulf82b] Ulfsby, S., Meen, S., and Oian, J., "Tornado: A data base management system for graphics applications," in *IEEE Computer graphics and automation* (May 1982).

[Ull80] Ullman, J.D., *Principles of database systems*, Computer Science Press, Potomac, MD (1980).

[Voe] Voelcker, H., Requicha, A., Hartquist, E., Fisher, W., Metzger, J., Tilove, R., Birrell, N., Hunt, W., Armstrong, G., Check, T., Moote, R., and Sweeney, J. Mc, *The PADL-1.0/2 system for defining and displaying solid objects*, Production Automotion Project, The University of Rochester, Rochester, NY.

[Web83] Weber, H., "Object-oriented DDBS design," pp. 196-221 in *Proceedings of 2nd international conference on databases*, ed. S.M. Dean, Wiley Heyden, Cambridge, England (1983).

[Wel76] Weller, D. and Williams, R., "Graphic and relational data base support for problem solving," in *Proceedings of 3rd annual conference on computer graphics. interactive techniques and image processing*, Philadelphia, PA (1976).

[Wel79] Weller, D., *Software support for graphical interaction,* IBM Research Laboratory, San Jose, CA (October 1979).

[Win75] Winston, P.H., *The psychology of computer vision,* McGraw, Inc., New York (1975).

[Woo83] Woo, T.C., "Interfacing solid modeling to CAD and CAM: Data structures and algorithms for decomposing a solid," Technical Report 83-6, University of Michigan, Ann Arbor, MI (1983).

[Zan86a] Zaniolo, C., Ait-Kaci, H., Beech, D., Cammarata, S., Kerschberg, L., and Maier, D., "Object-oriented database systems and knowledge systems," pp. 50-65 in *Expert database systems*, ed. L. Kerschberg, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

[Zan86b] Zaniolo, C., "Prolog: A database query language for all seasons," pp. 219-232 in *Expert database systems*, ed. L. Kerschberg, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1986).

[Zuc85] Zucherman, M., June 1985, March 1986. Personal communication.

[Zuc86] Zucherman, M. I., *A knowledge base development for producibility feedback in mechanical design,* Masters thesis, UCLA (1986).

APPENDIX A

ABBREVIATIONS AND ACRONYMS

AI	Artificial intelligence
APPAS	Automatic Process Planning and Selection
BCSS	Boeing Computing Support System
BOM	Bill of materials
B-rep	Boundary representation
CAD	Computer-aided design
CAE	Computer-aided engineering
CAM	Computer-aided manufacturing
CCA	Computer Corporation of America
CCDBMS	CAD/CAM DBMS
CIMS	Computer integrated manufacturing system
CNC	Computer numerical control
CPL	Computerized Parts List
CSG	Constructive solid geometry
DBA	Data base administrator
DBMS	Data base management system
DDL	Data definition language
DML	Data manipulation language
DNC	Direct numerical control
E-R	Entity-relationship

FMS	Flexible manufacturing systems
GPM	Geometric product model
HICLASS	Hughes Integrated Classification
IDB	Integrated Data Base
IGES	Initial graphics exchange specification
KBMS	Knowledge base management system
MCL	Master Component Libraries
MML	Model manipulation language
NC	Numerical control
ODM	Object Data Model
OEL	Object entry language
OML	Object manipulation language
PCB	Printed circuit board
PDDS	Product Design Data System
PF	Producibility Feedback
PIR	Production Inspection Record
PL	Parts List
PWA	Printed wiring assembly
PWB	Printed wiring board
SAM	Semantic Association Model
VHDL	VHSIC Hardware Description Language
VHSIC	Very high speed integrated circuits
VLSI	Very large scale integration

APPENDIX B

OML SYNTAX

Defining intensions

```
(send db def-intension <intension> <opt-props>)
(send <superpart-intension> def-subpart <intension> <opt-props>)
(send <superclass-intension> def-subclass <intension> <opt-props>)

(send <superpart-intension> def-subpart-intension <intension> <opt-props>)
(send <superclass-intension> def-subclass-intension <intension> <opt-props>)
```

Defining instances

```
(send <intension> def-instance <instance>)
(send <superpart-instance> def-subpart <instance>)
(send <intension> def-subpart-instance <instance> <superpart-instance>)
(send <superpart-instance> def-subpart-instance <instance> <intension>)

(send <superpart-intension> set-subpart-qty <subpart-intension> <qty>)
(send <superpart-intension> get-subpart-qty <subpart-intension>)
(send <superpart-instance> set-subpart-qty <subpart-instance> <qty>)
(send <superpart-instance> get-subpart-qty <subpart-instance>)
```

Querying intensions and instances

```
(send db is-intension? <obj>)
(send db is-instance? <obj>)

(send <intension> get-specializations)
(send <intension> get-all-specializations)

(send <intension> get-generalizations)
(send <intension> get-all-generalizations)
```

```
(send <intension> get-subparts)
(send <intension> get-all-subparts)

(send <intension> get-superparts)
(send <intension> get-all-superparts)

(send <intension> get-instantiations)
(send <intension> get-all-instantiations)

(send <instance> get-parts)
(send <instance> get-all-parts)

(send <instance> get-assemblies)
(send <instance> get-all-assemblies)

(send <instance> get-intension)
(send <instance> get-all-intensions)

(send <intension> is-specialization? <intension>)
(send <instance> is-instantiation? <intension>)
(send <intension> is-subpart? <intension>)
(send <instance> is-part? <instance>)
```

Defining and querying properties

```
(send <intension> def-property <propname>)

(send <intension> get-properties)
(send <instance> get-properties)

(send <intension> get-all-properties)
(send <instance> get-all-properties)

(send <intension> is-property? <propname>)
(send <instance> is-property? <propname>
```

Setting and retrieving property values

```
(send <intension> set-property-slot <propname> <slotname> <slotvalue>)

(send <intension> get-property-slots)
(send <instance> get-property-slots)

(send <intension> get-property-slot <propname> <slotname>)
(send <instance> get-property-slot <propname> <slotname>)
```

(send <instance> set-property-value <propname> <propvalue>)
(send <instance> get-property-value <propname>)

(send <intension> get-all-instances-where <propname> <propvalue>)

Displaying intensions, instances, properties

(send <intension> show-self)
(send <instance> show-self)
(send <intension> show-self-in-detail)
(send <instance> show-self-in-detail)
(send <intension> show-property <propname>)
(send <instance> show-property-value <propname>)

Using extensions

(send <intension> get-extension)
(send db is-extension? <obj>)

(send <extension> get-members)
(send <extension> get-all-members)

(send <instance> get-extension)
(send <instance> get-all-extensions)
(send <extension> get-subextensions)
(send <extension> get-all-subextensions)

(send <extension> get-superextensions)
(send <extension> get-all-superextensions)

(send <instance> is-member? <extension>)
(send <extension> is-subextension? <extension>)

Defining relations

(send db def-relation-intension <relation> <opt-roles>)

(send <relation> def-argument <role>)
(send <relation> set-argument-lambda <role-name> <lambda-exp>)

(send <relation> def-relation-instance)

(send <relation-instance> set-argument-value <role-value>)
(send <relation-instance> def-relation-instance

<argument-name/argument-value pairs>)

Quering relations

(send <relation> get-arguments)
(send <relation> get-argument-lambda <role>)
(send <relation> get-instantiations)
(send <relation-instance> get-argument-value <argument>)

APPENDIX C

OUTPUT OF OEL PARSING

```
(send db def-intension VEHICLE)
(send db def-intension DWELLING)
(send VEHICLE def-subclass-intension MOTOR-HOME)
(send DWELLING def-subclass-intension MOTOR-HOME)
(send VEHICLE def-subclass-intension AUTOMOBILE)
(send AUTOMOBILE def-subclass-intension HONDA)
(send AUTOMOBILE def-subclass-intension CADILLAC)
(send db def-intension ENGINE)
(send AUTOMOBILE def-subpart ENGINE)
(send db def-intension BODY)
(send AUTOMOBILE def-subpart BODY)
(send db def-intension FENDER)
(send BODY def-subpart FENDER)
(send ENGINE def-subclass-intension HONDA-ENGINE)
(send MOTOR-HOME def-instance MOTOR-HOME08)
(send CADILLAC def-instance CADILLAC06)
(send HONDA def-instance HONDA03)
(send HONDA-ENGINE def-instance HONDA03-ENGINE)
(send HONDA03 def-subpart HONDA03-ENGINE)
(send BODY def-instance HONDA03-BODY)
(send HONDA03 def-subpart HONDA03-BODY)
(send FENDER def-instance HONDA03-FENDER)
(send HONDA03-BODY def-subpart HONDA03-FENDER)
(send BODY def-instance CADILLAC06-BODY)
(send CADILLAC06 def-subpart CADILLAC06-BODY)
```

APPENDIX D

BILL OF MATERIALS DATA FOR PWA M87706172

QTY	PART NUMBER	REF DESIG	XORG	YORG	XOFF	YOFF	OREN	CL	LIB REF	LEN	DIA	LEAD
3	M60985/94-7380	C003	3.325	1.550	0.200	0.000	0.	0	A001	.160	.090	.027
	M60985/94-7380	C004	2.750	0.450	0.250	0.000	0.	0	A001	.160	.090	.027
	M60985/94-7380	C008	3.125	1.900	0.350	0.000	0.	0	A001	.160	.090	.027
5	M60985/94-2691	C001	3.125	1.700	0.350	0.000	0.	0	A003	.390	.140	.027
	M60985/94-2691	C002	1.625	3.100	0.000	0.350	90.	0	A003	.390	.140	.027
	M60985/94-2691	C005	5.125	-0.075	0.000	0.350	90.	0	A003	.390	.140	.027
	M60985/94-2691	C006	4.950	1.300	0.000	0.350	90.	0	A003	.390	.140	.027
	M60985/94-2691	C007	5.125	1.400	0.000	0.350	90.	0	A003	.390	.140	.027
1	M55302/61-1-24	P002	5.363	4.850			0	1				
1	M55302/61-A-20	P003	0.138	0.650			0	1				
1	M55302/61-A-36	P001	5.363	2.400			0	1				
1	JANTX2N2222A	Q001	4.100	3.550	0.075	0.000	0.	3	C007	TO-18		
1	JANTX2N3767	Q002	4.300	4.750	0.100	-0.100	270	3	C019	TO-66		
1	M39015/3-005WM	R044	2.300	1.700	0.100	0.000	0.	4				
20	RCR07G103JM	R002	0.675	1.200	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R005	0.675	1.050	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R007	0.675	0.900	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R009	0.675	0.750	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R012	0.675	0.600	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R013	0.675	0.450	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R014	0.675	0.300	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R017	1.325	0.300	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R019	1.325	0.450	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R021	1.325	0.600	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R024	1.325	0.750	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R026	1.325	0.900	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R027	1.325	1.050	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R028	1.325	1.200	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R029	1.325	1.350	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G103JM	R031	0.225	3.900	0.250	0.000	0.	0	A049	.281	.098	.027

	RCR07G103JM	R033	4.950	0.025	0.000	0.250	90.	0	A049	.281	.098	.027
	RCR07G103JM	R034	4.700	0.025	0.000	0.250	90.	0	A049	.281	.098	.027
	RCR07G103JM	R035	4.525	0.025	0.000	0.250	90.	0	A049	.281	.098	.027
	RCR07G103JM	R036	4.275	0.025	0.000	0.250	90.	0	A049	.281	.098	.027
1	RCR07G220JM	R048	3.825	3.125	0.250	0.000	0.	0	A049	.281	.098	.027
3	RCR07G363JM	R053	4.175	1.275	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G363JM	R054	4.175	1.500	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G363JM	R055	4.175	1.700	0.250	0.000	0.	0	A049	.281	.098	.027
4	RCR07G391JM	R032	3.825	2.875	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G391JM	R037	3.825	3.250	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G391JM	R038	3.825	2.750	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G391JM	R039	3.825	3.000	0.250	0.000	0.	0	A049	.281	.098	.027
1	RCR07G510JM	R051	4.175	2.225	0.250	0.000	0.	0	A049	.281	.098	.027
23	RCR07G511JM	R001	3.025	5.325	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R003	3.025	5.175	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R004	3.025	5.025	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R006	3.025	4.875	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R008	3.025	4.725	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R010	3.025	5.600	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R011	3.025	5.475	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R015	3.025	4.275	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R016	3.025	4.125	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R018	3.025	3.950	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R020	3.025	3.800	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R022	3.025	3.650	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R023	3.025	4.575	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R025	3.025	4.425	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R045	1.975	1.825	0.000	0.250	90.	0	A049	.281	.098	.027
	RCR07G511JM	R056	0.925	5.200	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R057	0.925	5.050	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R058	0.925	4.900	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R059	0.925	4.750	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R060	0.925	4.600	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R061	0.925	4.450	0.250	0.000	0.	0	A049	.281	.098	.027
	RCR07G511JM	R062	0.925	4.300	0.250	0.000	0.	0	A049	.281	.098	.027

	RCR07G511JM	R063	0.925	4.150	0.250	0.000	0.	0	A049	.281	.098	.027
1	RCR32G241JM	R047	4.600	2.650	0.000	0.450	90.	0	A051	.593	.240	.045
1	RNC60H1071FM	R042	2.300	0.800	0.000	0.350	90.	0	A054	.437	.165	.027
1	RNC60H1181FM	R030	1.975	2.725	0.000	0.350	90.	0	A054	.437	.165	.027
1	RNC60H2003FM	R052	3.125	1.175	0.350	0.000	0.	0	A054	.437	.165	.027
1	RNC60H3012FM	R040	2.425	0.675	0.350	0.000	0.	0	A054	.437	.165	.027
1	RNC60H4992FM	R043	3.050	1.375	0.350	0.000	0.	0	A054	.437	.165	.027
1	RNC60H4993FM	R041	3.125	0.950	0.350	0.000	0.	0	A054	.437	.165	.027
4	M38510/00101BCB	U003	0.400	2.975	-0.150	0.300	90.	1	D002	14-PIN DIP		
	M38510/00101BCB	U004	1.050	2.975	-0.150	0.300	90.	1	D002	14-PIN DIP		
	M38510/00101BCB	U006	0.625	1.975	-0.150	0.300	90.	1	D002	14-PIN DIP		
	M38510/00101BCB	U007	1.450	1.975	-0.150	0.300	90.	1	D002	14-PIN DIP		
2	M38510/01009BCB	U001	2.525	5.500	0.150	-0.300	270	1	D002	14-PIN DIP		
	M38510/01009BCB	U005	2.525	4.650	0.150	-0.300	270	1	D002	14-PIN DIP		
1	M38510/11301BEB	U008	2.125	3.600	-0.150	0.350	90.	1				
1	12293337	U009	0.075	5.175	0.300	-0.550	270	1				
1	12296004	U002	2.125	4.900	-0.150	0.350	90.	1				
1	12296005	U010	3.275	3.100	0.150	-0.300	270	1	D002	14-PIN DIP		
2	M38510/10104BGA	AR001	2.337	3.213	0.110	0.000	270	3	C030	TO-100		
	M38510/10104BGA	AR002	2.807	1.231	0.110	0.000	180	3	C030	TO-100		
3	JANTX1N485B	CR001	4.000	0.550	-0.300	0.000	180	0	A038	.300	.130	.022
	JANTX1N485B	CR002	4.000	0.375	-0.300	0.000	180	0	A038	.300	.130	.022
	JANTX1N485B	CR003	4.000	0.200	-0.300	0.000	180	0	A038	.300	.130	.022
2	JANTX1N3019B	VR001	4.875	2.950	0.000	0.350	90.					
	JANTX1N3019B	VR002	5.175	3.650	0.000	-0.350	270	0				

APPENDIX E

COMPONENT PHYSICAL DATA FOR PWA M87706172

PART NUMBER	REF DESIG	XORG	YORG	XOFF	YOFF	OREN	CL	LIB REF	LEN	DIA	TOL	LEAD	MILITARY SPECIFICATION
M38510/00101	U003	0.400	2.975	-0.150	0.300	90.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/001
M38510/00101	U004	1.050	2.975	-0.150	0.300	90.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/001
M38510/00101	U006	0.625	1.975	-0.150	0.300	90.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/001
M38510/00101	U007	1.450	1.975	-0.150	0.300	90.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/001
M38510/01009	U001	2.525	5.500	0.150	-0.300	270.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/010
M38510/01009	U005	2.525	4.650	0.150	-0.300	270.	1	DIP14	M-001AA	@	@	0.023	MIL-S-38510/010
M38510/10104	AR001	2.325	3.210	0.110	0.000	0.	3	TO-99	TO-99	@	@	0.023	MIL-S-38510/101
M38510/10104	AR002	2.810	1.225	0.110	0.000	0.	3	TO-99	TO-99	@	@	0.023	MIL-S-38510/101
M38510/11301	U008	2.125	3.600	-0.150	0.350	90.	1	DIP16	M-001AC	@	@	0.023	MIL-S-38510/113
12293337	U009	0.075	5.175	0.300	-0.550	270.	1	DIP24	M-001AG	@		@ 0.023@	
12296004	U002	2.125	4.900	-0.150	0.350	90.	1	DIP14	M-001AA	@		@ 0.023@	
12296005	U010	3.275	3.100	0.150	-0.300	270.	1	DIP14	M-001AA	@		@ 0.023@	
JANTX1N3019B	VR001	4.875	2.950	0.000	0.350	90.	0	DIOD05	0.460	0.265	@	0.033	MIL-S-19500/115
JANTX1N3019B	VR002	5.175	3.650	0.000	-0.350	270.	0	DIOD05	0.460	0.265	@	0.033	MIL-S-19500/115
JANTX1N485B	CR001	4.000	0.550	-0.300	0.000	180.	0	DIOD02	0.300	0.130	@	0.022	MIL-S-19500/118
JANTX1N485B	CR002	4.000	0.375	-0.300	0.000	180.	0	DIOD02	0.300	0.130	@	0.022	MIL-S-19500/118
JANTX1N485B	CR003	4.000	0.200	-0.300	0.000	180.	0	DIOD02	0.300	0.130	@	0.022	MIL-S-19500/118
JANTX2N2222A	Q001	4.090	3.550	0.085	0.000	0.	3	TO-18	TO-18	@	@	0.021	MIL-S-19500/255
JANTX2N3767	Q002	4.300	4.750	0.100	-0.100	270.	3	TO-66	TO-66	@	@	0.034	MIL-S-19500/518
M60985/94-7380	C003	3.325	1.550	0.200	0.000	0.	0	CKR11	0.160	0.090	@	0.027	MIL-C-60985/94
M60985/94-7380	C004	2.750	0.450	0.250	0.000	0.	0	CKR11	0.160	0.090	@	0.027	MIL-C-60985/94
M60985/94-7380	C008	3.125	1.900	0.350	0.000	0.	0	CKR11	0.160	0.090	@	0.027	MIL-C-60985/94
M60985/94-2691	C001	3.125	1.700	0.350	0.000	0.	0	CKR14	0.390	0.140	@	0.027	MIL-C-60985/94
M60985/94-2691	C002	1.625	3.100	0.000	0.350	90.	0	CKR14	0.390	0.140	@	0.027	MIL-C-60985/94
M60985/94-2691	C005	5.125	-0.075	0.000	0.350	90.	0	CKR14	0.390	0.140	@	0.027	MIL-C-60985/94
M60985/94-2691	C006	4.950	1.300	0.000	0.350	90.	0	CKR14	0.390	0.140	@	0.027	MIL-C-60985/94
M60985/94-2691	C007	5.125	1.400	0.000	0.350	90.	0	CKR14	0.390	0.140	@	0.027	MIL-C-60985/94
RCR07G103JM	R012	0.675	0.600	0.250	0.000	0.	0	RCR07	0.281	0.098	@	0.027	MIL-R-39008/1
RCR07G103JM	R013	0.675	0.450	0.250	0.000	0.	0	RCR07	0.281	0.098	@	0.027	MIL-R-39008/1
RCR07G103JM	R014	0.675	0.300	0.250	0.000	0.	0	RCR07	0.281	0.098	@	0.027	MIL-R-39008/1

RCR07G103JM	R017	1.325	0.300	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R019	1.325	0.450	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R002	0.675	1.200	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R021	1.325	0.600	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R024	1.325	0.750	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R026	1.325	0.900	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R027	1.325	1.050	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R028	1.325	1.200	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R029	1.325	1.350	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R031	0.225	3.900	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R033	4.950	0.025	0.000	0.250	90.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R034	4.700	0.025	0.000	0.250	90.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R035	4.525	0.025	0.000	0.250	90.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R036	4.275	0.025	0.000	0.250	90.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R005	0.675	1.050	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R007	0.675	0.900	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G103JM	R009	0.675	0.750	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G220JM	R048	3.825	3.125	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G363JM	R053	4.175	1.275	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G363JM	R054	4.175	1.500	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G363JM	R055	4.175	1.700	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G391JM	R032	3.825	2.875	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G391JM	R037	3.825	3.250	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G391JM	R038	3.825	2.750	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G391JM	R039	3.825	3.000	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G510JM	R051	4.175	2.225	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R001	3.025	5.325	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R010	3.025	5.600	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R011	3.025	5.475	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R015	3.025	4.275	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R016	3.025	4.125	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R018	3.025	3.950	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R020	3.025	3.800	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R022	3.025	3.650	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R023	3.025	4.575	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1

RCR07G511JM	R025	3.025	4.425	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R003	3.025	5.175	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R004	3.025	5.025	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R045	1.975	1.825	0.000	0.250	90.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R056	0.925	5.200	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R057	0.925	5.050	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R058	0.925	4.900	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R059	0.925	4.750	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R006	3.025	4.875	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R060	0.925	4.600	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R061	0.925	4.450	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R062	0.925	4.300	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R063	0.925	4.150	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR07G511JM	R008	3.025	4.725	0.250	0.000	0.	0	RCR07	0.281	0.098 @ 0.027	MIL-R-39008/1
RCR32G241JM	R047	4.600	2.650	0.000	0.450	90.	0	RCR32	0.593	0.240 @ 0.045	MIL-R-39008/3
RNC60H1071FM	R042	2.300	0.800	0.000	0.350	90.	0	RNC60	0.437	0.165 @ 0.027	MIL-R-55182/3
RNC60H1181FM	R030	1.975	2.725	0.000	0.350	90.	0	RNC60	0.437	0.165 @ 0.027	MIL-R-55182/3
RNC60H2003FM	R052	3.125	1.175	0.350	0.000	0.	0	RNC60	0.437	0.165 @ 0.027	MIL-R-55182/3

APPENDIX F

COMPONENT ELECTRICAL DATA FOR PWA M87706172

REF DESIG	DEVICE	PART NUMBER	DESCRIPTION	VALUE	TOL	RATE	LIB REF	MILITARY SPECIFICATION
C003	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	A001	MIL-C-60985/94
C004	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	A001	MIL-C-60985/94
C008	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	A001	MIL-C-60985/94
C001	CK14-.1	M60985/94-2691	CAP,CERAM	0.1	@10%	50V	A003	MIL-C-60985/94
C002	CK14-.1	M60985/94-2691	CAP,CERAM	0.1	@10%	50V	A003	MIL-C-60985/94
C005	CK14-.1	M60985/94-2691	CAP,CERAM	0.1	@10%	50V	A003	MIL-C-60985/94
C006	CK14-.1	M60985/94-2691	CAP,CERAM	0.1	@10%	50V	A003	MIL-C-60985/94
C007	CK14-.1	M60985/94-2691	CAP,CERAM	0.1	@10%	50V	A003	MIL-C-60985/94
P001								
P002								
P003								
Q001	2N2222	JANTX2N2222A	TRANSISTOR				C007	MIL-S-19500/225
Q002	2N3767	JANTX2N3767	TRANSISTOR				C019	MIL-S-19500/518
R044	RT24-200	M39015/3-005WM	RES,VARIABLE,WW	200				MIL-R-30915/3
R002	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R005	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R007	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R009	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R012	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R013	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R014	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R017	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R019	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R021	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R024	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R026	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R027	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R028	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R029	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R031	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1

R033	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R034	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R035	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R036	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	A049	MIL-R-39008/1
R048	RC07-22	RCR07G220JM	RES,FIXED,COMP	22	@5%	1/4W	A049	MIL-R-39008/1
R053	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	A049	MIL-R-39008/1
R054	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	A049	MIL-R-39008/1
R055	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	A049	MIL-R-39008/1
R032	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	A049	MIL-R-39008/1
R037	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	A049	MIL-R-39008/1
R038	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	A049	MIL-R-39008/1
R039	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	A049	MIL-R-39008/1
R051	RC07-51	RCR07G510JM	RES,FIXED,COMP	51	@5%	1/4W	A049	MIL-R-39008/1
R001	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R003	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R004	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R006	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R008	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R010	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R011	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R015	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R016	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R018	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R020	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R022	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R023	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R025	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R045	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R056	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R057	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R058	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R059	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R060	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R061	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R062	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1

R063	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	A049	MIL-R-39008/1
R047	RC32-240	RCR32G241JM	RES,FIXED,COMP	240	@5%	1W	A051	MIL-R-39008/3
R042	RN60-1.07K	RNC60H1071FM	RES,FIXED,FILM	1.07K	@1%	1/8W	A054	MIL-R-55182/3
R030	RN60-1.18K	RNC60H1181FM	RES,FIXED,FILM	1.18K	@1%	1/8W	A054	MIL-R-55182/3
R052	RN60-200K	RNC60H2003FM	RES,FIXED,FILM	200K	@1%	1/8W	A054	MIL-R-55182/3
R040	RN60-30.1K	RNC60H3012FM	RES,FIXED,FILM	30.1K	@1%	1/8W	A054	MIL-R-55182/3
R043	RN60-49.9K	RNC60H4992FM	RES,FIXED,FILM	49.9K	@1%	1/8W	A054	MIL-R-55182/3
R041	RN60-499K	RNC60H4993FM	RES,FIXED,FILM	499K	@1%	1/8W	A054	MIL-R-55182/3
U003	7430	/00101BCB	INTEGRATED CCT				D002	MIL-S-38510/001
U004	7430	/00101BCB	INTEGRATED CCT				D002	MIL-S-38510/001
U006	7430	/00101BCB	INTEGRATED CCT				D002	MIL-S-38510/001
U009	MCM5303	12293337	INTEGRATED CCT					
U002	74184	12296004	INTEGRATED CCT					
U010	CD4041	12296005	INTEGRATED CCT					D002
AR001	108	/10104BGA	INTEGRATED CCT				C030	MIL-S-38510/101
AR002	108	/10104BGA	INTEGRATED CCT				C030	MIL-S-38510/101
CR001	1N485	JANTX1N485B	DIODE,SIGNAL,GP				A038	MIL-S-19500/118
CR002	1N485	JANTX1N485B	DIODE,SIGNAL,GP				A038	MIL-S-19500/118
CR003	1N485	JANTX1N485B	DIODE,SIGNAL,GP				A038	MIL-S-19500/118
VR001	1N3019	JANTX1N3019B						
VR002	1N3019	JANTX1N3019B						

APPENDIX G

TEST INFORMATION FOR PWA M87706172

REF DESIG	DEVICE	PART NUMBER	DESCRIPTION	VALUE	TOL	RATE	LIB REF	MILITARY SPECIFICATION
AR001	108	M38510/10104	AMPLIFIER	@	@	@	TO-99	MIL-S-38510/101
AR002	108	M38510/10104	AMPLIFIER	@	@	@	TO-99	MIL-S-38510/101
VR001	1N3019	JANTX1N3019B	DIODE,ZENER,9.1V	@	@	@	DIOD05	MIL-S-19500/115
VR002	1N3019	JANTX1N3019B	DIODE,ZENER,9.1V	@	@	@	DIOD05	MIL-S-19500/115
CR001	1N5614	JANTX1N5614	DIODE,SIGNAL,GP	@	@	@	DIOD02	MIL-S-19500/427
CR002	1N5614	JANTX1N5614	DIODE,SIGNAL,GP	@	@	@	DIOD02	MIL-S-19500/427
CR003	1N485	JANTX1N485B	DIODE,SIGNAL,GP	@	@	@	DIOD02	MIL-S-19500/118
Q001	2N2222	JANTX2N2222A	XSTR, GP	@	@	@	TO-18	MIL-S-19500/255
Q002	2N3767	JANTX2N3767	XSTR, POWER	@	@	@	TO-66	MIL-S-19500/518
U002	74184	12296004	INTEGRATED CIRCUIT	@	@	@	DIP14	@
U003	7430	M38510/00101	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/001
U004	7430	M38510/00101	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/001
U006	7430	M38510/00101	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/001
U007	7430	M38510/00101	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/001
U001	7449	M38510/01009	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/010
U005	7449	M38510/01009	INTEGRATED CIRCUIT	@	@	@	DIP14	MIL-S-38510/010
U010	CD4041	12296005	INTEGRATED CIRCUIT	@	@	@	DIP14	@
C003	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	CKR11	MIL-C-60985/94
C004	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	CKR11	MIL-C-60985/94
C008	CK11-100PF	M60985/94-7380	CAP,CERAM	100PF	@10%	100V	CKR11	MIL-C-60985/94
C001	CK14-.1	M60985/94-2691	CAP,CERAM	.1UF	@10%	50V	CKR14	MIL-C-60985/94
C005	CK14-.1	M60985/94-2691	CAP,CERAM	.1UF	@10%	50V	CKR14	MIL-C-60985/94
C006	CK14-.1	M60985/94-2691	CAP,CERAM	.1UF	@10%	50V	CKR14	MIL-C-60985/94
C007	CK14-.1	M60985/94-2691	CAP,CERAM	.1UF	@10%	50V	CKR14	MIL-C-60985/94
U008	DAC08	M38510/11301	INTEGRATED CIRCUIT	@	@	@	DIP16	MIL-S-38510/113
U009	MCM5303	12293337	INTEGRATED CIRCUIT	@	@	@	DIP24	@
R012	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R013	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R014	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R017	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1

R019	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R002	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R021	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R024	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R026	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R027	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R028	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R029	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R031	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R033	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R034	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R035	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R036	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R005	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R007	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R009	RC07-10K	RCR07G103JM	RES,FIXED,COMP	10K	@5%	1/4W	RCR07	MIL-R-39008/1
R048	RC07-22	RCR07G220JM	RES,FIXED,COMP	22	@5%	1/4W	RCR07	MIL-R-39008/1
R053	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	RCR07	MIL-R-39008/1
R054	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	RCR07	MIL-R-39008/1
R055	RC07-36K	RCR07G363JM	RES,FIXED,COMP	36K	@5%	1/4W	RCR07	MIL-R-39008/1
R032	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	RCR07	MIL-R-39008/1
R037	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	RCR07	MIL-R-39008/1
R038	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	RCR07	MIL-R-39008/1
R039	RC07-390	RCR07G391JM	RES,FIXED,COMP	390	@5%	1/4W	RCR07	MIL-R-39008/1
R051	RC07-51	RCR07G510JM	RES,FIXED,COMP	51	@5%	1/4W	RCR07	MIL-R-39008/1
R001	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R010	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R011	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R015	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R016	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R018	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R020	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R022	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R023	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R025	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1

R003	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R004	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R045	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R056	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R057	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R058	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R059	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R006	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R060	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R061	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R062	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R063	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R008	RC07-510	RCR07G511JM	RES,FIXED,COMP	510	@5%	1/4W	RCR07	MIL-R-39008/1
R047	RC32-240	RCR32G241JM	RES,FIXED,COMP	240	@5%	1W	RCR32	MIL-R-39008/3
R042	RN60-1.07K	RNC60H1071FM	RES,FIXED,FILM	1.07K	@1%	1/8W	RNC60	MIL-R-55182/3
R030	RN60-1.18K	RNC60H1181FM	RES,FIXED,FILM	1.18K	@1%	1/8W	RNC60	MIL-R-55182/3
R052	RN60-200K	RNC60H2003FM	RES,FIXED,FILM	200K	@1%	1/8W	RNC60	MIL-R-55182/3
R040	RN60-30.1K	RNC60H3012FM	RES,FIXED,FILM	30.1K	@1%	1/8W	RNC60	MIL-R-55182/3
R043	RN60-49.9K	RNC60H4992FM	RES,FIXED,FILM	49.9K	@1%	1/8W	RNC60	MIL-R-55182/3
R041	RN60-499K	RNC60H4993FM	RES,FIXED,FILM	499K	@1%	1/8W	RNC60	MIL-R-55182/3
R044	RT24-200	M39015/3-005WM	RES,VAR,WIREWOUN	200	@5%	3/4W	RT24	MIL-R-39015/3

APPENDIX H

REFERENCE INFORMATION FOR PWA M87706172

TITLE	VALUE
MAXIMUM COMPONENT HEIGHT	.650
BOARD THICKNESS	.062
MAXIMUM LEAD PROTRUSION	
MINIMUM COMPONENT SPACING	
USED ON	12293301
NEXT ASSEMBLY	12293600
SCHEMATIC DIAGRAM	M12293828
MIN SPACE: CLNCHD LD/CKT	
MAXIMUM BOW AND TWIST	

APPENDIX I

ENGINEERING NOTES FOR PWA M87706172

1. SEE M12293828 FOR SCHEMATIC DRAWING

2. TEST BOARD PER 12296024

6. ASSEMBLE AND SOLDER BOARD PER SD3589037

7. TORQUE SCREWS TO 6-8 INCH-POUNDS *Q2

8. IDENTIFY BOARD SERNO PER MIL-STD-130; CHARACTER 0.06 INCH HIGH.
AIR CURE OR HEAT CURE AT 140 F MAX.

9. PRIME AND SEAL THREADS USING MATERIAL PER MIL-S-22473 GRADE C *Q2,P1,P2,P3

10. TRIM LEADS ON FARSIDE OF BOARD; TO 0.040 INCHES *C5,P3

11. BOND SPACERS USING MATERIAL PER 12296065 TYPE 2 CLASS A.
REMOVE BONDING MATERIAL INSIDE SPACER AND HOLE

13. CONFORMAL COAT BOARD PER 12296050; EXCLUDING TOP SIDE AREAS SHOWN;
USING MATERIAL PER MIL-I-46058 TYPE UR

14. HANDLE COMPONENT USING STATIC SAFE PRECAUTIONS PER HP10-39;
EXCEPT PARAGRAPHS 3.2.9, 3.4.6.4, 3.4.7, 3.4.9, 3.4.10 AND 3.4.11
*AR1,AR2,U2,U8,U10

15. SLEEVE PER HP14-23 COVERING BODY AND LEADS *VR1,VR2

16. ACCEPTABLE ALTERNATES FOR M38510/01009BCB ARE UA5449DMQB 5449/883B, SNJ54LS49J
*U1,U5

17. ACCEPTABLE ALTERNATE FOR M38510/11301BEB IS 12296035 *U8

APPENDIX J

IGES BOARD OUTLINE FOR PWA M87706172

```
M87706172                                                              S      1
BOARD OUTLINE                                                          S      2
1H,,1H;,,22HMFVS.3827.IGES.OUTLINE,45HCOMPUTERVISION.REV 11.00.CADDS GRAG      1
PHIC SYSTEM,14HIGES REV 01.00,16,08,24,08,56,,201.8000,1,4HINCH,,,13H831G      2
2 7, 94609,,,,8H72-24-33;                                              G      3
     124       1       1                                     000000    D      1
     124                      3                                        D      2
     124       4       1                                     000000    D      3
     124                      3                                        D      4
     124       7       1                                     000000    D      5
     124                      3                                        D      6
     124      10       1                                     000000    D      7
     124                      3                                        D      8
     124      13       1                                     000000    D      9
     124                      3                                        D     10
     124      16       1                                     000000    D     11
     124                      3                                        D     12
     110      19       1      1                       1      000000    D     13
     110                      2                                        D     14
     110      21       1      1                       1      000000    D     15
     110                      2                                        D     16
     110      23       1      1                       1      000000    D     17
     110                      2                                        D     18
     110      25       1      1                       1      000000    D     19
     110                      2                                        D     20
124,      1.000000,      0.000000,      0.000000,      0.000000,      1P      1
          0.000000,      1.000000,      0.000000,      0.000000,      1P      2
          0.000000,      0.000000,      1.000000,      0.000000;      1P      3
124,      1.000000,      0.000000,      0.000000,      0.000000,      3P      4
          0.000000,      0.000000,     -1.000000,      0.000000,      3P      5
          0.000000,      1.000000,      0.000000,      0.000000;      3P      6
124,      0.000000,      0.000000,      1.000000,      0.000000,      5P      7
          1.000000,      0.000000,      0.000000,      0.000000,      5P      8
          0.000000,      1.000000,      0.000000,      0.000000;      5P      9
```

```
124,     1.000000,    0.000000,    0.000000,    0.000000,       7P     10
         0.000000,   -1.000000,    0.000000,    0.000000,       7P     11
         0.000000,    0.000000,   -1.000000,    0.000000;       7P     12
124,     0.000000,    0.000000,   -1.000000,    0.000000,       9P     13
        -1.000000,    0.000000,    0.000000,    0.000000,       9P     14
         0.000000,    1.000000,    0.000000,    0.000000;       9P     15
124,    -1.000000,    0.000000,    0.000000,    0.000000,      11P     16
         0.000000,    0.000000,    1.000000,    0.000000,      11P     17
         0.000000,    1.000000,    0.000000,    0.000000;      11P     18
110,    -0.250000,   -0.250000,    0.000000,   -0.250000,      13P     19
         5.750000,    0.000000;                                13P     20
110,    -0.250000,    5.750000,    0.000000,    5.750000,      15P     21
         5.750000,    0.000000;                                15P     22
110,     5.750000,    5.750000,    0.000000,    5.750000,      17P     23
        -0.250000,    0.000000;                                17P     24
110,     5.750000,   -0.250000,    0.000000,   -0.250000,      19P     25
        -0.250000,    0.000000;                                19P     26
S      2G      3D      20P     26                                  T      1
```

APPENDIX K

IGES TOP VIEW OF PWA M87706172

```
M87706172                                                                S      1
P1                                                                       S      2
TOP VIEW                                                                 S      3
1H,,1H;,,17HMFVS.3827.IGES.P1,45HCOMPUTERVISION.REV 11.00.CADDS GRAPHIC G      1
SYSTEM,14HIGES REV 01.00,16,08,24,08,56,,201.8000,1,4HINCH,,,13H8312 7, G      2
94141,,,,8H72-24-33;                                                     G      3
     124       1       1                                        000000 D      1
     124                       3                                       D      2
     124       4       1                                        000000 D      3
     124                       3                                       D      4
     124       7       1                                        000000 D      5
     124                       3                                       D      6
     124      10       1                                        000000 D      7
     124                       3                                       D      8
     124      13       1                                        000000 D      9
     124                       3                                       D     10
     124      16       1                                        000000 D     11
     124                       3                                       D     12
     110      19       1       1                       1        000000 D     13
     110                       2                                       D     14
     110      21       1       1                       1        000000 D     15
     110                       2                                       D     16
     110      23       1       1                       1        000000 D     17
     110                       2                                       D     18
     110      25       1       1                       1        000000 D     19
     110                       2                                       D     20
     110      27       1       1                       1        000000 D     21
     110                       2                                       D     22
     110      29       1       1                       1        000000 D     23
     110                       2                                       D     24
     110      31       1       1                       1        000000 D     25
     110                       2                                       D     26
     110      33       1       1                       1        000000 D     27
     110                       2                                       D     28
```

```
     110      35      1      1                     1       000000  D      29
     110                     2                                     D      30
     110      37      1      1                     1       000000  D      31
     110                     2                                     D      32
     110      39      1      1                     1       000000  D      33
     110                     2                                     D      34

124,      1.000000,     0.000000,     0.000000,     0.000000,        1P      1
          0.000000,     1.000000,     0.000000,     0.000000,        1P      2
          0.000000,     0.000000,     1.000000,     0.000000;        1P      3
124,      1.000000,     0.000000,     0.000000,     0.000000,        3P      4
          0.000000,     0.000000,    -1.000000,     0.000000,        3P      5
          0.000000,     1.000000,     0.000000,     0.000000;        3P      6
124,      0.000000,     0.000000,     1.000000,     0.000000,        5P      7
          1.000000,     0.000000,     0.000000,     0.000000,        5P      8
          0.000000,     1.000000,     0.000000,     0.000000;        5P      9
124,      1.000000,     0.000000,     0.000000,     0.000000,        7P     10
          0.000000,    -1.000000,     0.000000,     0.000000,        7P     11
          0.000000,     0.000000,    -1.000000,     0.000000;        7P     12
124,      0.000000,     0.000000,    -1.000000,     0.000000,        9P     13
         -1.000000,     0.000000,     0.000000,     0.000000,        9P     14
          0.000000,     1.000000,     0.000000,     0.000000;        9P     15
124,     -1.000000,     0.000000,     0.000000,     0.000000,       11P     16
          0.000000,     0.000000,     1.000000,     0.000000,       11P     17
          0.000000,     1.000000,     0.000000,     0.000000;       11P     18
110,      5.533000,     2.664000,     0.000000,     5.554000,       13P     19
          2.668000,     0.000000;                                   13P     20
110,      5.554000,     2.668000,     0.000000,     5.572000,       15P     21
          2.678000,     0.000000;                                   15P     22
110,      5.572000,     2.678000,     0.000000,     5.586000,       17P     23
          2.695000,     0.000000;                                   17P     24
110,      5.586000,     2.695000,     0.000000,     5.593000,       19P     25
          2.714000,     0.000000;                                   19P     26
110,      5.593000,     2.714000,     0.000000,     5.593000,       21P     27
          2.736000,     0.000000;                                   21P     28
```

```
110,     5.593000,    2.736000,    0.000000,    5.586000,    23P    29
         2.755000,    0.000000;                              23P    30
110,     5.586000,    2.755000,    0.000000,    5.573000,    25P    31
         2.771000,    0.000000;                              25P    32
110,     5.573000,    2.771000,    0.000000,    5.554000,    27P    33
         2.782000,    0.000000;                              27P    34
110,     5.554000,    2.782000,    0.000000,    5.534000,    29P    35
         2.785000,    0.000000;                              29P    36
110,     5.534000,    2.785000,    0.000000,    5.513000,    31P    37
         2.782000,    0.000000;                              31P    38
110,     5.513000,    2.782000,    0.000000,    5.495000,    33P    39
         2.772000,    0.000000;                              33P    40
110,     5.495000,    2.772000,    0.000000,    5.481000,    35P    41
```

APPENDIX L

MCL COMPONENT DETAILS

COMP-PART-NUM COMP-DESC	COMP-STATUS	COMP-GROUP	COMP-TYPE
CMR04F101JPDL CAPACITOR			CAPACITOR
CMR06F102FODL CAPACITOR			CAPACITOR
CMR06F472JODM CAPACITOR			CAPACITOR
CMR03C3RODOCM			
D2272044			
DG303AP 14-PIN-DIP			DIP
JANIN4100 DIODE			DIODE
JANIN4105 DIODE			DIODE
JANIN4150-1 DIODE			DIODE
JANIN4454 DIODE			DIODE
JANIN4454 DIODE			DIODE
JANIN4954 DIODE			DIODE

COMP-PART-NUM COMP-DESC	COMP-STATUS	COMP-GROUP	COMP-TYPE
DIODE			
JANIN645 DIODE			DIODE
JANTX1N3019B DIODE			DIODE
JANTX1N3600 DIODE			DIODE
JANTX1N485B DIODE			DIODE
JANTX1N5804 DIODE			DIODE
JANTX1N6074 DIODE			DIODE
JANTX1N645-1 DIODE			
JANTX1N746A DIODE			DIODE
JANTX1N755A DIODE			DIODE
JANTX1N964B DIODE			DIODE
JANTX2N2222A SWITCH, HIGH SPEED			TO

APPENDIX M

MCL COMPONENT PART DESCRIPTIONS

PART NUMBER	POLARITY	TOL+	TOL-	COMP-VALUE	SEQUENCE-ID	VALUE-UOM ST	X-OFFSET	Y-OFFSET
M55302/61-A-36	0	0	0				0	0
12272054	0	0	0				0	0
3500.	0	0	0				1450	1450
RCR07G820JM	0	0	5	82		OHMS	0	0
M38510/30808BEB	0	0	0				0	0
NAS871C4	0	0	0				0	0
JANIN4100	0	0	0				0	0
M14933-770580EX								
M38510/05001BCB	0	0	0				0	0
RWR81S1R69FM	0	0	1	1.69		OHMS	0	0
JANTX2N2222A	0	0	0				0	0
M38510/08101BCB	0	0	0				0	0

PART NUMBER / POLARITY	TOL+	TOL-	COMP-VALUE / SEQUENCE-ID	VALUE-UOM	ST	X-OFFSET	Y-OFFSET
RCR07G681JM / 0	0	0				0	0
M39014/22-014							
RNC65H1004FM / 0	0	0				0	0
RCR07G380JM / 0	0	0				0	0
RNC50H1912FM / 0	0	0				0	0
NAS1102E04-6 / 0	0	0				0	0
M6106/28-023 / 0	0	0				0	0
NAS620C4L / 0	0	0				0	0
M55302/61-1-24 / 0	0	0				0	0
M38510/05101BCB / 0	0	0				0	0
M39014/22-178 / 0	0	0				0	0
M38510/01009BCB / 0	0	0				0	0

APPENDIX N

MCL PAD PATTERN DATA

PAD-PATTERN#	PAD-SIZE	DELTA-X	DELTA-Y
20	250	2000	0
25	250	0	0
25	250	10500	0
26	250	0	0
26	250	750	-750
26	250	1250	0
28	250	0	0
28	250	1000	-1000
28	250	2000	0
30	250	0	0
30	250	293	707
30	250	1000	1000
30	250	1707	707
30	250	2000	0
30	250	1707	-707
30	250	1000	-1000

PAD-PATTERNB	PAD-SIZE	DELTA-X	DELTA-Y
31	250	1707	707
31	250	1707	-707
31	250	1000	-1000
31	250	293	-707
32	250	0	0
32	250	213	676
32	250	795	1094
32	250	1505	1094
32	250	2087	676
32	250	2300	0
32	250	213	-676
32	250	795	-1094
32	250	1505	-1094
32	250	2087	-676
36	250	0	0
44	250	0	0

APPENDIX O

MCL CASE STYLE DATA

PART-NUM	PAD-PATTERN#	PAD-SPAN	NOM-LENGTH	NOM-WIDTH	NOM-HEIGHT	NO-OF-PINS	WEIGHT	WEIGHT-UOM	NOM-LEAD-DIAM
CMR06F102FODL	00	0	5600	1100	5200	2	0		320
CMR06F472JODM	00	0	6100	2200	2200	2	0	GRAMS	5800
CMR03C3RODOCM	00	0	2700	1100	1900	2	0		3400
D2272044	00	0	8400	2650	1400	16	0		1850
DG303AP	44	3000	7500	2500	0	14	0		0
JANIN4100	00	0	2650	1075	1075	20	0		200
JANIN4105	00	0	2650	1075	1075	2	0		200
JANIN4150-1	00	0	3000	2110	2110	20	0		200
JANIN4454	00	0	3000	1250	1250	20	0		200
JANIN4454	00	0	3000	1450	1250	20	0		200
JANIN4954	00	0	3500	1450	1450	20	0		400
JANIN4960	00	0	3500	1450	1450	2	0		400

PART-NUM	PAD-PATTERN#	PAD-SPAN	NOM-LENGTH	NOM-WIDTH	NOM-HEIGHT	NO-OF-PINS	WEIGHT	WEIGHT-UOM	NOM-LEAD-DIAM
JANIN645	00	0	2725	1085	1085	200	0		200
JANTX1N3019B	04	7000	5600	2650	2650	2	0		320
JANTX1N3600	04	7000	3000	900	900	2	0		200
JANTX1N485B	02	5000	3000	1400	1400	2	0		200
JANTX1N5804	01	4000	1500	750	750	2	0		290
JANTX1N6074	01	4000	1520	660	660	2	0		290
JANTX1N645-1	02	5000	3000	1400	1400	2	0		220
JANTX1N746A	02	5000	3000	1400	1400	2	0		200
JANTX1N755A	02	5000	3000	1400	1400	2	0		200
JANTX1N964B	02	5000	3000	1400	1400	2	0		200
JANTX2N2222A	26	1000	2190	2190	2500	3	100	OZ.	190
JANTX2N2907A	26	1000	2190	2190	2500	3	100	OZ.	190

APPENDIX P

OEL SPECIFICATION OF PF DATA

```
(i new_part |detail_part|
  size_x_axis: 5.000
  size_y_axis: 2.500
  size_z_axis: 3.000
  part_volume: 20.5
  material: aluminum
  original_form: casting
  original_form_x_axis: 5.165
  original_form_y_axis: 2.625
  original_form_z_axis: 3.165
  number_of_holes: 6
  number_of_surfaces: 8)

(i old_part |detail_part|
  size_x_axis: 5.000
  size_y_axis: 2.500
  size_z_axis: 3.000
  part_volume: 20.5
  material: aluminum
  original_form: casting
  original_form_x_axis: 5.165
  original_form_y_axis: 2.625
  original_form_z_axis: 3.165
  number_of_holes: 6
  number_of_surfaces: 8)

(i new_part_draw_form |draw_form|
  detail_part: new_part
  designer: clark
  revisions: rev_a
  block_tolerance: .001
  project: demo
  program: ucla)
```

```
(i new_part_datum |datum|
  detail_part: new_part
  primary_datum: s4
  secondary_datum: s7
  tertiary_datum: s5
  ref_datum_a: ha
  ref_datum_b: hb
  ref_datum_c: hd)

(i new_hole_data |hole|
  detail_part: new_part
  ent_surface: s1
  exit_surface: s2
  int_x_geo: s1
  diameter: .5
  dia_tol: .001
  bottom_cond: flat
  surface_cond: .001
  tap_size: 4-48
  pos_tol: 0.01)

(i old_hole_data |hole|
  detail_part: old_part
  ent_surface: s1
  exit_surface: s1
  int_x_geo: s3
  diameter: .3750
  dia_tol: .001
  bottom_cond: flat
  surface_cond: .001
  tap_size: 8-32
  pos_tol: 0.01)

(i hole_a_data |hole|
  detail_part: new_part
  ent_surface: s5
  exit_surface: s3
  int_x_geo: s1
  diameter: .5
  dia_tol: .001
  bottom_cond: thru
  surface_cond: .001
  tap_size: 3-56
  pos_tol: 0.001)
```

```
(i hole_b_data |hole|
  detail_part: new_part
  ent_surface: s5
  exit_surface: s3
  int_x_geo: s1
  diameter: .125
  dia_tol: .001
  bottom_cond: thru
  surface_cond: .001
  tap_size: 3-56
  pos_tol: 0.001)

(i hole_c_data |hole|
  detail_part: new_part
  ent_surface: s5
  exit_surface: s3
  int_x_geo: s1
  diameter: .125
  dia_tol: .001
  bottom_cond: thru
  surface_cond: .001
  tap_size: 3-56
  pos_tol: 0.001)

(i hole_d_data |hole|
  detail_part: new_part
  ent_surface: s5
  exit_surface: s3
  int_x_geo: s1
  diameter: .125
  dia_tol: .001
  bottom_cond: thru
  surface_cond: .001
  tap_size: 3-56
  pos_tol: 0.001)
```

```
(i hole_e_data |hole|
  detail_part: new_part
  ent_surface: s5
  exit_surface: s3
  int_x_geo: s1
  diameter: .125
  dia_tol: .001
  bottom_cond: thru
  surface_cond: .001
  tap_size: 10-24
  pos_tol: 0.001)

(i hole_f_data |hole|
  detail_part: new_part
  ent_surface: s2
  exit_surface: null
  int_x_geo: s6
  diameter: .5
  dia_tol: .001
  bottom_cond: flat
  surface_cond: .001
  tap_size: 10-24
  pos_tol: 0.001)

(i new_hole_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 2.0
  x_start_ref_surface: s1
  x_end_loc: 1.0
  x_end_ref_surface: s1
  y_start_loc: 1.2
  y_start_ref_surface: s1
  y_end_loc: -.4
  y_end_ref_surface: s2
  z_start_loc: 0.0
  z_start_ref_surface: s2
  z_end_loc: -3.0
  z_end_ref_surface: s2)
```

```
(i old_hole_ref |hole_ref|
  detail_part: old_part
  x_start_loc: 3.0
  x_start_ref_surface: s2
  x_end_loc: 1.5
  x_end_ref_surface: s2
  y_start_loc: .7
  y_start_ref_surface: s2
  y_end_loc: 0.0
  y_end_ref_surface: s3
  z_start_loc: 3.0
  z_start_ref_surface: s3
  z_end_loc: -3.0
  z_end_ref_surface: s1)

(i hole_a_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 3.0
  x_start_ref_surface: null
  x_end_loc: 3.0
  x_end_ref_surface: null
  y_start_loc: 1.25
  y_start_ref_surface: s7
  y_end_loc: 1.25
  y_end_ref_surface: s7
  z_start_loc: 0
  z_start_ref_surface: null
  z_end_loc: -.75
  z_end_ref_surface: s5)

(i hole_b_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 1.5
  x_start_ref_surface: s5
  x_end_loc: 1.5
  x_end_ref_surface: s5
  y_start_loc: .5
  y_start_ref_surface: s7
  y_end_loc: .5
  y_end_ref_surface: s4
  z_start_loc: 0
  z_start_ref_surface: null
  z_end_loc: -.75
  z_end_ref_surface: s5)
```

```
(i hole_c_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 1.5
  x_start_ref_surface: s5
  x_end_loc: 1.5
  x_end_ref_surface: s5
  y_start_loc: 2.0
  y_start_ref_surface: s5
  y_end_loc: 2.0
  y_end_ref_surface: s5
  z_start_loc: 0
  z_start_ref_surface: s5
  z_end_loc: -.75
  z_end_ref_surface: s5)

(i hole_d_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 4.5
  x_start_ref_surface: s6
  x_end_loc: 4.5
  x_end_ref_surface: s6
  y_start_loc: 2.0
  y_start_ref_surface: s6
  y_end_loc: 2.0
  y_end_ref_surface: s6
  z_start_loc: 0.0
  z_start_ref_surface: s6
  z_end_loc: -.75
  z_end_ref_surface: s5)

(i hole_e_ref |hole_ref|
  detail_part: new_part
  x_start_loc: 4.5
  x_start_ref_surface: s7
  x_end_loc: 4.5
  x_end_ref_surface: s7
  y_start_loc: .5
  y_start_ref_surface: s7
  y_end_loc: .5
  y_end_ref_surface: s7
  z_start_loc: 0.0
  z_start_ref_surface: s7
  z_end_loc: -.75
  z_end_ref_surface: s5)
```

```
(i hole_f_ref |hole_ref|
  detail_part: new_part
  x_start_loc: .25
  x_start_ref_surface: s7
  x_end_loc: .75
  x_end_ref_surface: s8
  y_start_loc: 1.25
  y_start_ref_surface: s8
  y_end_loc: 1.25
  y_end_ref_surface: s8
  z_start_loc: 2.0
  z_start_ref_surface: s8
  z_end_loc: 2.0
  z_end_ref_surface: s2)

(i s1 |surface|
  detail_part: new_part
  resident_plane: (x y)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: no
  fillet_radius: .025
  corner_radius: .020
  type_of_surface: cast
  surface_finish: .001
  number_of_intersecting_holes: 0)

(i s2 |surface|
  detail_part: new_part
  resident_plane: (y z)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: no
  fillet_radius: .020
  corner_radius: .018
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 1)
```

```
(i s3 |surface|
  detail_part: new_part
  resident_plane: (x y)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: no
  fillet_radius: .020
  corner_radius: .028
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 5)

(i s4 |surface|
  detail_part: new_part
  resident_plane: (y z)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: a
  fillet_radius: .022
  corner_radius: .020
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 0)
```

```
(i s5 |surface|
  detail_part: new_part
  resident_plane: (x y)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: c
  fillet_radius: .018
  corner_radius: .025
  type_of_surface: cast
  surface_finish: .001
  number_of_intersecting_holes: 0)

(i s6 |surface|
  detail_part: new_part
  resident_plane: (y z)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: no
  fillet_radius: .020
  corner_radius: .030
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 0)
```

```
(i s7 |surface|
  detail_part: new_part
  resident_plane: (x z)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: b
  fillet_radius: .018
  corner_radius: .020
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 0)

(i s8 |surface|
  detail_part: new_part
  resident_plane: (x z)
  x_bounding_plane_xy: ()
  y_bounding_plane_xy: ()
  x_bounding_plane_xz: ()
  z_bounding_plane_xz: ()
  y_bounding_plane_yz: ()
  z_bounding_plane_yz: ()
  datum_plane: no
  fillet_radius: .050
  corner_radius: 0.030
  type_of_surface: mach
  surface_finish: .001
  number_of_intersecting_holes: 0)
```