2

AL-TR-1992-0109

AD-A261 512

ARMSTRONG LABORATORY

DESIGN KNOWLEDGE MANAGEMENT SYSTEM (DKMS) TECHNOLOGY IMPACT REPORT

Richard J. Mayer
Thomas M. Blinn
David C. Browne
Matthew A. Grisius
Arthur A. Keen
Jeffery C. Lockledge
Les Sanders

KNOWLEDGE BASED SYSTEMS, INC.
2726 LONGMIRE
COLLEGE STATION, TX 77845-5424

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION

NOVEMBER 1992

FINAL TECHNICAL REPORT FOR PERIOD NOVEMBER 1990 - JUNE 1992

AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573

NOTICES

[illegible]

JOANN M. SARTOR
Program Manager

[illegible]

BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1992	3. REPORT TYPE AND DATES COVERED Final - November 1990 - June 1992

4. TITLE AND SUBTITLE Design Knowledge Management System (DKMS) Technology Impact Report	5. FUNDING NUMBERS C - F33615-90-C-001 PE - 65502F PR - 3005 TA - L2 WU - 03
6. AUTHOR(S) Richard J. Mayer, Arthur A. Keen, Matthew A. Grisius, Thomas M. Blinn, Jeffery C. Lockledge, David C. Browne, Les Sanders	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Knowledge Based Systems, Inc. 2746 Longmire College Station, TX 77845-5424	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Armstrong Laboratory Human Resources Directorate Logistics Research Division Wright-Patterson AFB, OH 45433-6573	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AL-TR-1992-0109

11. SUPPLEMENTARY NOTES
Armstrong Laboratory Technical Monitor: Capt JoAnn Sartor, (513) 255-5775. This research was conducted under the Small Business Innovation Research (SBIR) Program as a Phase II effort

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Technology Impact Report, prepared under a Phase II Small Business Innovation Research (SBIR) effort, documents key concepts of the Design Knowledge Management System (DKMS) as it impacts current design technology. A technology overview is given for the major DKMS components. The DKMS consists of components which can be used to streamline facets of the design process: Container Object System: a flexible means of storing computer objects and constraints, so that a design can easily evolve as information changes. Model Design Support Environment: a method for automatic solution and code generation based on the problem, the "givens", and the real-world equations that rule the environment. High-Productivity CAD: a toolbox for CAD designers. Shape-based Design Knowledge Representation and Reasoning: component which allows shape information as a key factor in the design. (For example: "Does the shape of this part create unacceptable stress points?") Integration Platform: component which helps user select and translate services.

14. SUBJECT TERMS artificial intelligence automatic code generation	CAD systems design computer-aided design concurrent engineering design	design evolution S3IR program shape-based reasoning	15. NUMBER OF PAGES 36
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

Table of Contents

List of Figures

List of Acronyms

CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CAM	Computer-Aided Manufacturing
CE	Concurrent Engineering
CIM	Computer-Integrated Manufacturing
COS	Container Object System
CSG	Constructive Solid Geometry
DKMS	Design Knowledge Management System
DOM	Design Object Manager
ECAD	Electronic Computer-Aided Design
FAE	Finite Element Analysis
GCSG	Generalized Constructive Solid Geometry
HPCAD	High Productivity Computer-aided Design
IP	Integration Platform
ISM	Integration Services Manager
MDSE	Model Development Support Environment
NURBS	Non Uniform Rational B-Spline
SBDKRR	Shape-based Design Knowledge Representation

Preface

This DKMS Technology Impact Report (Contract Data Requirements List, Sequence Number 10) presents the key concepts of DKMS and its major components as they impact current design technology. The roles and underlying concepts of DKMS and its five major components are outlined in this report. The five components are:

1. Integration Platform,
2. Shaped-Based Design Knowledge Representation and Reasoning,
3. High Productivity Computer-Aided Design,
4. Container Object System, and
5. Model Development Support Environment.

Each section presents a technology overview of one of these components, including its functionality, key concepts, and motivation as they relate to the overall DKMS strategy. A final subsection for each component summarizes its contribution to DKMS or its impact on current or future technology.

1.0 DKMS Overview

Henry Ford established a collection of inventions, architectural exhibits, and common devices that the average person might consider eccentricity in the extreme. However, what is now known as the Henry Ford museum might better be considered the first serious attempt to assemble a large technology knowledge base. This facility provided designers, product engineers, and manufacturing engineers with an environment in which they could browse design evolution across a wide variety of innovations. The essence of DKMS for concurrent engineering (CE) is a modern version of this concept. It includes a processable representation of what it means to design (Design Ontology), a design rule base of stable design rules (Design Base), and a repository for design objects (1).

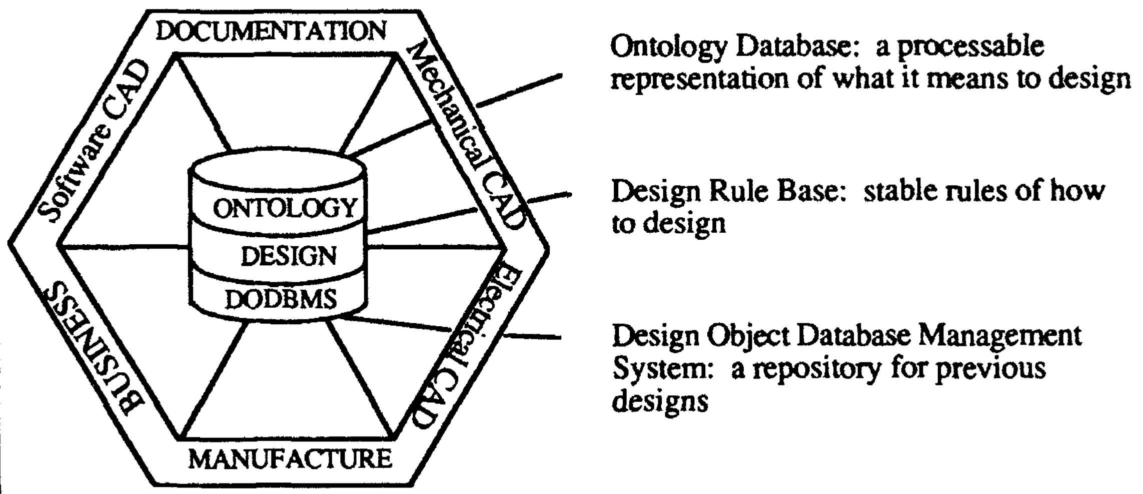
DOCUMENTATION
Mechanical CAD
Electrical CAD
MANUFACTURE
BUSINESS
Software CAD
ONTOLOGY
DESIGN
DODBMS
Ontology Database: a processable representation of what it means to design
Design Rule Base: stable rules of how to design
Design Object Database Management System: a repository for previous designs

Figure 1. Design Knowledge Management Architecture

The key integration philosophy applied in DKMS is that info⁻..ation that flows between different activities (opportunities for concurrent integration) in the product life cycle is made possible by constraints or relations between these different activities. DKMS focuses on overall system problems rather than individual ones. This focus differs from general efforts in knowledge-asset management systems in that it attempts to blend or merge existing computer-integrated manufacturing (CIM) capabilities.

1.1 DKMS Motivation

In the 1970s and early 1980s, a crisis was recognized in the United States: The growth rate of industrial productivity and the quality of American-made products were slipping. This gave rise to increased investment in automation and modernization of manufacturing equipment, facilities, systems, organizations, and labor management. In the late 1980s, the focus shifted to problems in the worldwide competitive position. This forced the U.S. to acknowledge that many of the previous problems were actually symptoms of deep-rooted issues that encompassed the entire corporation--from marketing and finance through research, engineering, manufacturing, and field support. The following are among the key issues contributing to the erosion of the worldwide competitive position of the United States:

1. long lead-times to take a concept to rate production,
2. ineffective technology transfer,
3. lack of a total systematic approach to quality management,
4. retiring or otherwise lost knowledge base, and
5. high rates of change in the product definition.

In short, DKMS is part of the answer to the questions, "Why does it take so long to design/engineer/produce a product? Why are so many past mistakes repeated? Why are we always the last to use the new ideas and technology produced by our own research laboratories?" DKMS focuses on the problems associated with acquisition, management, and effective delivery of engineering knowledge, experience, and rationale. This life-cycle engineering knowledge must be marshaled to achieve the CE and total quality management goals.

1.2 DKMS Impact on Technology

Design knowledge management is distinguished from general efforts in knowledge-asset management and attempts to merge existing CIM capabilities by:

1. its focus on engineering knowledge,
2. its heterogeneous form and the distributed nature of such an engineering knowledge base,
3. its focus on shape as the primary indexing and organization mechanism of such engineering knowledge, and
4. its need for consideration of knowledge acquisition, application, and evolution as well as storage and retrieval.

The prototype DKMS being built was designed from the start to handle 1) life-cycle engineering knowledge representation, 2) design knowledge acquisition support, 3) design knowledge storage, 4) design knowledge retrieval, and 5) design knowledge and information integration services. Many attempts at CIM and CE have failed because the capability to evolve (systems and data) was not a design goal or objective but merely an afterthought.

The core of DKMS consists of a container object representation that facilitates the evolution of life-cycle artifact definitions and the flow of information between different life-cycle activities concerning the life-cycle artifacts. Evolution is facilitated by using a service-based integration platform as a transport layer. Knowledge representation is populated using shape-based knowledge acquisition and reasoning module interfaces in addition to an engineering model development support module. The display and modeling capabilities are supported by an open geometry toolbox that includes geometric data format translators, surface translators, a generalized constructive solid geometry (GCSG) engine, and form feature geometry input interfaces to allow rapid construction of geometric integration services. The toolbox is also used by the shape-based knowledge acquisition module to capture shape-based design knowledge.

DKMS research appears to have predicted the current software panacea of client-server-type applications. The trend is to build small manageable modules that are easily and cheaply integrated into a tools framework that satisfies individual requirements rather than the monolithic software monsters with the "do-all" mentality of yesteryear. Almost every major player in the software market, as well as some hardware manufacturers, is touting a software architecture to integrate applications and data. The major drawback is that most of these attempts must design, engineer, and build these applications from scratch using this new technology. However, DKMS proposes an architecture that integrates existing legacy applications and data into an integrated homogeneous access format. While this level of integration can vary widely within DKMS, it is essential that DKMS offer an inexpensive and less risky way to experiment with CE by providing a legacy application integration path.

By the final stages of Phase II, the project expectations, results, and technologies greatly exceeded initial expectations. Not only were the required deliverables completed on time and on budget, but additional materials that contribute to the overall DKMS project were produced. These accomplishments include the following.

1. Detailed design documents were completed, including software requirements specifications documents and user manuals for DKMS and all its subsystems.
2. The major DKMS components were manufactured, engineered, and unit-tested.
3. Beta-test information was gathered from as many sources as possible using numerous collection methods to maximize potential benefits.
4. Some key subsystems and technologies were integrated into other environments and projects to further their utilization and continue their development and refinement.
5. A marketing direction and interest group was developed for some key DKMS subsystems and technologies.

There were also many potential pitfalls and barriers that have been managed during the Phase II development, including the following.

1. DKMS compatibility with the evolving standards was maintained.
2. In-depth knowledge of government and private initiatives in integration schemes and mechanisms was maintained.
3. The difficulty and apparent apprehension of integrating and testing systems in-situ with "legacy" production systems were addressed.
4. The relative newness and acceptance of CE concepts were addressed.

5. A project with the scope of DKMS requires an enterprise-wide cooperational effort that would require inter- and intracompany-wide planning and implementation—a multi-year effort.

6. While design is "generic," each company, site, or department has different cultures, philosophies, methodologies, and tools; thus, a true directed and meaningful beta test is virtually impossible within the given time frame.

7. Some DKMS technologies are quite diverse and forward-thinking. Evaluation tasks typically require extensive training.

2.0 Integration Platform (IP) Overview

Under the philosophy, the integration of engineering applications is viewed as an opportunity to provide greater function to the environment by supplying new services and resources. If protocols and interfaces can be defined that allow other tools to take advantage of the functionality provided by the new applications, powerful integrated environments for system or product development could result. This service and resource-based approach is directed toward making such environments possible. With a service-based approach to information integration, an application simply advertises the services it will provide and their invocation procedures. In essence, the advertisements define external interfaces that allow other tools or users to take advantage of the functionality provided by the new application. IP provides the support necessary to organize and maintain these interface definitions as well as to route the integration service requests.

2.1 IP Motivation

Despite today's open systems with a plethora of available standards, application and data integration standards are not being addressed. Rather than focusing on the construction of an "integrated system," the services-based approach to information integration focuses on the "integration services" that the IP and functional applications will provide. In previous approaches, the burden was assumed to be on the platform to provide the integration support desired (1,2,3). This traditional approach has severe problems. One problem is the need to define comprehensive standards in advance and build applications that meet these standards. This presumes that an organization can foresee 1) the integration services that will be required, and 2) the relative demands for those services. Presuming that the former could be predicted but the latter could not be determined, the traditional integration approaches force a leveling of integration support across all needs. This implies a massive overhaul of existing legacy systems and unjustifiable modifications by vendors of existing systems to achieve even minimal integration support.

The flexibility of the information integration mechanism is important when considering the overall CE process. Previous integration efforts have focused on a particular domain and the way elements within that domain interact (4,5). This vertical integration, while important and useful, does not address the way that various domains (e.g., marketing, manufacturing, design) interact.

2.2 IP Impact on Technology

The IP is designed to be a distributed, heterogeneous environment for the integration of software applications and management of design process artifacts. IP operation revolves around the generation, management, and execution of service plans. The main concept in the integration services approach is the "service." A "service" is a functional unit that performs a service for a requesting agent (software module or user). Normally, services are operations that integrate two or more existing computer-aided design (CAD), computer-aided engineering (CAE), or computer-aided manufacturing (CAM) tools, though this is not a requirement.

Service advertisements describe sets of service protocols that have been defined between different classes of data formats. Each service advertisement bridges two format classes and captures all operations that can occur between the two format classes. When the IP attempts to generate a plan, it scans the list of service advertisements to determine if there is a path from the source format class to the destination format class. If a path is found, the Integration Services Manager (ISM) examines the service protocols associated with the service advertisement to determine which service to invoke.

Service contracts define the information necessary to execute a service. These contracts specify to the ISM what information must be collected to invoke the utility and how to a perform the invocation once the information is collected. These service contracts represent actual executable utilities and organize information about these utilities into four units of information: the utility, the argument specifications, the data specifications, and the invocation structure.

However, a representation for the registration of services, is not enough. The IP must have a strategy or plan for manipulating these service representations. A service plan is simply a sequence of operations that, when performed, provide the requested service. A service plan can exist as a functional or executable plan. The functional plan is a sequence of service identifiers that provide the requested service. It is independent of any hardware or software implementation details. True "planning" occurs in the generation of the functional plan where the object is to find a sequence of services that will provide the requested service. This process is accomplished by examining the service advertisements and protocols. An executable plan, produced only when a functional plan is generated successfully, is a sequence of machine operations derived from the service contracts that will actually be executed to provide the requested service.

Operationally, every machine that is part of the IP will have an implementation of the ISM running. Each ISM will have access to information about all available services and the ability to interpret service requests, build executable service plans based on those requests, execute the service plans, and, finally, collect and return the results of the service plans.

Figure 2 shows the operation of the ISM and the interaction of key ISM components. The key components include the service registration tool, the plan builder, the plan executor/monitor, and the message decoder/encoder.

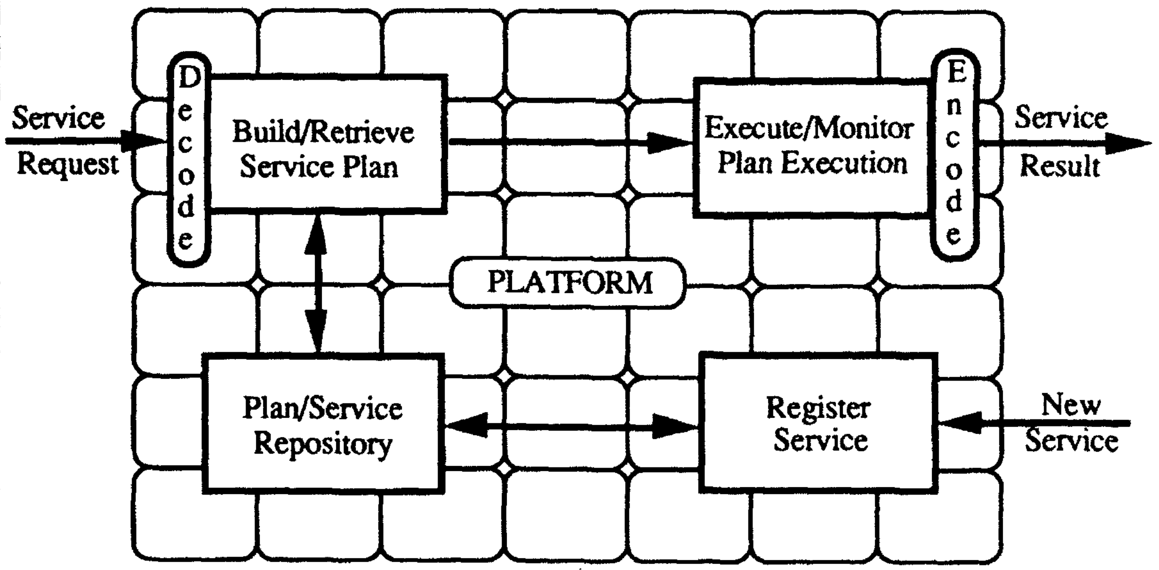

Figure 2. Operation of the Integration Services Manager

The Design Object Manager (DOM) is responsible for managing design artifacts; thus, the DOM provides functionality for registering data artifacts in the repository and maintaining

Following this simple analogy, design artifacts managed by the DOM can be checked out by users; however, these artifacts must first be catalogued. During the cataloging process, the DOM analyzes the version and configuration management constraints of the artifacts to determine if any configuration management operations are required.

The Integration Services approach to integration provides considerable room for flexibility. As new tools become available, they can be loaded directly onto the IP. If these tools are to be integrated into the development process, their integration services are registered with the ISM. Once this has been accomplished, the tool is completely integrated into the development environment.

The Integration Services approach, a powerful--yet flexible--means of integrating systems, focuses on the advertisement and execution of integration services provided by tools. The approach is aimed at using a standard protocol for interfacing and invoking tools to provide an environment that will effectively integrate legacy tools from many different vendors. Although this approach was defined to integrate systems in a CE environment, applying its concepts to software engineering can be as beneficial to software development as it is to engineering.

3.0 Shape-Based Design Knowledge Representation and Reasoning (SBDKRR) Overview

The SBDKRR component of DKMS provides the capability to represent and reason about design knowledge that contains arbitrary geometric structures (generalized structures). The SBDKRR system can represent multiple compatible descriptions of part geometry through a mechanism known as a frame of reference. The frame of reference is the mapping-to-reality definition perceived by the agent or individual interacting with the part and the constraints to which the agent or individual is attuned relative to the part and its perceived environment. The frame of reference includes a granularity specification, a directional system, and a set of spatial prepositions. The granularity specification defines how the agent perceives the part (i.e., resolution, dimensions, etc.) and consists primarily of mapping functions such as equality, adjacency, and containment; in other words, the meaning of *adjacency* and *equality* in this agent's view of the world. The directional system defines data that is used by the agent as reference points. The spatial prepositions consist of the name and definition concepts (such as *between, top, below,* and *on)* that the individual may use to describe the arrangement of geometric primitives perceived in the frame of reference. The reasoning strategy employed by the SBDKRR module consists of the following three phases:

1. a recognition process that extracts elementary geometric patterns from the product definition,
2. a prototype matcher that assigns prototypes to the geometric patterns, and
3. an interpretation phase in which the implications of the prototypes are generated.

3.1 SBDKRR Motivation

The part representation used by conventional CAD systems (including solid modeling) is incomplete in terms of the semantic and qualitative part descriptions needed for CE (6). However, for effective CE applications, one would like to identify problem areas in a design and provide recommendations interactively during the design creation process (7). This problem of impoverished part representation has inhibiting effects on the cost-efficient application of CAE. For example, a hole will be represented as a cylindrical face connected to the faces around it in a solid modeler. This representation is not suitable for grid refinement algorithms used in finite element analysis (FEA) that need to know that the hole is a stress raiser and, consequently, needs a finer mesh. Also, the representation does not show that the cylinder represents a hole that may be mapped by a process planning system to drilling, reaming, or boring operations; or that the hole presents problems to manufacturing because of the way it interacts with other features.

3.2 SBDKRR Impact on Technology

A key issue in shape-based knowledge representation architecture is whether the representation should be incorporated into or maintained external to the CAD system. The choice of strategy hinges on the available CAD system. In the literature, both strategies have been attempted; the choice of strategy should depend on the complexity of the knowledge stored. Clearly, for parametric-dimension-driven design systems and single-abstraction spatial reasoners, it would be better to integrate the representation *into the* CAD system. However, for large, complex knowledge bases, it is advantageous to separate the representation from the CAD system to improve maintainability and allow for its applicabil'ty with multiple CAD systems. Without exception, the external-shape representation strategy was employed where the available CAD system was either closed or the data structures employed by the CAD system were unsuitable for integrating with a knowledge representation. The choice here is to: 1) support a variety of CAD systems by designing a representation that includes geometric and topological relations and Boolean operations, and 2) build a system that interfaces to CAD systems that have an open architecture (i.e., an architecture that allows users access to the data structures and functions of the system). DKMS is unique in providing both a geometric representation capability and an interfacing method to open-architecture CAD systems. The geometric representation is supported through a COS-based implementation of SBDKRR concepts. CAD system integration is accomplished with the services provided by the IP and High Productivity CAD (HPCAD) Toolkit.

The reasoning strategy employed by the SBDKRR module consists of three phases: 1) Recognition, 2) Pattern Matching, and 3) Interpretation. The geometric algorithms necessary for the Recognition Phase consist of: 1) predicates such as *coplanar, intersect,* and *convex;* 2) derived properties such as *area, perimeter,* and *volume;* and 3) constructions such as *convex hull, minimum enclosing box,* and *Boolean operations.* The prototype Matching Phase uses graph algorithms such as articulation, biconnected and triconnected components, and depth first search (8). To support the Interpretation Phase, artificial intelligence inferencing algorithms have been modified for use with these geometric algorithms. These inferencing methods include: abduction (9), truth maintenance, and constraint propagation (7). The combination of these methods plus the pattern-matching and shape-recognition SBDKRR components creates a geometric reasoning capability unique to DKMS.

4.0 High Productivity Computer-Aided Design (HPCAD) Overview

One set of services for which the IP was developed is the translation of geometry from one representation technique or parameterization to another and the translation of geometry from one data representation to another. The HPCAD Toolkit bridges these representational discontinuities by providing a convenient means of creating services for translations, and performing Boolean operations on surfaces that have been created in different modeling systems and from different parameterizations. The HPCAD Toolkit also facilitates the creation of customized CAD systems and can be used in a supporting role (such as in a shape-based reasoning application) in which the Toolkit routines may be used as predicates.

The Toolkit will support viewing and manipulation of Bézier and NURBS surfaces, solids via polygonal boundary representations (both winged-triangle and winged-edge representations), and constructive solid geometry (CSG) as well as hybrid surface/solid representations using GCSG. *Solids may be modified using Euler operators and* CSG. The Toolkit will support the algebraic (exact) translation between parametric and approximate translations. It will also directly support data exchange between standard data formats such as Initial Graphics Exchange Specifications, Data Exchange Format, and Product Data Exchange.

4.1 HPCAD Motivation

To support design engineers in the rapid development of product specifications and engineering prototypes, a design environment must:

1. use a solid modeling representation and organization that is isomorphic to the representations used to perceive and structure the design situation;

2. support the rapid entry of a design concept or intent into the CAD system;

3. intelligently set default dimensions, spatial orientation, and surface blends;

4. allow inheritance and/or merging of design attributes between parts or components (e.g., blending, corner radii, etc.);

5. manage and propagate constraints specified in the product needs analysis or in design goal specifications and enforce those constraints in the evolving solid models;

6. relate the design attributes and designer rationale to the components of the evolving product definition;

7. capture, store, and reason about an efficient repıesentation of the designer's intent;

8. transparently control the configuration of the design artifacts; and

9. support rapid browsing and modification of the above information about an evolving design.

Most conventional solid modeling or CAD systems are inadequate for mechanical design due to the complexity of design modifications, difficulty of manipulation, and lack of product information (e.g., material properties, surface finish, tolerance, surface condition, etc.). However, the above requirements have been used as guidelines in the development of the HPCAD Toolkit.

4.2 HPCAD Impact on Technology

Some of the most significant advances in HPCAD development, with respect to technology, have been in the GCSG area. A GCSG algorithm has been developed to support the incorporation of sculptured surfaces (e.g., Bézier curves and surfaces, B-spline curves and surfaces, and super quadrics) into a shape/feature-based HPCAD system. The GCSG algorithm provides the capability to combine definitions of assorted geometric solids which have been stored in different representations by performing Boolean operations on the solid objects. For instance, one may want to intersect an object represented as a polyhedral representation with an object represented as a B-spline representation. The existence of multiple solid representations can only be partly credited to the historical use of a variety of CAD packages in engineering/manufacturing enterprises. The variety of representations can also be credited to the greater efficiency or effectiveness of certain representations for specific purposes. Thus, the use of multiple solid representations is *inherent* to the engineering process. Consequently, an open toolbox of geometric routines is needed to translate geometry between applications and perform mixed representation Boolean operations on solids and surfaces. The emerging HPCAD Toolkit components will fill this need. Also inherent within the engineering process is the need for design evolution capabilities. Current solid modeling environments make this evolution difficult because of their limitations with respect to reversibility. The HPCAD Euler-based CSG provides a solid model modification and editing capability not currently available in systems other than the DKMS.

5.0 Container Object System (COS) Overview

The primary impetus behind the COS design and implementation is the overwhelming need for a design knowledge representation that can support the CE design process. This research is based on the observation that for any design knowledge representation scheme to be appropriate for CE applications, it must satisfy the following criteria:

1. be able to express CE types of design knowledge,
2. allow simultaneous modification of various design components, and
3. promote product design evolution.

Except for the DKMS COS, no existing representation satisfies all these requirements. The working hypothesis is that a knowledge representation scheme will adequately support product design evolution if its representational constructs are defeasible (i.e., they allow dynamic reorganization of descriptive knowledge).

5.1 COS Motivation

The CE paradigm places great demands on any candidate design knowledge representation scheme. In particular, three issues must be addressed. First, this scheme must be powerful enough to express the concepts within a design domain, such as geometric, temporal, and hierarchical information. Second, it must not force its implementation to inhibit the simultaneous utilization and modification of system and subsystem designs by different users. Furthermore, in any useful implementation, the effects of any updates must be computable and available to those designing the affected components. Third, perhaps the most subtle demand CE places on knowledge representation schemes is that they must allow product descriptions to evolve naturally over time. Without this capability, product design cannot be easily and efficiently modified and updated as the design process progresses.

Many proposed knowledge-representation schemes have suffered from an inability to support product-design-data evolution. The source of this shortcoming is that representations of descriptive knowledge are almost always based on rigid structures which are not amenable to continual reorganization. This rigidity prohibits the evolution of object descriptions, since the elements necessary to describe the final product usually are not known at the beginning of the design process. Unfortunately, nearly all current representation schemes suffer from this problem.

5.2 COS Impact on Technology

The core of DKMS consists of an object representation that facilitates the evolution of the definition of life-cycle artifacts and the flow of information between different life-cycle activities concerning the life-cycle artifacts. COS permeates nearly every DKMS component based on the common need to store, retrieve, and modify "container objects." The name "container object" derives from a basic assumption about how a modeled entity should be represented. The view of the container-object-based representation agrees with the traditional view of objects in that all descriptive information regarding a real-world entity should be considered state information which is local to the stored representation of that entity (10). Unlike the rigid descriptors used by traditional object systems, a *"container object" is one in which the stored representation simply contains all the* information describing its real-world entity; very little structure is assumed.

The rigidity of traditional object systems stems, in part, from the assumption that the existence of an object depends on its description. There is usually not a convenient or natural way for the designer to tell the system, "I know that we need an engine, but I haven't decided on the type of engine yet, so I do not know what characteristics it will have."

In the Container Object Paradigm, the separation of existential knowledge from descriptive knowledge makes this possible. The descriptive knowledge associated with a container object is represented much like the descriptive knowledge in other object representations; existential knowledge is captured by a nonqualitative characteristic property, unique to that object, called the *haecceity*.[1] By distinguishing between object haecceity and description, a container object representation allows the programmer to declare the existence of objects without describing them.

Philosophers have noted that existence of an object *per se* should be distinguished from the collection of descriptive properties it exemplifies. Linguistically, this translates into the previously discussed idea that the declaration of the existence of an object should be separate from its description. To capture this idea, the container object representation has a statement to declare existence, and others to add and delete descriptions (perspectives).

Whereas haecceity is the basis for representing existential knowledge about container objects, the basis for representing descriptive knowledge is a data organization structure called a *perspective*.[2] A perspective, *p*, may be visualized simply as a set of essential properties which an object having that perspective must possess. That is, a perspective is used to describe some real-world concept by enumerating all and only those properties which convey the essence of the real-world concept. A simple example might be described as follows.

A person can be described by many essential attributes, such as sex, race, mother's name, father's name, etc. Therefore, a model of an actual person could be described by a *person* perspective having some or all of those attributes. Then, in the model, if an object, *o*, has (or is given) the perspective *person*, *o* must possess those properties that define the *person* perspective. Therefore, if *person* contains a property *race*, then *o* should possess an attribute called *race* which should also be associated with some value.

The difference between a perspective and a regular class is that perspective description information can be dynamically added to or removed from a container object description.

[1] The term "haecceity" (pronounced "heck-say¢-it-ee") derives from medieval philosophy, especially the Schoolman Duns Scotus, and has been revived in recent developments in the philosophical issues surrounding modality and modal logic. See, e.g., (11).

[2] The term *perspective* comes from the KRL paper (12). However, the definition used here is substantially different than the one used in the KRL paper.

Thus, a container object is not necessarily given a particular data type (e.g., an item that starts as a toaster may evolve to an automobile). Therefore, the COS supports the subtle CE demand for supporting the natural evolution of product descriptions over time. This dynamic reorganization of descriptive knowledge gives the COS a major technical advantage over current representation systems.

6.0 Model Developm 1t Support Environment (MDSE) Overview

The MDSE component of DKMS is a method for the computer-assisted development of computer models. MDSE is a system for the creation, maintenance, and modification of engineering models and their numerically oriented software implementations. It combines several mature technologies, such as computer algebra, constraint management, dimensional analysis, object-oriented databases, automated-code generation, and automated-documentation generation.

6.1 MDSE Motivation

The management of design knowledge involves support for the creation and maintenance of engineering models that are often no more than FORTRAN subroutines. By the time a piece of engineering analysis software is in use, it is often obsolete. The state of the art in the product technology and associated modeling techniques changes so quickly that it is difficult to keep software up-to-date. Clearly, basing design decisions on models that have become outmoded is a mistake.

6.2 MDSE Impact on Technology

MDSE emphasizes the separation of knowledge acquisition and model development. Mathematical equations used for analysis in a field are broken into two separate components: the state variables representing physical properties and the equations representing relationships and correspondences (Figure 3). Reuse of engineering modeling knowledge and accuracy of the associated model implementation software is improved because of this separation. Accumulation of engineering modeling rationale and knowledge is simplified to make the system as unobtrusive as possible.

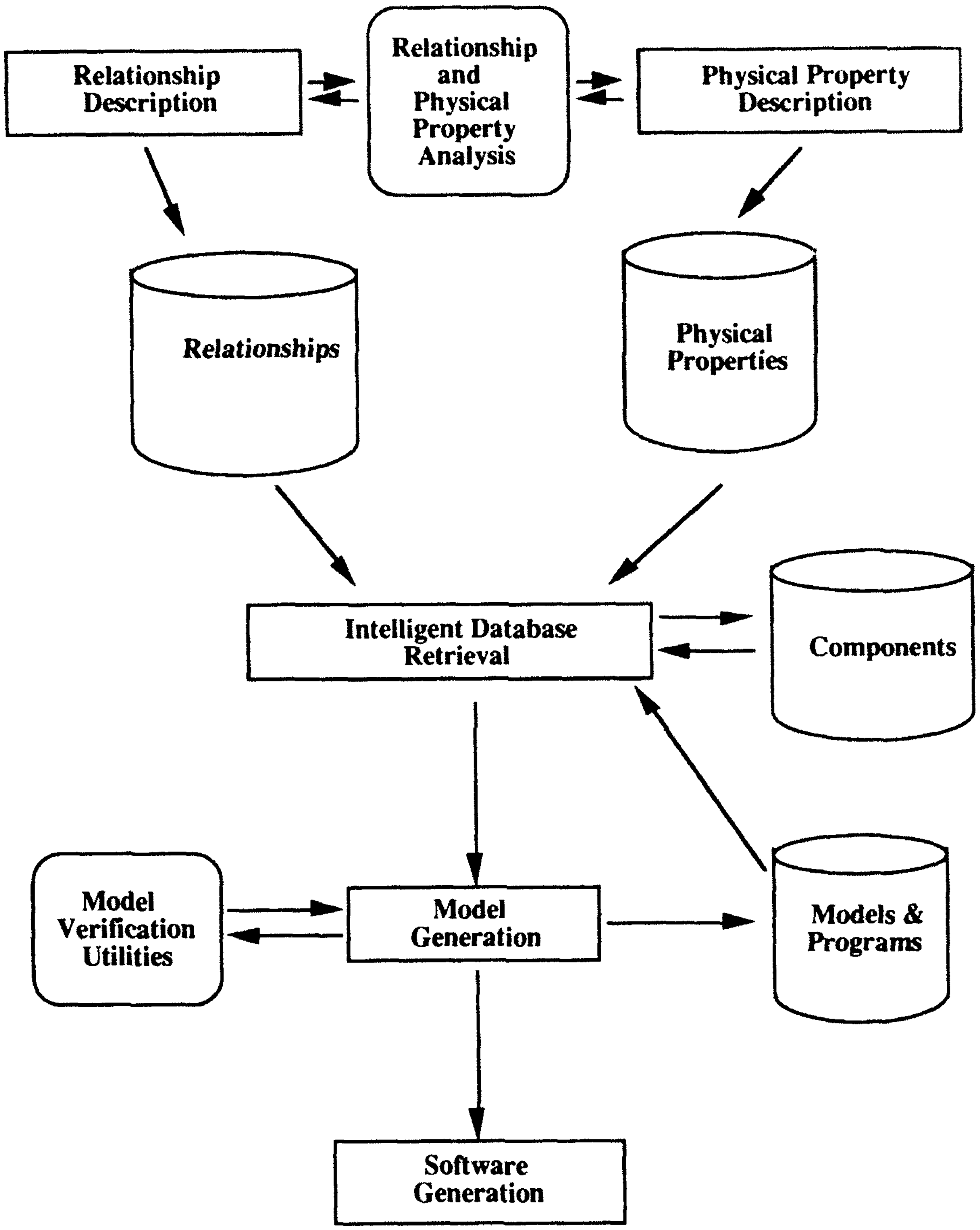
Relationship Description
Relationship and Physical Property Analysis
Physical Property Description
Relationships
Physical Properties
Intelligent Database Retrieval
Components
Model Verification Utilities
Model Generation
Models & Programs
Software Generation

Figure 3. MDSE Architecture

Both relationships and physical properties are arranged in a hierarchy, and the default values of each are inherited. This reduces the effort required to enter information. Further, locating a relationship or physical property is simplified because of the logical structuring enforced by the hierarchy. This structure allows the modeler easy access to similar relationships. If the modeler felt that the uncertainty inherent in this empirical relationship were too large, he or she could simply move up the hierarchy to a level with the correct generic relationship, then look at other, more specific relationships which have lower uncertainty.

A new model definition consists of: 1) defining a program, 2) selecting a set of relationships and physical properties for that program, 3) mapping the physical properties to the variables in the relationships, and 4) selecting the physical properties that will be known at the time the model will be run. With this definition, constraint management can be applied to find the potential sequences in which the equations can be solved. These sequences may vary in the number of sets of simultaneous equations and the size of those sets. These factors may make one solution sequence preferable to another; therefore, the sequences are presented to the user for perusal.

Traditional engineering analytical modeling support has focused on either domain-specific "customized" modelers (e.g., an automotive cooling system modeler) or generalized mathematical-equation solving packages. Domain-specific modelers suffer from traditionally high development and maintenance costs as well as a lack of reusable modeling knowledge and code. At the other end of the spectrum are the prepackaged, generalized, mathematical-solver software packages which, while less expensive to implement, provide little or no support for the actual engineering model development process. MDSE bridges the gap between the reusability and flexibility of the mathematical packages and the modeling power of domain-specific modelers.

7.0 Conclusions

The successful application of design knowledge management will provide a massive improvement over any currently available design automation concept. DKMS technology will provide support for design engineers to better integrate the trade-off of various design attributes such as performance, cost, schedule manufacturability, and supportability. An order of magnitude reduction in redesign and engineering change requests can be realized by supporting the knowledge-based delivery of manufacturing and maintenance experience to the initial product designers. Finally, DKMS will be useful in many areas other than product design because of the integration and HPCAD support it offers. For example, in manufacturing planning and production planning, access to the design rationale and design knowledge bases will allow automation of a number of these manufacturing engineering activities and significantly reduce quality and reliability problems in the final product. DKMS will also promote better university/industry/government ties in that it would provide a direct vehicle for moving design, manufacturing, and field experience into the classroom.

8.0 Acknowledgment

The work discussed in this paper was made possible at Knowledge Based Systems, Inc., by a Small Business Innovative Research Phase II research grant, Contract No. F33615-90-C-0011, United States Air Force Human Resources Laboratory, Wright-Patterson Air Force Base, under the technical direction of Captain JoAnn M. Sartor.

9.0 References

1. Linn, J. L., & Winner, R. I. (Eds.). (1986). The Department of Defense for engineering information systems: Volume 1 - Operational concepts; Volume 2 - Requirements. Alexandria, VA: The Institute for Defense Analyses.

2. Judson, D. L. (1985). Integrated information support system (IISS): An evolutionary approach to integration. WPAFB, OH: AFWAL/ML.

3. Integrated design support system (IDS): Volume I - Executive overview; Volume II - IDS introduction and summary; Volume III - IDS requirements; Volume IV - IDS task results; Volume V - IDS software documentation (AFHRL-TR-89-6). (1989). WPAFB, OH: AFHRL.

4. Engineering information systems: Volume I - Organization and concepts; Volume II - Specifications and guidelines. (1989). Minneapolis, MN: Honeywell Systems and Research Center.

5. Tool encapsulation specification. (1991). CAD Framework Initiative.

6. da Silva, R. E., Wood, K. L., Beaman, J. J. (1991). An algebraic approach to geometric query processing in CAD/CAM applications. ACM Siggraph, ACM Press. New York, NY.

7. Knowledge Based Systems, Incorporated. (1990). A design knowledge management system (DKMS): SBIR phase I final report (Contract No. F41622-89-C-1018). WPAFB, OH: AFHRL.

8. De Floriani, L. (1989). Feature extraction from boundary models of three dimensional objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(8), 785-798.

9. Poole, D. A. (1990). A methodology for using a default and abductive reasoning system. International Journal of Intelligent Systems, 5, 521-548.

10. Stefik, M., & Bobrow, D. G. (1986). Object-oriented programming: Themes and variations. AI Magazine, 6 (4), 182-204.

11. Plantinga, A. (1974). The Nature of Necessity. Oxford, UK: Oxford University Press.

12. Bobrow, D. G., & Winograd, T. (1977). An overview of KRL, a knowledge representation language. Cognitive Science, 1(1), 3-46.

13. Barwise, J., & Perry, J. (1983). Situations and attitudes. Cambridge, MA: MIT Press.

14. Forbus, K. D. (1984). Qualitative process theory (AI-TR-789). Cambridge, MA: MIT.

15. Hobbs, J. (1986). Introduction in formal theories of the commonsense world. Norwood, NJ: Ablex Publishing Corporation.

16. Keene, S.E. (1989). Object-oriented programming in common lisp: A programmer's guide to CLOS. Reading, MA: Addison-Wesley.

17. Kuipers, B. (1984). Common sense reasoning about causality: Deriving behavior from structure. Artificial Intelligence, 24, 169-203.

18 Painter, M. (1991). Information integration for concurrent engineering: Program foundation and philosophy [Presentation]. WPAFB, OH: AL/HR.

19. Serrano, D. (1988). Constraint management in conceptual design. Unpublished doctoral dissertation, MIT, Cambridge, MA.

20. Steele, G. L., Jr. (1990). Common lisp: The language (2nd ed.). Bedford, MA. Digital Press.

21. Systems engineering methodologies: Volume 1 - 7 (ICAM Project 1701). (1985). WPAFB, OH: AFWAL/MLTC.

☆U.S.Government Printing Office: 1993 — 750-061/60239