WL-TR-91-8048

AD-A254 013

2

Engineering Information System (EIS)

WRIGHT LABORATORY

R. Kant
Honeywell Systems and Research Center
3660 Technology Drive
Minneapolis, Minnesota 55418

31 January 1992

Final Report for Period September 1987–July 1991

92-20232

Manufacturing Technology Directorate
Wright Laboratory
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433-6543

92 7 27 148

Notice

This technical report has been reviewed and is approved for publication.

Daniel R. Lewallen — 5 March 1992

Daniel Lewallen
Project Manager

Date

[illegible] — 6 March 1992

Bruce A. Rasmussen, Chief
Integration Technology Division
Manufacturing Technology Directorate

Date

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1/31/92	3. REPORT TYPE AND DATES COVERED Final Report, 9/87 to 7/91

4. TITLE AND SUBTITLE Engineering Information System (EIS)	5. FUNDING NUMBERS PE-78011F, 63452F, C-F33615-87-C-1401, PR-2700, TA-02, WU-28
6. AUTHOR(S) Raj Kant, Jon Krueger	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Honeywell Systems and Research Center 3660 Technology Drive Minneapolis, Minnesota 55418	8. PERFORMING ORGANIZATION REPORT NUMBER C920049
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Daniel Lewallen (513-255-7371) Manufacturing Technology Directorate (WL/MTIA) Wright Laboratory Wright-Patterson Air Force Base, Ohio 45433-6533	10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-91-8048

11. SUPPLEMENTARY NOTES

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

13. ABSTRACT *(Maximum 200 words)*

The Engineering Information System (EIS) was created in response to increasing concerns about the exchange of engineering information between contractors and the government, contractors on a single team, and contractors on different teams who share technical data and diverse tools. The EIS specifications provide a means for managing heterogeneity among hardware and software platforms, data formats, tools, site-specific policies and methodologies, and interfaces to create multiple-domain, multiple-life-cycle environments.

An Engineering Information Model (EIM) and the EIS framework comprise the EIS. The EIM is a semantic model of information found in an EIS. An EIM realization, called the reference schema, is a set of object types that form a specific logical organization of data and operations. The EIS framework contains automated services embodied in software to support the use of the EIM to manage and control the data and activities of the engineering process. Current computer networking gateways require significant enhancement to meet the needs of cost-effective and concurrent engineering information access. The use of integration information gateways is the key to achieving the levels of technical interchange needed between organizations and various engineering environments. The EIS technologies can provide the integration necessary to allow transparent information access across the information exchange networks.

14. SUBJECT TERMS Framework, information model, information integration, information integration framework	15. NUMBER OF PAGES 73
	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 UL

NSN 7540-01-280-5500

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

Table of Contents

Table of Contents (Concluded)

List of Figures

List of Figures (Concluded)

List of Tables

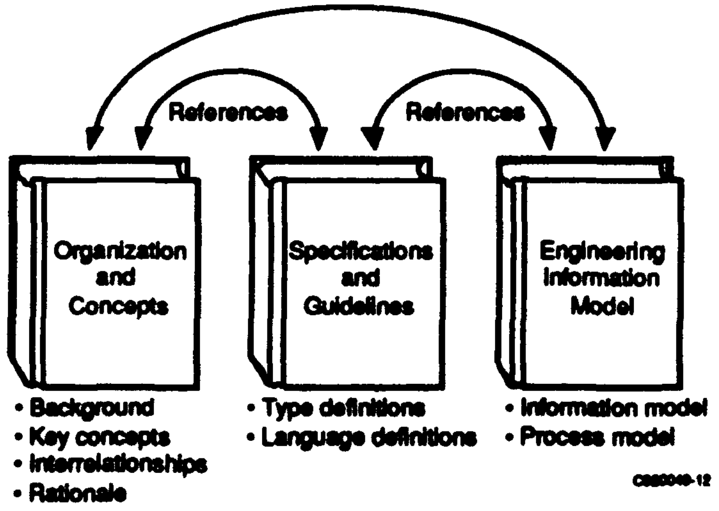
Executive Summary

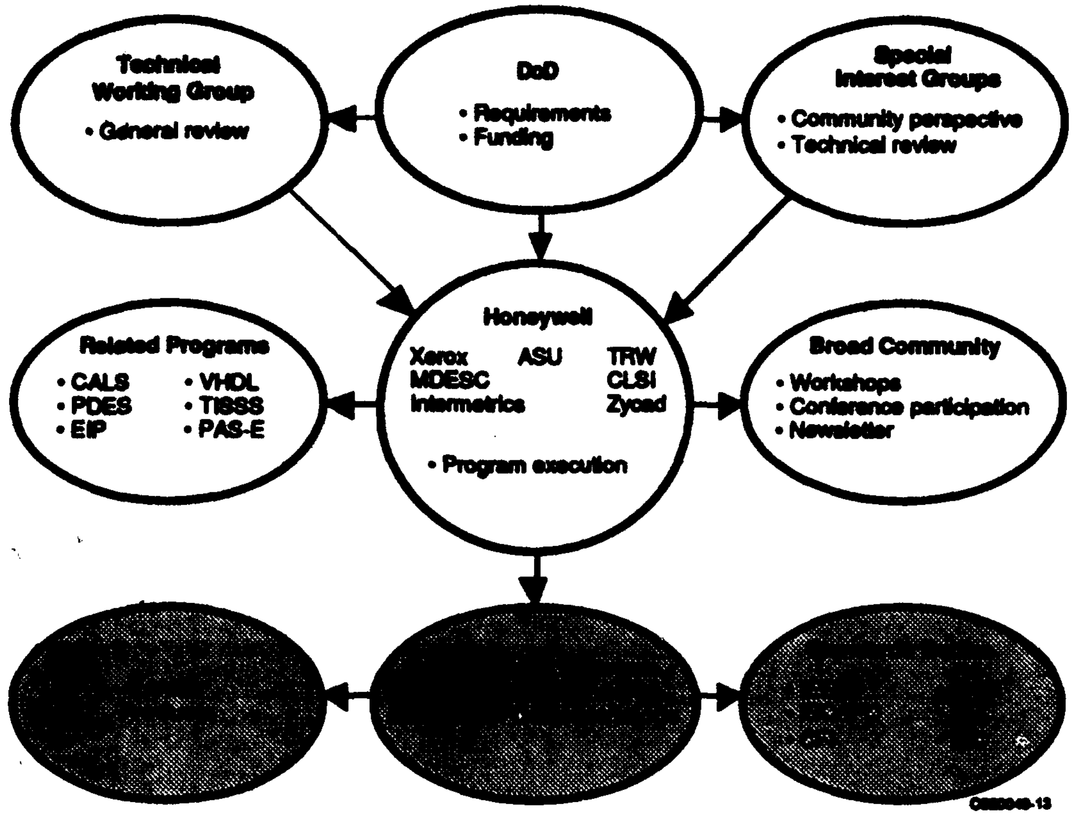
The engineering information system (EIS) program was created by the U.S. Department of Defense (DoD) in 1987 in response to increasing concerns about the exchange of engineering information between contractors and the government, contractors on a single team, contractors on different teams who share technical data, and diverse tools at a single site. The EIS specifications provide a means for managing heterogeneity among hardware and software platforms, data formats, tools, site-specific policies and methodologies, and interfaces to create multiple-domain, multiple-life-cycle environments.

EIS is a highly successful and visible program in the DoD and related industry communities. Program participants have maintained a high degree of community interaction to produce the specifications and to catalyze the industry to develop standards for engineering environments. The vendor community is expected to implement these emerging standards in their products to meet the needs of the DoD and the commercial user community. All sectors of the community (tool vendors, workstation developers, engineering users, administrators, management, and DoD agencies) stand to profit from highly tailorable engineering environments that will allow licensed sharing of tools and data.

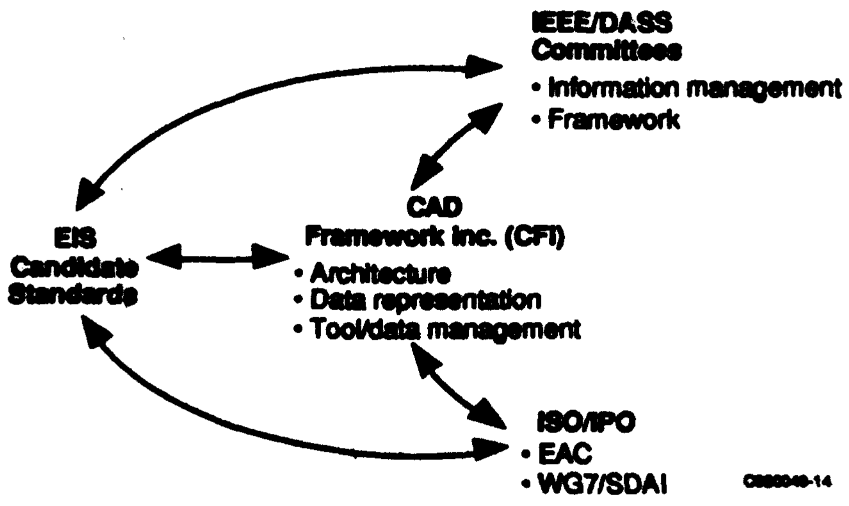
The three phases of the EIS effort are development of the EIS specifications, development and demonstration of a prototype that shows the feasibility of the specifications, and application demonstrations to illustrate the specifications at work in multiple-domain, multiple-life-cycle environments.

Between 1987 and 1989, the EIS specifications were developed and reviewed. Reviews by the community continue to be overwhelmingly positive. The specifications are documented in three volumes: *EIS Organizations and Concepts* (a narrative description of EIS functionality), *EIS Specifications and Guidelines* (template designs of the actual interface specifications), and *Engineering Information Model* (descriptions of both the administrative and electronic computer-aided design (ECAD) domain models).

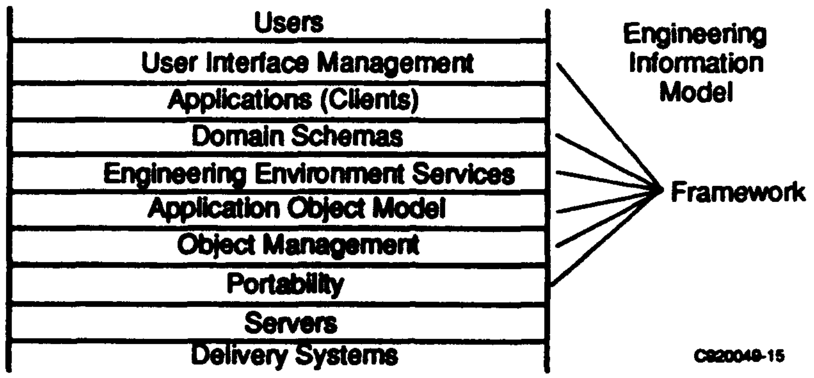




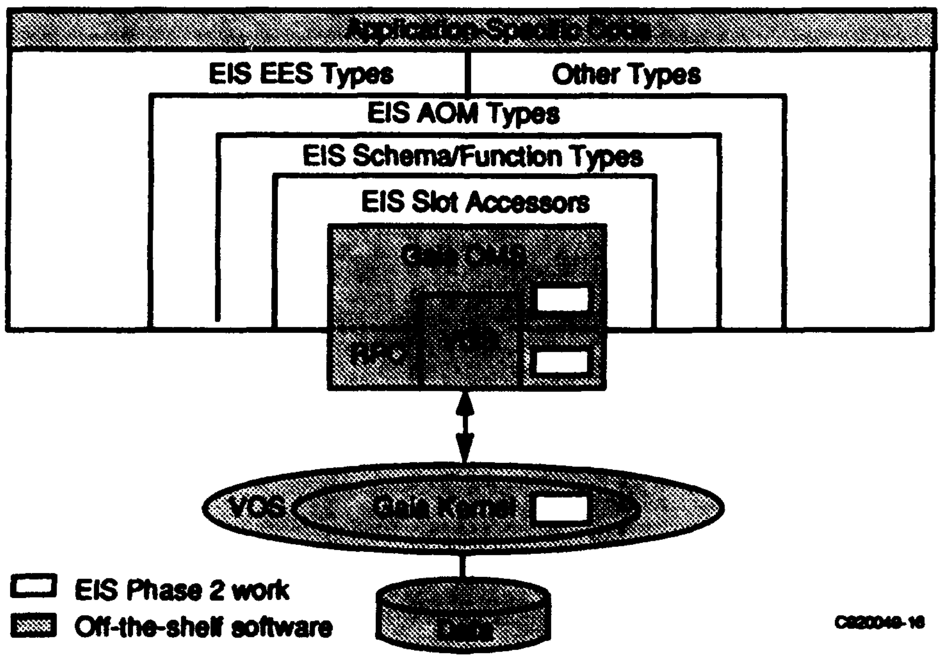
Progress on the EIS program has been presented at more than 35 conferences and workshops, 25 standards group meetings, 20 special interest group meetings, and 20 one-on-one meetings with users and vendors. The EIS specifications were developed under this significant public scrutiny, and parts of the EIS specifications have already been accepted by the involved standards organizations for incorporation into their evolving framework and information model standards.



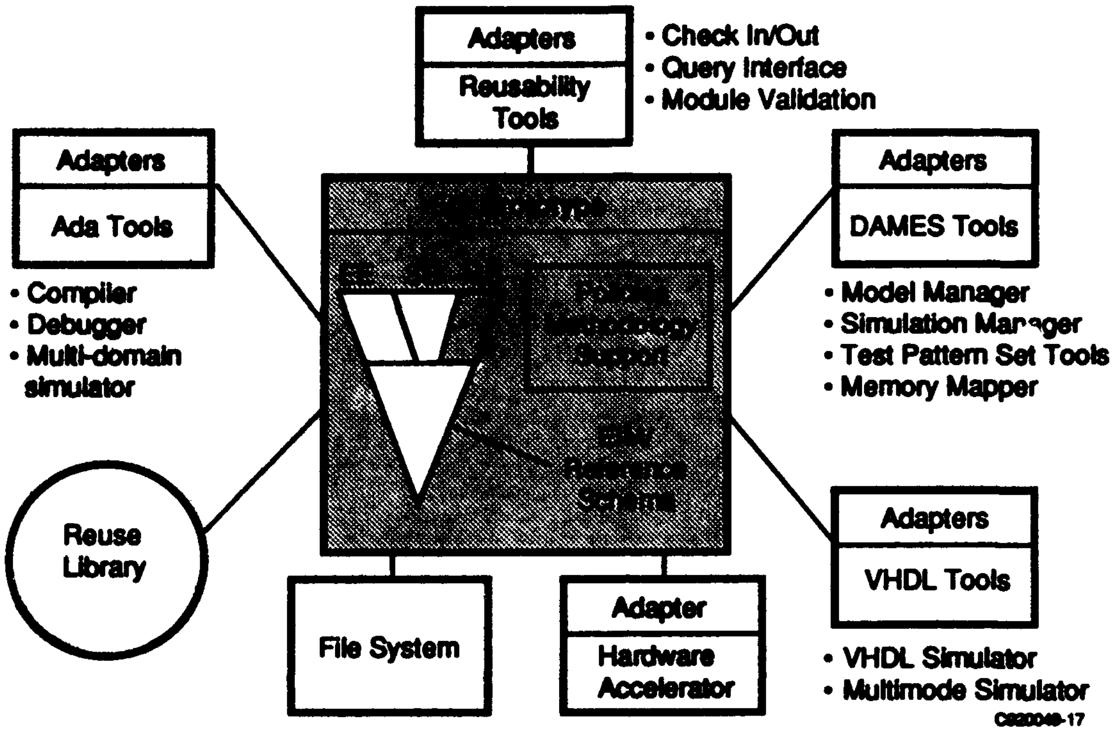
The engineering information model (EIM) and the EIS framework (underlying foundation) comprise the EIS. The EIM is a semantic model of information found in an EIS. The EIM has a realization called the reference schema; it is a set of object types that form a specific logical organization of data and operations. The EIS framework contains automated services embodied in software to support the use of the EIM to manage and control the data and activities of the engineering process.



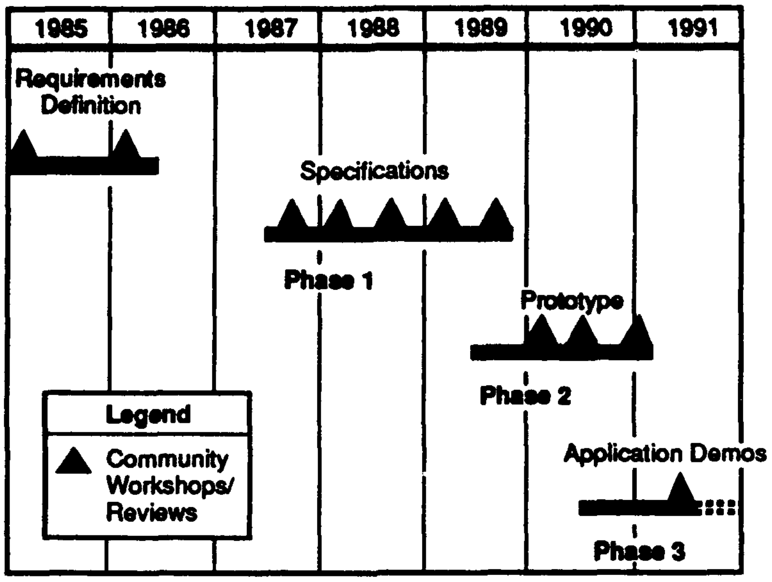
More than 250 object types are implemented in the EIS prototype (PREIS). The PREIS demonstrates both the viability of the EIS specifications and some ways to improve them. It successfully combines off-the-shelf, first-generation frameworks to meet EIS specifications—a critical issue that many vendors face as they try to adhere to an evolving industry-standard framework.



The EIS application demonstrations for electrical, software, and mechanical engineering environments will occur through 1995. One of these, the EIS application demonstration for the Advanced Tactical Fighter/Joint Integrated Avionics Working Group (ATF/JIAWG), environment will incorporate "reuse technology" and will demonstrate a "virtual system space" in which software modules can be integrated and tested with simulated hardware.

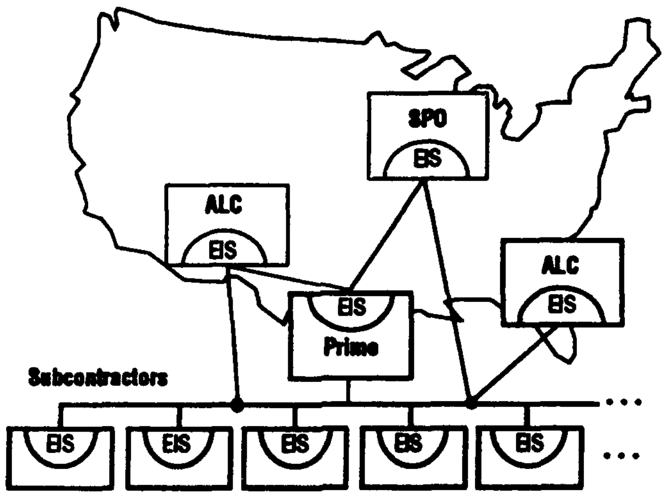


Since its inception, the EIS program has seen enthusiastic and intense community participation and reviews. The EIS requirements were developed with the help of numerous technical experts from 150 industry and academic organizations, resulting in the 1986 EIS requirements document. The EIS specifications were released in October 1989. PREIS, the EIS prototype, was completed in March 1991. Phase 3 of the EIS program will performed through 1995, with the performance of several EIS application demonstration tasks.



C920049-18

The EIS technologies (framework and information model) will likely impact the future operation of multiorganizational DoD contracting and subcontracting teams. Current computer networking gateways require significant enhancement to meet the needs of cost-effective and concurrent engineering information access. The use of integrating information gateways are the key to achieving the levels of technical interchange needed between organizations and various engineering environments. The EIS technologies can provide the integration necessary to allow transparent information access across the information exchange networks.



C920049-29

Significant success has been achieved in the EIS program in generating demand for standards. The results—the specifications, prototype, and demonstrations—will promote the evolution of conforming information models, frameworks, and engineering environments. Participants in the program and other vendors and users will implement portions of the EIS, continuing the development of the EIS concepts toward implementation and further refinement and usability of the concepts.

Section 1
An Overview of the EIS

The Engineering Information System (EIS) program was created by the DoD in 1987 in response to increasing concerns about the exchange of engineering information between contractors and the government, contractors on a single team, contractors on different teams who share technical data, and diverse tools at a single site. The creation of the EIS candidate standards provides a means to create heterogeneity among hardware and software platforms, data formats, tools, site-specific policies and methodologies, and interfaces.

The EIS is a highly successful program that is very visible in the DoD and in related industry communities. The EIS requirements were developed by the Institute for Defense Analysis following an extensive study. Numerous technical experts from 150 industry and academic organizations helped in the development of the 1986 EIS requirements document.

What Is an EIS?

An EIS is an environment that uses computers to manage massive amounts of engineering and administrative data. The DoD funded the EIS research program not only to make engineering products less expensive to develop, but (in some cases) to make their development possible.

EIS standards are very important to the realization of the next generation of computer-aided engineering (CAE) systems. The success of these standards relies heavily on community input and involvement; community understanding of the EIS program is vital. The statements and explanations that follow clarify the EIS program and its goals.

The EIS team did not actually build a marketable EIS. The EIS research program was designed to develop *candidate standards* for ensuring uniformity among EISs, a prototype to show the feasibility of building to the standards, and a demonstration of the prototype to illustrate that the standards actually work. The culmination of these goals exhibits the EIS market potential to tool and workstation vendors and is expected to result in the development of marketable EISs.

The EIS program is not a database program. It is designed to manage information and to address the incompatibilities of engineering design tools. The EIS relies on database management functions that are supplied by local database management systems (DBMSs), but is not restricted to any particular DBMS.

The object management system (OMS) of the EIS is not an object-oriented DBMS. A system object manager is required because an EIS is designed in an object-oriented paradigm; the OMS is the object manager of the EIS. Like a DBMS, the OMS handles requests for services by tools and by other parts of the EIS. Unlike a DBMS, the OMS uses one or more existing underlying file servers or DBMSs to store data, rather than a single integrated object storage server.

The EIS domain is not limited to ECAD. The initial focus of the EIS was on ECAD; however, the EIS is extensible to other engineering disciplines (e.g., software and mechanical engineering).

The EIS is not a computer-aided engineering system. It is, however, complementary with CAE systems. It is an information management environment in which CAE systems can execute. The EIS facilitates the exchange and use of data between CAE tools and CAE systems. Because an EIS is very general, it can be used across such life-cycle phases as design, implementation, and maintenance. Although the EIS is not a CAE system in the usual sense, the EIS standards define a family of information management systems. Some of the standards could be tailored to be computer-aided design systems.

The EIS program is not an attempt to standardize a set of computer-aided engineering tools. The program defines candidate standards for interoperability. It supports uniformity among working environments, but does not standardize particular CAE tools. The EIS framework is easily adaptable to a number of working environments by allowing on-line access to a variety of CAE tools. This uniformity and tool/system extensibility do not hinder competition among tool and workstation vendors. Instead, they promote a healthy competition and open new markets for many vendors.

Backup, restoration, and recovery are not part of the EIS kernel. An EIS has a kernel that is concerned with managing objects and applying operations. EIS engineering management functions are supported by the kernel. Backup, restoration, and recovery are an important part of any system; many argue that they should be part of the EIS kernel. The EIS provides a general mechanism for specifying and supporting individual backup, restoration, and recovery policies in its engineering environment services (EES). The EES is an important system component that covers a wide range of functions; it exists in every EIS. Backup, restoration, and recovery capabilities are provided within the EIS engineeing environment services instead of within the kernel so they can be tailored by sites to behave according to local policy.

The EIS team predicts that vendors will emerge to supply EIS products that are complete with policies. The notion of policy presumes that some EIS functionality is not standardized; it remains open to various implementations to provide system flexibility. It is sometimes wrongly assumed that significant engineering effort will be required to write policies at each EIS site. The EIS provides a consistent interface and a mechanism for applying policies. Some vendors are expected to recognize the opportunities in providing policies, while others may specialize in integrating a variety of EIS products from interoperable parts of another vendor.

Language standards are not the only way to achieve tool interoperability. The definition of language standards is just one step toward achieving tool interoperability; it is not the entire solution. The only sound way to achieve complete tool interoperability is to develop a common information model. This issue is addressed in the EIS program through the definition of the EIM as a candidate standard.

The EIM is perhaps the most important element of the EIS program in terms of user community acceptance. The EIS is described in English, pictorially represented with IDEF1X-like figures, supplemented with template descriptions, and coded in the machine-readable object-oriented data language (OODL). The EIS is both stored and can be queried in the EIS prototype.

The real value of the EIM is as an unambiguous definition of engineering objects. For example, when tool A produces a netlist that conforms with the EIM, tool B is able to consume that netlist. This is the cornerstone for data exchange, transfer, and interoperability. The flexibility of the EIM fundamentally relies on the unambiguous reference standard (or EIM documentation) followed by humans during the construction of tools A and B.

Object-oriented programming is a discipline that has evolved since the 1960s. With an increasing number of programmers discovering the advantages of object-oriented programming, it is often thought of as a new programming technology. Object-oriented programming has foundations in software engineering concepts such as data abstraction, inheritance, encapsulation, and message passing. It also has foundations in DBMS data models like hierarchical, distributed, relational, entity-relationship, and artificial intelligence (e.g., LISP, Smalltalk, Semantic Nets, and Frames). Extensive academic research was conducted before the object-oriented environment was selected as an EIS requirement. Industry review proved that it is the best approach for dealing with the structure and behavior of the complex information involved with engineering. Although the concepts behind object-oriented systems are not new, object-oriented programming is an important development in programming technology that is a benefit to an increasing number of applications.

The user interface management system (UIMS), when taken alone, is not an EIS candidate standard. The UIMS is an EIS component that produces candidate standards, such as guidelines which show how to construct a user interface that is consistent with other system interfaces. The UIMS also produces interaction objects that include the basic functions to convey information to and from the user, and a UIMS script language for creating user interfaces. The guidelines, the interaction objects, and the UIMS script language are candidate standards—not the prototype UIMS produced for the demonstration.

Attachment of tools to the EIS will be made easier. The EIS candidate standards specify the interface for tools to the EIS. The EIS architecture supports a variety of tool attachment methods ranging from loose attachment to tight integration and numerous partial integration possibilities. The tool attachment candidate standard is intended to lower the cost of integration for existing tools and new tools built to comply with the standard.

The EIS does not force users to share their proprietary tools and data. The flexibility behind the EIS candidate standards would be destroyed with forced sharing of tools and data. While the EIS is a framework for attaching site-specific tools and databases, the EIS does not in any way compromise the confidentiality of tools and data. Communication with an EIS is strictly controlled, even from other EIS sites. In fact, the EIS provides a very flexible way to control access based on the object-oriented paradigm.

The successful completion of EIS will benefit many. A significant portion of the cost of procuring large systems is associated with the management, control, access, and exchange of engineering information. The EIS candidate standards reduce that cost. Many EIS vendors are expected to emerge to supply the needs of contractors and subcontractors for EIS products after the candidate standards have gained user community acceptance. All sectors of the community, including tool vendors, workstation developers, engineering users, administrators, management, and the DoD, stand to profit from a highly tailorable EIS that allows licensed sharing of tools and data.

A Computer-Aided Engineering Environment

Before discussing the scope of the EIS specifications, it is important to establish the problem/solution domain. The EIS is part of a computer-aided engineering (CAE) environment. A *populated and installed* EIS is a CAE, including a complement of tools and tailored site-specific features (Figure 1).

A given EIS site may use proprietary and/or commercial tool sets. Site tailoring includes the specific design policies that the EIS is to enforce for the organization.

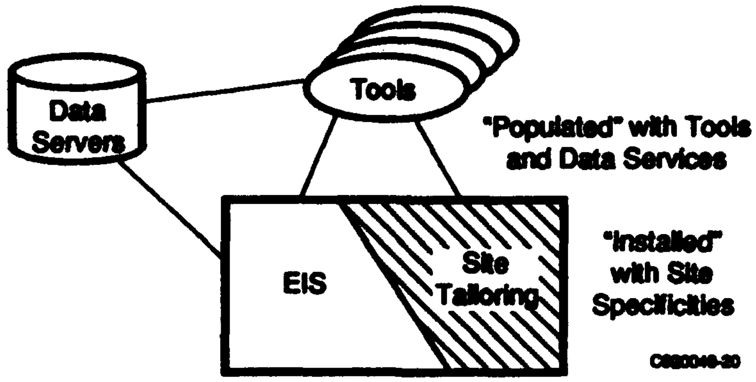

Tools
Data Servers
"Populated" with Tools and Data Services
EIS
Site Tailoring
"Installed" with Site Specificities
C880049-20

Figure 1. Populated EIS Environment

From the EIS perspective, database management and file systems (collectively called data servers) are a special kind of tool with the function to administer persistent data. There is an additional facet to data servers; many contemporary tools have achieved separation of data from computation using database management systems. The EIS is intended to support attachment of these data and file servers while maintaining the role of intermediary between tools and data.

A natural conclusion of this description is that it is important that an EIS can be used to assemble and integrate an unforeseen set of tools of largely unknown origin and behavior. It is also important that EIS mechanisms for policy support be available and useful for creating powerful tailored control and management functions.

Model and Framework

With further elaboration of the EIS portion of Figure 1, the EIS can be conceived as two basic parts—the EIM and the EIS framework.

- The EIM is a semantic model of information found in an EIS. It has a candidate standard realization, called the reference schema. The reference schema is a set of object types that form a specific, logical organization of data and operations. The

EIM and reference schema may be extended with site-specific semantics models and types, respectively, to capture the meaning and structure of data particular to that site.

- The EIS framework contains the automated services, embodied in software, to support the use of the EIM to manage and control data and activities of the engineering process.

The EIM is a conceptual model of administrative and electronic design information. It records common concepts and makes their meaning explicit for the primary purpose of aiding effective communication. The EIM is important for everyone involved with an EIS. A CAD tool builder uses the EIM to find common names and definitions for the information to be processed. A framework vendor uses the EIM to identify common operations and to consider whether they should be built into the vendor framework. An administrator of a CAD environment uses the EIM to quickly understand the scope and organization of EIS information to plan adaptation of the current system in preparation for implementing EIS concepts.

The EIM consists of two major components: the administrative domain model (ADM) and the electronic computer-aided design domain model (ECAD model) (Figure 2). The ADM is generic, modeling management appropriate, and control concepts appropriate for all design disciplines (e.g., interhost communication, access control, configuration management, and audit trails).

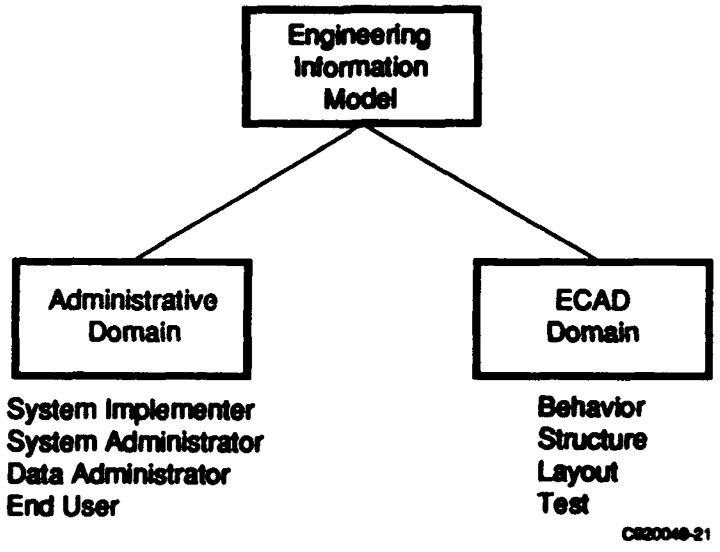


Figure 2. Engineering Information Model Domains

The ECAD model focuses on digital integrated circuit (IC) design information. Most information with which a digital IC designer works directly is found within the ECAD model. Analog and board information are outside the scope of this ECAD model but can be integrated into the model in the future.

The approach to developing the EIM is based on conventional information modeling techniques, but with some critical improvisations and departures. The IDEF1X methodology developed by DACOM (D. Appleton Company) is the strongest influence. In the case of the ADM, the topics and scope were largely drawn from the EIS requirements. While much of the "front-end" work was already done, the task of interpreting requirements and organizing topics into cohesive sections for review remained.

A traditional information flow analysis was performed for the ECAD model to resolve scoping issues and to improve model integration. The activity can be characterized as "top-down" modeling, with intensive interaction with IC designers. The Honeywell Solid State Electronics Center was used as a representative IC design environment. This activity produced a detailed process model expressing the data and operations associated with each major IC design process step. The ECAD model is based on this process model.

Entities, relationships, and definitions of the EIM form the common understanding needed to communicate the information behind the data, analogous to the organization of and meaning behind English words. Just as English has semantic, syntactic, and lexical levels of definition, so EIS has an EIM (semantics and "syntax") and a reference schema (lexicon).

Object type definitions of the EIS reference schema establish the "words" (function interfaces) to EIS engineering and administrative "phrases" (data), with an "alphabet" (schema language) consisting of application object model (AOM) attributes, relationships, and operations implemented by functions. These object type definitions appear in Sections 2 through 5 of the *EIS Specifications and Guidelines* (OMS, engineering environment services, User Interface, and Integrated Circuit Design).

EIS Framework

Figure 3 expands the EIS framework, retaining the EIM/reference schema dimension.

- The OMS is responsible for providing access to objects and operations. In a sense, it is the common "bus" to which everything connects, including tools and data servers. Data servers are files, file systems, databases, database management systems, computer RAM, and anything else that stores data. By being on the same bus, tools (and their users) more readily gain controlled access to data in the various servers.

- The AOM is a layer built on top of the OMS to simplify the development of applications. It bundles OMS abstractions in a way that resembles several popular object systems (e.g., Smalltalk-80, CLOS, and C++).

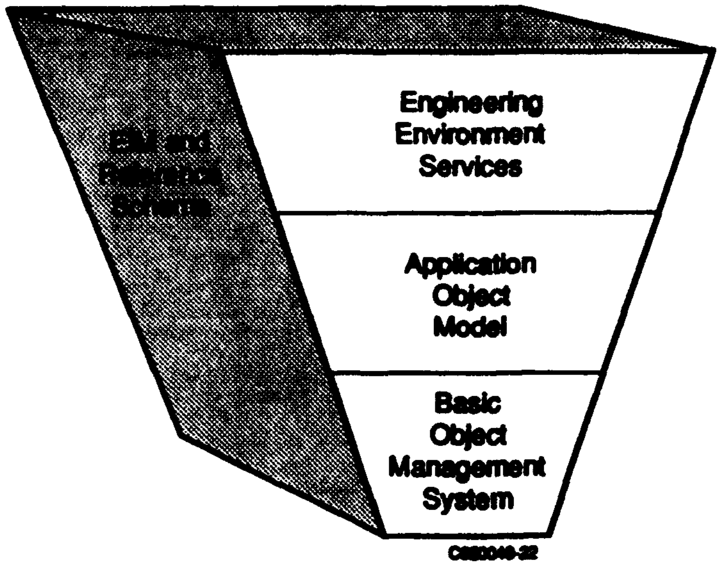

EIM and Reference Schema
Engineering Environment Services
Application Object Model
Basic Object Management System
C880040-22

Figure 3. EIS Model and Framework

- The EES encompass a broad range of services of interest to application developers or end users. Topics include configuration management, inter-EIS information exchange, policy management, audit trails, etc. Policies are used to automatically determine when system activity is needed and initiate it (system administrators can inject their own policies to tune system behavior to their specific needs).

Object Management System

The OMS performs type checking; maps operation requests to an appropriate implementation based on availability of data and hardware devices; maps parameters specified in the context of the client to corresponding parameters in the context of the server; invokes the selected procedure in a distributed, heterogeneous run-time environment; and controls its execution. Primary components of the OMS are:

- A meta-model and language;
- Schema management facilities;
- The function applier and execution control facilities;
- Support for object identification; and
- Data access facilities.

The role each OMS component served in the EIS is described briefly below.

The meta-model provides abstractions to hide details of request specifications handled by the OMS. For example, the *function* abstraction allows requests to be specified in terms of the required service rather than in terms of the piece of code that supplies it or the hardware on which it will execute. Use of the abstraction hides many details that are sources of incompatibilities among components (Figure 4).

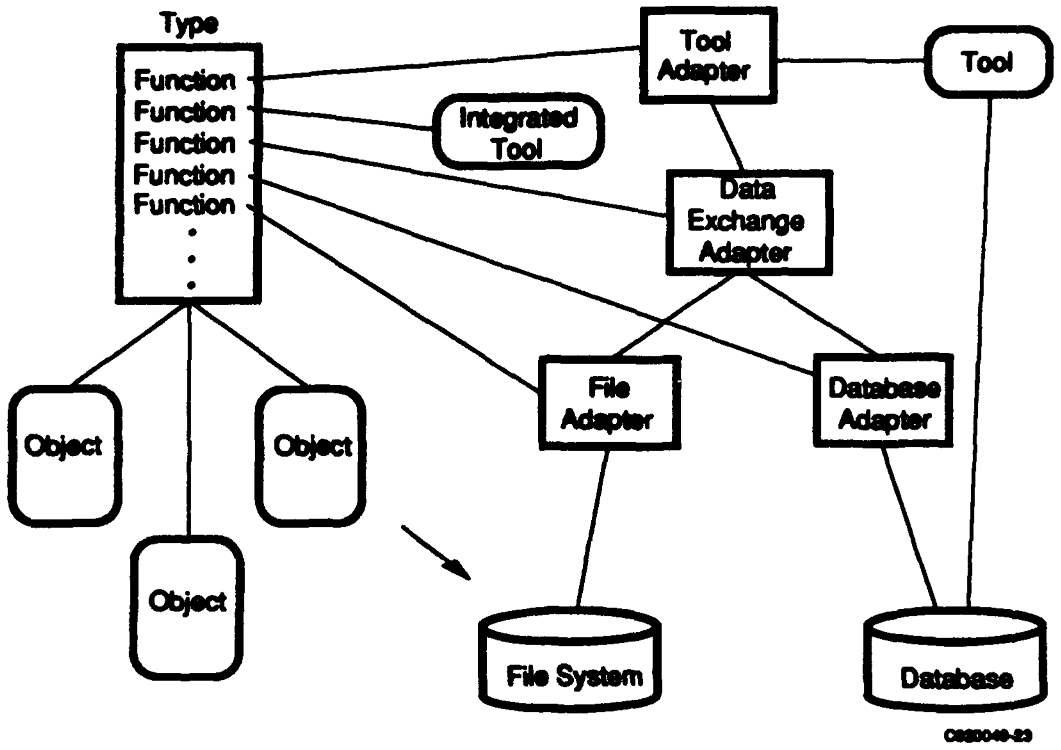


Figure 4. Use of Object/Function/Type Concepts to Hide Implementation Details

Abstractions aid in formulating detailed specifications of properties and relationships among programs and data objects that comprise EIS to allow generic mappings supplied by the OMS to be tailored to the actual components. The main abstractions provided are *object*, which describes component identity and persistence; *function*, which describes the semantics of procedures and their external interfaces; and *type*, which describes the structure and behavior of components.

Schema management facilities provide internal support for specifying and maintaining *descriptions* of components. They provide storage and manipulation for the specifications of assumptions and guarantees of clients and servers in support of function application and various mappings supplied by the OMS.

Function application resolves a client's procedure references to specific servers connected to the OMS. This resolution is based on the function identification and types of arguments specified in the request and the current configuration of hardware, software, and data servers.

Given a specific procedure to implement the specified function, function application involves type-, programming-language-, and hardware-specific mappings that are applied to specified arguments to provide appropriate parameters to the selected server procedure. Programming-language- and hardware-specific mappings are provided by various execution engineers that also provide appropriate protocols for procedures invocation.

Finally, function application invokes the selected procedures, passing parameters to them. It performs similar mappings of results returned from the server before passing results back to the client.

The OODL interpreter allows multiple requests for function application to be expressed and optimized as a unit before individual components of the compound request are handled by the function applier. Execution control facilities ensure that exceptions raised by a server are handled either by a client or by OMS-provided default handlers. They also support synchronization of client and server tasks and provide constructs to describe transaction semantics.

Object identification services support type-specific identifiers and naming conventions that permit creation of object identifiers that are meaningful in multiple contexts. The identifiers have embedded object semantics so they can be interpreted uniformly from the vantage points of the OMS and various clients and servers.

Data abstraction services provide mappings between identifiers of abstract objects and their representation. This allows client procedures to reference abstract objects and servers to manipulate the corresponding representation objects in a uniform controlled way in which the system executes actual mappings between objects and contexts.

Data access facilities provide an object-oriented query language and transformations from this language to the query language(s) of the heterogeneous underlying servers. These facilities also accommodate incompatibilities among heterogeneous servers by taking up the slack for weak servers in the areas of search and result structuring.

Application Object Model

The AOM builds a higher level layer of constructs on top of the OMS (Figure 5). New applications can use the constructs (similar to those found in several popular object-oriented programming languages and database systems) to define new types, create instances of them, and install and invoke operations on those types.

A type in the AOM consists of a set of attributes and relationships, collectively called *instance variables*. An instance variable represents a pair of data access operations, one for retrieving and one for setting. Names and parameters of the access operations are derived from the instance variable name and type. An attribute conceptually holds one or more instances of a *basic type*; e.g., a value that is independent of the state of the rest of the system. A relationship contains a (collection of) reference(s) to another object(s).

Operations represent one or more functions. When a tool requests the application of an AOM operation, types of the tool-supplies parameters are used to resolve the request to a function in much the same way that a function is resolved to an implementation. The difference is that the AOM defines a specific search pattern to be used that follows the type hierarchy to find the applicable function.

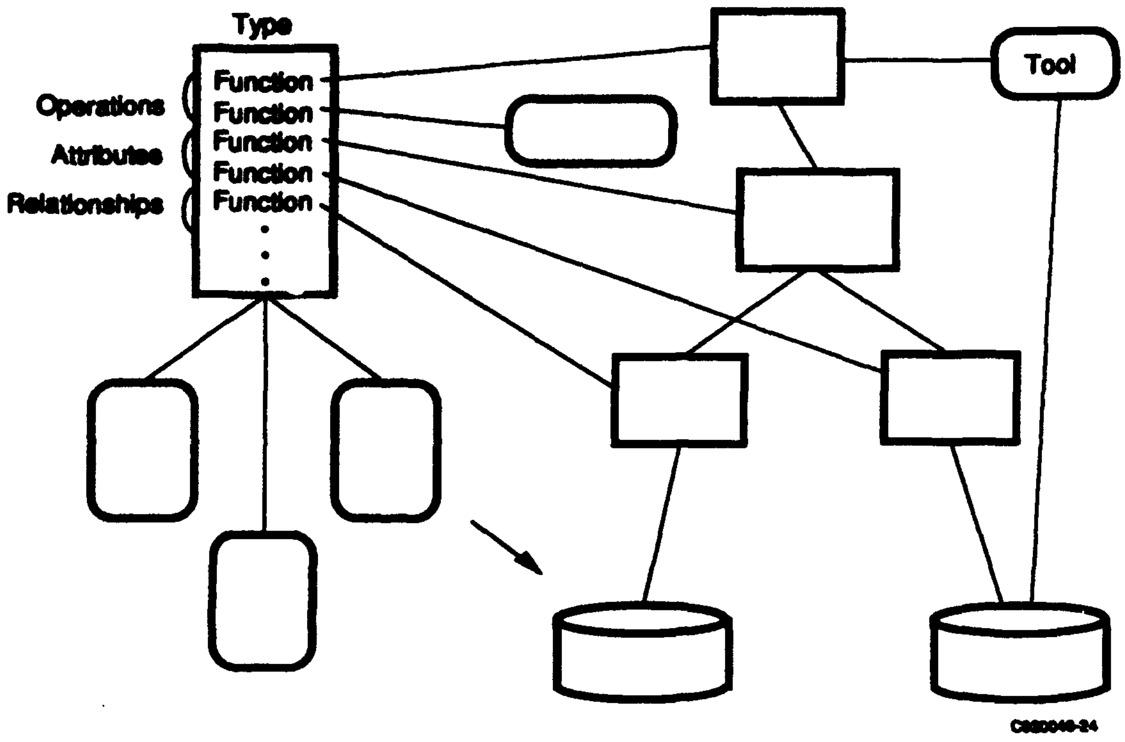


Figure 5. AOM Concepts Build on OMS Meta-Model Concepts

An application creates instances of AOM-based types in exactly the same way as defined by the OMS—applying the create function (operation) to the appropriate type object.

Engineering Environment Services (EES)

EES refers to a large collection of types, operations, and default policies that support the engineering and administrative processes of a design organization. Topic areas include:

- ***Control Points and Policies***—These are crucial to tailoring an EIS. Policies conditionally modify system behavior based on context. Control points act to trigger specific policies in response to selected transitions in the system (e.g., function implementation termination).

- ***Engineering Management and Control***—

  - Access Control—EIS access control is based on policy. Default policies exist to govern access to objects based on a three-dimensional matrix (user role, operation, object). The Boolean elements of the matrix indicate whether a particular operation may be applied to a given object based on the user's effective role at the time of the request.

- Audit Trail—Using these services, an EIS can be configured to automatically capture and summarize information about requested operations.

- Configuration Management—This topic area includes the subtopics of change control, dependency tracking, and version control.

- ***Information Management Services***—

  - Backup and Archive—In EIS, archive differs from backup: backup *copies* the object to stable storage; archive *moves* the object. Backed up data is still active; archived data is inactive and must respond differently to access attempts. Operations for inverse of backup and archive are also provided.

  - Repository Support—Repositories are used to organize objects around common themes, much like directories in a file system. The repository model and services specified in the EES have many features in common with hypertext.

- ***Inter-EIS Information Exchange***—These services can be separated into three layers that correspond to the OSI reference model (session, presentation, and application layers). The session layer takes care of establishing a point-to-point channel to convey a high-level protocol. The presentation layer handles formatting of EIS data for transmission and reception. The application layer, the most common point of contact for clients, uses a (tailorable) protocol to exchange complete EIS objects. Special hooks are provided to move objects between two systems with different schemas.

- ***User Environment Services***—While the UIMS provides the machinery to change a video display and convert user actions to framework actions, user environment services help formulate a high-level context for this activity. Login, work space, system alerts, notification, and system status monitoring are among subtopics in this area.

- ***Rule Processing***—Rule processing provides support for logic programming within the context of an EIS. The EIS Rule Specification Language (RSL) is based on *safe datalog*, in which programs are built from atomic formulas having *limited variables*. RSL is a declarative tool for writing policies, constraints, powerful queries, and (as is becoming increasingly popular with Prolog) CAD tools and CAD tool modules. There is also a flexible architecture for rule processing in future EISs.

Nearly all EES are policy-based; they provide basic services for use by site-specific policies. So ıe default policies are also provided, but they can be overridden or tuned to the specific needs of an installation.

User Interface Management System (UIMS)

The UIMS is a family of guidelines and candidate standards for an interface to a CAE system. It is a set of design guidelines and computer programs that guide and support development, execution, and maintenance of user interfaces. Design of the EIS user interface management system is independent of tools and design data in any given EIS installation. It is a multiwindow interface that supports the major dialog styles of command line, menu selection, form filling, question and answer, and direct manipulation. The UIMS design concentrates on interfaces for interactive EIS applications and tool adapters while providing interfaces for tying in external tools.

Applications developed using the EIS user interface management system consist of three component types: a set of *noninteractive core (NIC) functions* that take care of data access, transformation, and update; a set of *interaction objects* to manage display and keyboard; and a *script* that orchestrates the dialog (Figure 6). All of these are regular objects managed by the OMS.

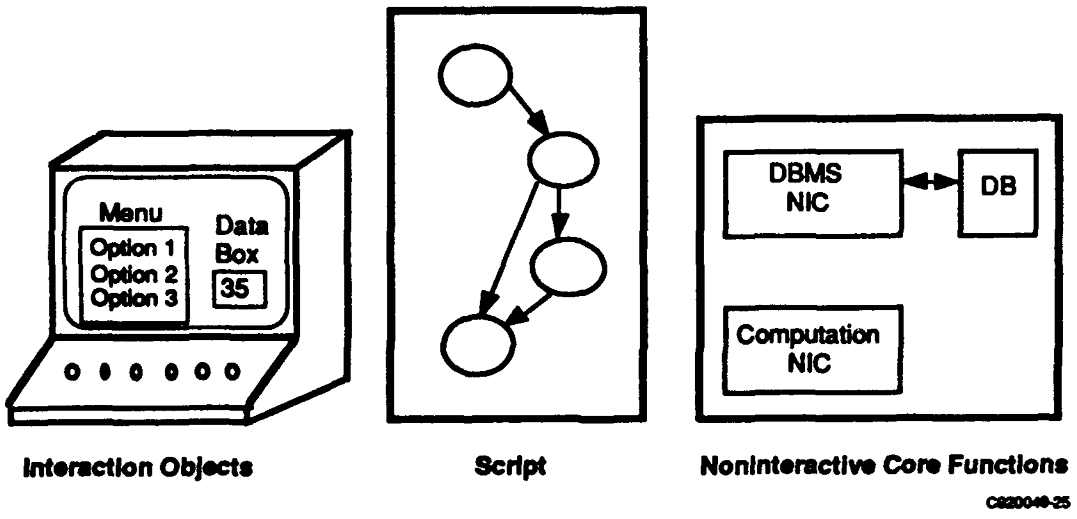


Figure 6. Interactive Application Components

The EIS user interface management system comes with an extensive set of guidelines and conventions for developing consistent, useful, and intuitive user interfaces. The guidelines and conventions address the major dialog styles mentioned above and are based on the extensive *Guidelines for Designing User Interface Software* by Sidney L. Smith and Jane M. Mosier. Their guidelines cover data entry, data display, sequence control, user guidance, data transmission, and data protection.

The *guidelines* are general recommendations to be used with judgment and adapted to the specifics of the applications. Alone they are too general to provide sufficient consistency—a common "look and feel"—across a set of user interfaces; however, conformance to them will improve the quality of user interfaces. User interface *conventions* offer specific interpretations to some of the guidelines for use across an entire system to improve consistency.

Composite Architecture Overview

Architecturally, the kernel of the EIS is the OMS/AOM. It is the hub for resolving references, causing invocation, and mapping requests and responses. Other parts of the EIS are extensions to the primitive services of the OMS and AOM; they could be implemented as kernel extensions or could appear to be tools running on the OMS. This organization is summarized in Figure 7; the figure is not a software block diagram, but a reference architecture. The components are logical, not necessarily physical software parts. This reference architecture could be built in many ways according to the following constraints and guidelines:

- An EIS must export the AOM and visible interfaces of the OMS. Implementations of the OMS/AOM follow the execution and data model semantics provided.
- The line between the OMS and AOM is for presentation purposes only; implementations could differ between these sets of services, could intermingle them, or could replace the low-level OMS with other primitives while preserving the external interface.
- A given EIS may include tailored versions of some or all of the EES; they can be implemented as kernel extensions or as tools.
- The reference schema is an EIS standard for the OMS/AOM, a candidate standard for other separable topic areas (i.e., configuration and version management), and may be tailored by either the EIS builder or a site administrator.

EIS Development Process

Participants in the EIS program have maintained a high degree of community interaction to produce a set of candidate standards. Many vendors are expected to implement the EIS candidate standards in their products to meet the needs of commercial and DoD users. All sectors of the community (tool vendors, workstation developers, engineering users, administrators, management, and DoD agencies) stand to profit from highly tailorable EIS implementations that will allow licensed sharing of tools and data.

The EIS program evolved in three phases, beginning with the development of candidate standards. The EIM and the EIS framework (integrating infrastructure) comprise the EIS. The EIM is a semantic model of information found in an EIS. The EIM has a candidate standard realization called the reference schema; it is a set of object types that form a specific logical organization of data and operations. The EIS framework contains automated services that are embodied in software to support the use of the EIM to manage and control the data and activities of the engineering process.

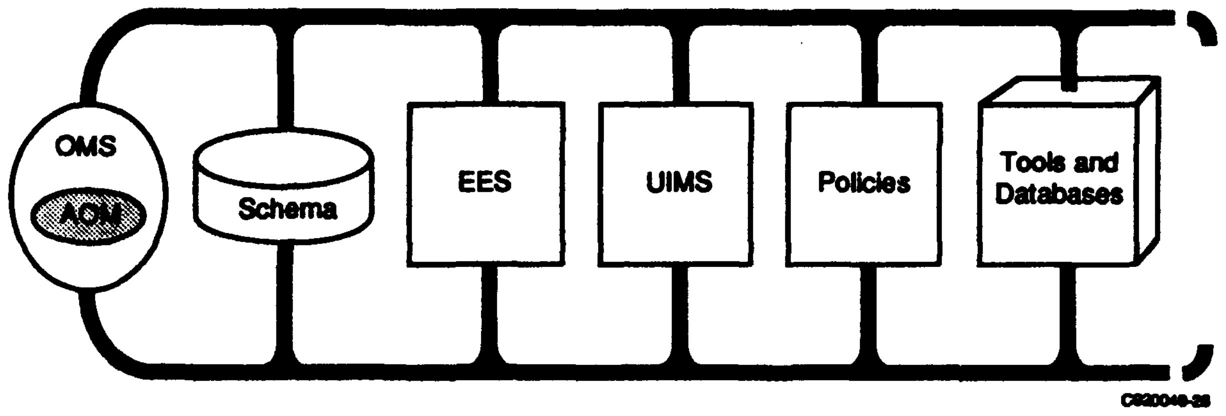


Figure 7. EIS Top-Level Reference Schema

The second phase was development of a prototype that shows the feasibility of the candidate standards. The EIS prototype implements more than 250 object types and demonstrates both the viability of the EIS specifications and some ways to improve them. The EIS prototype successfully combines off-the-shelf, first-generation frameworks into a standard EIS framework—a critical issue that many vendors face as they try to adhere to an evolving industry-standard framework.

Phase 3 of the program is the prototype demonstration to depict the candidate standards at work. EIS application demonstrations for electrical, mechanical, and software engineering environments are planned. One demonstration, the EIS application demonstration for ATE JIAWG environments, will demonstrate a virtual system space in which software modules can be integrated and tested with simulated hardware and in which "reuse technology" is incorporated.

Special Interest Groups

Special interest groups (SIGs) were formed as a cooperative effort between members of the EIS team and the user community to allow the EIS team to receive valuable user community input and affording them an opportunity to share in the creation of the EIS candidate standards. Meetings were held in various places throughout the United States, usually in conjunction with major software conferences, to simplify attendance for SIG members.

Technical Working Group

The EIS technical working group (TWG) was an industry and academic group convened by the Institute for Defense Analyses (IDA) to provide independent technical analysis and advice to the EIS program office.

The Three-Volume EIS Specification Set

Audience identification is a challenge that accompanied the formulation of the EIS specifications. Although there would be overlap in content (without time and resource constraints), it would be easy to have a separate set of EIS documents for end users, tool providers, system integrators, EIS suppliers, and standards bodies. The EIS specifications document set is organized by logical components rather than by user group.

The EIS specifications comprise three volumes: *Organizations and Concepts*, *Specifications and Guidelines*, and *Engineering Information Model*. The organization and concepts volume describes the various EIS components, how they fit together, and individual and collective behaviors. The specifications and guidelines volume contains object type definitions that define inheritance lineage, attributes, relationships, and operations for all of the EIS types. It also defines the EIS languages, portability service interfaces, and user interface guidelines. The EIM volume presents the administrative and the integrated circuit ECAD models.

The Standards Process

Since 1987, the progress of the EIS program has been presented at more than 35 conferences and workshops, 25 standards group meetings, and 20 one-to-one meetings with users and vendors. The EIS specifications were developed under this significant public scrutiny, and portions of the specifications have already been accepted by the involved standards organizations for incorporation into their evolving framework and information model standards. The EIS specifications, prototype, and demonstrations promote the development, validation, and harmonization of information models and the multidomain, multi-life-cycle environments. Note the following examples:

- The EIS program was instrumental in the reactivation of the IEEE/DASS/WGIM group in 1989.
- The EIS ECAD model made significant progress in the VHSIC Hardware Description Language/electronic data information file (VHDL/EDIF) harmonization technology.
- The EIS program/Honeywell were significant founding members of CAD Framework Initiative (CFI). The CFI has accepted the engineering information system ECAS model as one of their candidate domain models.
- PDES/EAC accepted the ECAD model as one of their candidate domain models.

The EIS program has also influenced the DoD and commercial user community with the ATF JIAWG reuse technology; the frameworks and object-oriented database (OODB) in the STARS program; the test and maintenance domain in ALC; and the multidomain CITIS environments in CALS/CE.

The Prototype

The PRototype EIS (PREIS) provides users with a subset implementation of an EIS. The PREIS implements more than 250 object types. It demonstrates both the viability of the EIS specifications and some ways to improve them. The PREIS successfully combines off-the-shelf, first-generation frameworks to meet the EIS specifications. This is a critical issue that many vendors face as they try to adhere to an evolving industry-standard framework. The following list is representative of the capabilities that are implemented in the PREIS framework:

- Schema management model,
- Application object model,
- Function model,
- Session management and access control,
- Control points and policies,
- Network communications,
- Common exchange format model,
- User tools.

Ten key types exist in an EIS; familiarity with these types is essential to understanding the functionality of an EIS. In keeping with the object-oriented model, all kinds of data objects known to the framework are defined as types. Each type acts as a model of some real-world object (e.g., a file, a program, a user). New types are defined in terms of existing types (supertypes). When a new type (subtype) is defined in terms of a supertype, the new type inherits characteristics of its supertype and typically has additional slots and functions to distinguish it from its supertypes. This reuse of existing type definitions reduces the effort required to define new types.

A data slot is an object that is used in defining the structure of a type. A slot object contains information relevant to its use (e.g., which type it is defined as part of and what types of values an instance having the slot can be set to). The slots slot on a type contains all of the attribute and relationship slots defined for that type. A type can have any number of slots; these slots can have any kind of value from a Boolean value to a list of objects. There are two kinds of slots—attribute and relationship. Attribute slots can contain only simple values while relationship slots can contain more complex structures.

A function is an object that is used to define the behavior of a type. Specification of a function describes its semantics and provides an interface for invoking it. A function can have any number of implementations and can perform any activity that the definer would like the function to perform. The three types of functions in the PREIS implementation are functions, express functions, and polities.

Application Demonstrations

The EIS concept covers both domain-specific information modeling (electronic, mechanical, and software engineering) and domain-independent framework services (versioning, configuration management, object management, etc.), as shown in Figure 8. The objectives of the potential demonstrations discussed in this section are to apply the generic EIS framework technology to various applications and to integrate the EIS information model with other domain-specific information models.

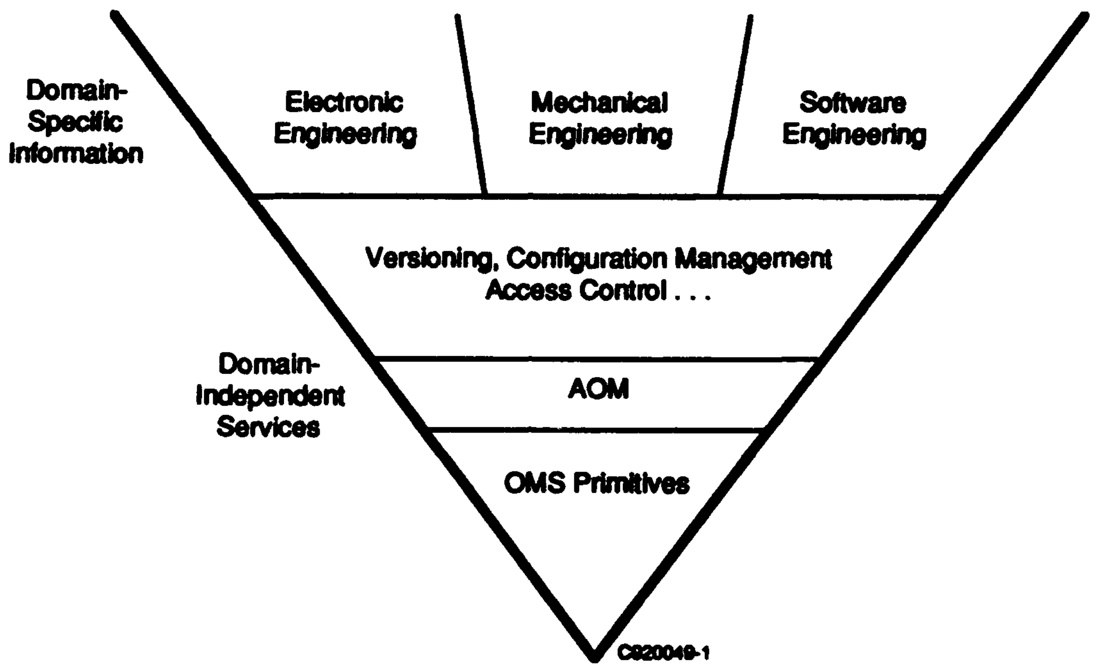

Domain-Specific Information
Electronic Engineering
Mechanical Engineering
Software Engineering
Versioning, Configuration Management Access Control . . .
Domain-Independent Services
AOM
OMS Primitives
C920049-1

Figure 8. Scope of the EIS

The potential demonstration examples presented in this section will illustrate how the various engineering domains can be integrated and how the variety of problem areas associated with each domain can be resolved. The EIS application areas and engineering domains are shown in Figure 9; the areas for demonstration are highlighted. Potential demonstrations include application and demonstration for the following environments:

- Electronic System Design (ECAD),
- Microelectronics Manufacturing Systems Technology (MMST),
- Product Data Exchange Standard (PDES) testbed,
- Microwave/Millimeter-Wave Monolithic Integrated Circuit (MIMIC),
- (Advanced Tactical Fighter)/Joint Industry Avionics Working Group (ATF/JIAWG),
- Air Logistics Center (ALC).

Although the EIS was originally designed to perform tool interoperability and data exchange for the ECAD domain, the EIS can be adapted to the needs of many different applications. Most adaptation is done at the information modeling level; however, the EIS framework can be modified, if desired, to meet the site-specific needs.

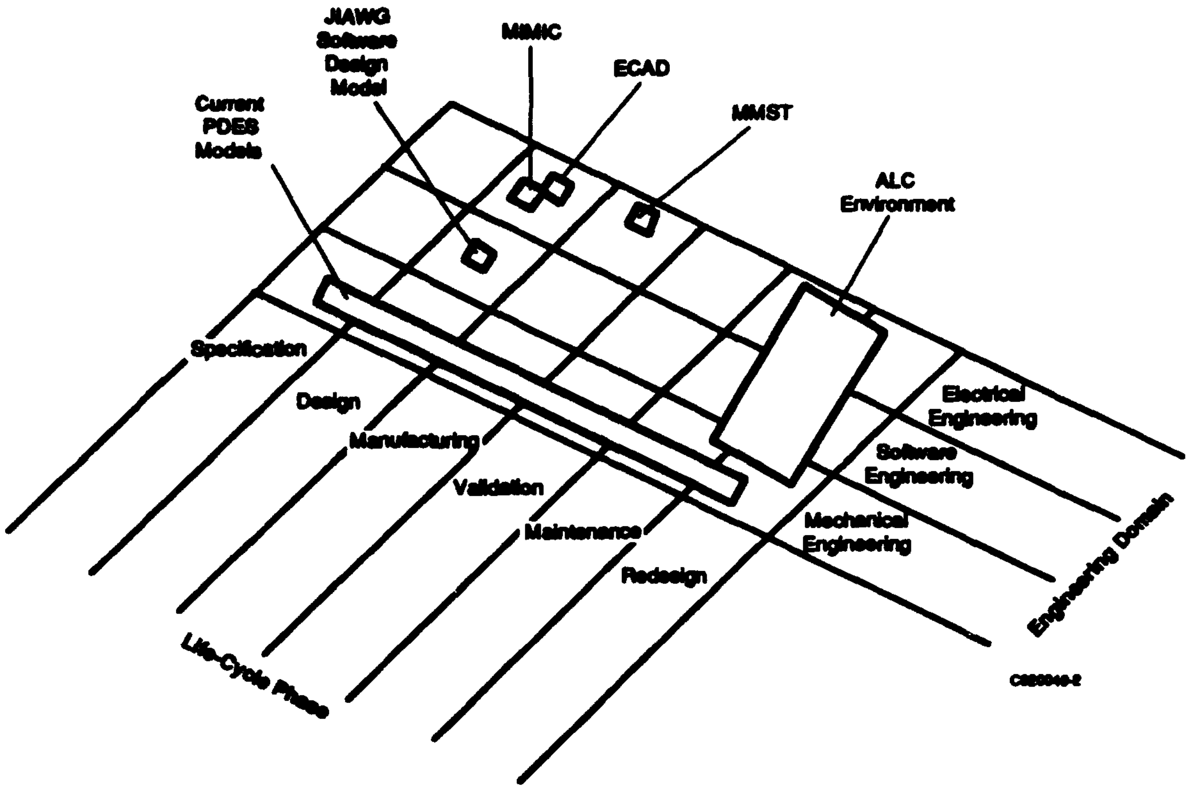


Figure 9. EIS Application to Engineering Domains and Life-Cycle Phases

Evolution of the Engineering Environment Marketplace

The EIS program has had significant success generating demand for standards. The results—the specification, prototype, and demonstrations—will promote evolution of conforming information models, frameworks, and engineering environments. Participants in the program and other vendors and users will implement parts of the EIS, continuing the development of the EIS concepts toward implementation and further refinement and usability of the concepts.

Members of the EIS team have actively participated in the CAD Frameworks Initiative Design Representation Subcommittee (CFI DRTSC) since February 1989. Attendance at subcommittee meetings averages from 20 to 25 members. Active input has been received from MCC, Cadence, IBM, Mentor, and EIS. ECAD engineering information model meetings have been held in conjunction with the CFI meetings, and input for development of the ECAD model occurred with periodic input and review by CFI members. Four walk-throughs of the ECAD model were conducted at DRTSC meetings. The EIS team has provided active input to the CFI DAC-90 and DAC-91 models; the ECAD model "instance hierarchy" subset is semantically equivalent to the CFI model. The CFI modeling notations (IDEF1X and EXPRESS) are compliant with the engineering information system ECAD model. The ECAD model has wider scope than is desired by CFI vendors for the 1991 demonstration. The engineering information system ECAD modei is currently being distributed as a strawman CFI document by MP Associates.

EIS team involvement has been evident in the Product Data Exchange for STEP Electrical Activities Committee (PDES EAC) since April 1989. Active input in this committee has been from PDES Inc., NIST, Boeing, Navy RAMP, and EIS. Average attendance at these meetings also ranges from 20 to 25 members. Four engineering information system ECAD model walk-throughs were conducted at PDES EAC meetings. PDES passed a resolution in their July meeting in Portland to adopt the ECAD model as an EAC strawman for integrated circuits. The ECAD model is the first outside model to be accepted by PDES EAC. PDES is currently working on a formal process for model review and standardization.

The EIS relationship with IEEE WGIM (IEEE DASS Working Group on Information Modeling) was active in the revitalization of the IEEE PAR on information modeling in June 1989. Five meetings of this group have been held, with about 15 to 20 members attending each meeting. The charter of this committee is to address model collection and federation in the electrical area—VHDL, EDIF, IPC, WAVES, etc. The ECAD model has been submitted to the group as a model for federation of VHDL and EDIF. The group is currently awaiting resource models from various groups such as VIFASG, EIA, WAVES, and CFI. Preliminary memorandums of understanding between WGIM and other groups have been drafted.

Section 2
Development of Candidate Standards

A series of community workshops culminated in two cornerstone documents—the *EIS Concepts* and *EIS Requirements*. These documents condense the EIS objectives to the set of nine shown in Figure 10. These objectives drive design of the EIS.

1. Provide a cost-effective tool integration framework.
2. Encourage portability of tools.
3. Encourage a uniform design environment.
4. Facilitate exchange of design information.
5. Support design management and reuse.
6. Achieve widespread acceptance of EIS.
7. Be adaptable to future engineering.
8. Be extensible to other disciplines.
9. Be compatible for transition from existing design environments.

Figure 10. EIS Objectives

It is important to note that not all projects that deal with an EIS-type problem (engineering information management, support, and data exchange) are driven by these same objectives and, therefore, can and do result in different frameworks, environments, and models. It is often not immediately obvious why particular EIS features or even the overall architecture differs from other efforts, but it becomes clear when a feature is tracked back to the motivating objective.

For EIS, the objectives have the design and scope effects shown in Figure 11. Two effects emerged as having particular significance for the design effort: (1) separation of domain-independent and domain-dependent parts and (2) specifications that must allow upward migration for tools and databases. Although they made the design more difficult in some ways, they proved to be two of the most widely appreciated and valuable characteristics.

EIS Requirements

The EIS requirements (as published in the IDA documents) are established as follows:

- Tool/workstation integration,
- Data exchange,
- Engineering management and control,
- Information management,
- External system interfaces,
- Object management system,
- Distributed operating system,
- Distributed data management facilities,
- Engineering information model,
- Rule processing,
- Control points,
- User interface,
- EIS administration,
- Programmatic interface,
- Design and implementation requirements.

1. **Provide a cost-effective tool integration framework.**
   → Existence of framework
2. **Encourage portability of tools.**
   → Insulation from host platform
3. **Encourage a uniform design environment.**
   → Existence of specifications standard and user interface guidelines
4. **Facilitate exchange of design information.**
   → Common exchange format and exchange services
5. **Support design management and reuse.**
   → Design management services
6. **Achieve widespread acceptance of EIS.**
   → Specifications must be a reasonable extrapolation of existing/emerging technologies
7. **Be adaptable of future engineering.**
   → Rigorous parts-based architecture and separation of description from execution.
8. **Be extensible to other disciplines.**
   → Separation of domain-dependent and domain-independent parts.
9. **Be compatible for transition from existing design environments.**
   → Specifications must demonstrate viability of using existing framework/database/ operating system bases and be able to host existing tools.

Figure 11. EIS Objectives Design Influences

The requirements groupings are summarized below, with a brief overview of each topic area:

- *Tool/Workstation Integration*—Requirements include the ability to invoke and monitor tools on both the EIS host platform and workstations networked to the host as part of the EIS installation. This includes flexible features for scheduling execution and supplying data for execution, as well as automated program networks.

- *Data Exchange*—General data exchange facilities, including assembly and disassembly of data, defining a format for data exchange, and automatic invocation of translators are required. Administrative information relating to data should be able to be exchanged.

- *Engineering Management and Control*—Comprehensive management facilities, including configuration management, dependency tracking, change control, version control, audit trails, and security and access control, are defined.

- *Information Management*—Basic object handling capabilities, including object creation and deletion, object identifiers, automatic versioning, concurrent access, and conflict management, are required.

- *External System Interface*—Data exchange across EIS boundaries must be provided.
- *Object Management System*—IDA documents require an object-oriented approach.
- *Distributed Operating System*—An EIS must be capable of accessing system resources in a distributed, heterogeneous environment, including scheduling, interconnection, monitoring, and exception handling.
- *Distributed Data Management*—Data distribution must be supported, including mapping of data access requests. There must be a schema supported by a model comprised of object descriptions. Requirements further define details of the object model and specific features.
- *Engineering Information Model*—The EIS must have an information model composed of both ECAD-specific information and administrative information. Requirements for methodology and language are included.
- *Rule Processing*—General rule processing requirements and accessibility to parts of the EIS are described.
- *Control Points*—Association of control points with events to cause policy is required.
- *User Interface*—Guidelines and requirements for a standard, but tailorable, user interface and a user interface processing system are provided.
- *EIS Administration*—Requirements are listed for object description and tailorable policy.
- *Programmatic Interfaces*—An EIS requires specification of interfaces to the major components, including language bindings for Ada and C.
- *Design and Implementation Requirements*—Portability and networking requirements are defined.

Using these requirements groupings, topics are covered in the *Organization and Concepts* and *Specifications and Guidelines* documented as shown in Table 1.

User's Guide to the EIS and Its Documentation

A challenge that accompanied the formulation of EIS presentations and documentation was identifying the audience. Five overlapping communities are expected to use this document: end users, tool providers, system integrators, EIS suppliers, and standards bodies. With no time and resource constraints, a set of EIS documents could be written for each group, even though there would be overlap in coverage.

Table 1. Correlation of Requirements Categories and EIS Topic Areas

Requirements Categories	EIS Topic Areas
Tool/Workstation Integration	Object Management System (OMS)
Data Exchange	Common Exchange Format and Inter-EIS Data
Engineering Management and Control	Exchange (EES)
Information Management	Engineering Environment Services (EES)
External System Interface	OMS
Inter-EIS Data Exchange (EES)	Network Extensions to Portability Services,
Object Management System	OMS
Distributed Operating System	OMS, EES, Portability Services
Distributed Data Management	OMS
Engineering Information Model	Engineering Information Model (EIM)
Rule Processing	Rule Processing (EES)
Control Points	Control Points in EES
User Interface	UIMS
EIS Administration	EES
Programmatic Interface	All
Design and Implementation Requirements	Portability Services

The *Organization and Concepts* and *Specifications and Guidelines* documents are organized by local components rather than by user group. Each major component is described in detail, without reference to particular users. As a result, an individual proceeding serially through the documents may find it difficult to gather the information needed to evaluate the system.

Tables 2 through 5 summarize major areas of interest for each of four primary groups of readers. The tables also provide a brief summary of how the group might use a particular topic area. The tables can be used by readers to determine sections of interest in the *Organization and Concepts* and *Specifications and Guidelines* documents.

Organization and Concepts Document

The primary objectives of the *Organization and Concepts Document* are to describe the various EIS components and to discuss how the components fit together and how the sum of the components behave. It is suggested that the *Organization and Concepts Document* be examined as a path into the *Specifications and Guidelines* and the *Engineering Information Model* documents.

Table 2. Areas of Interest to EIS Suppliers

Component	Specific Topic	Interest
AOM	Application Object Model	The EIS supplier must export the AOM.
OMS		The EIS supplier must export visible OMS services. The rest of the OMS specification establishes semantics of the exported interface.
EES	All Services	An EIS supplier may supply some or all of the EES as part of an EIS base installed product.
UIMS		The EIS supplier may wish to include a UIMS as part of an EIS product.
Portability Services		Using portability services in building an EIS maximizes portability across platforms.
EIM	ECAD, ADM	The EIS must have a skeletal reference schema and may include a comprehensive schema as part of an EIS product.

Table 3. Areas of Interest to EIS Tool Providers

Component	Specific Topic	Interest
AOM	Application Object Model	Tool providers must interface their tools or tool adapters to the EIS-exported interface (the AOM).
OMS	Interface to Function Application	Tool providers may wish to build OODL scripts within their tool(s) or adapters.
	Dynamics of Function Application	Tool providers must understand the basic execution paradigms of the OMS.
	Database and File Adapters	Existing data repositories must be interfaced to the EIS if a tool provider wishes EIS management
EES	All Services	Tool providers may wish to use some or all of these services to supplement or replace functionality in the tool.
UIMS	Guidelines	Guidelines for a uniform tool user interface.
	Interaction objects	Tool providers can use the set of provided interaction objects or tailor/extend/add to them.
	UIMS languages	Languages are provided to support building UI scripts and new interaction objects.
Portability Services		Tool providers can have a more portable tool if they build on these services.
EIM	ECAD, ADM	Tool providers can interface directly to EIS-managed objects or adapt to them. Tool providers may require EIS extensions/tailoring to support a specific tool's needs.

Table 4. Areas of Interest to EIS End Users

Component	Specific Topic	Interest
AOM	Application Object Model	The AOM is the EIS view of objects—how data and operations are organized. Tools will also export this view; it is important for end users to be comfortable with these concepts.
OMS	Interface to Function Application	Provides end user with a basic understanding of EIS operation.
	Dynamics of Function Application	End users may wish to build OODL scripts to create APNs, for example.
	Schema Management	End users should understand sufficient aspects of schema operation to be able to specify their site-specific needs.
EES	Control Points and Policies	End users should understand how site-specific policies are mechanized.
	Configuration Management Access Control, Audit Trail, Backup-Archive, Repositories	End users need to understand basic structural aspects of engineering management in EIS.
	CEF, Interchange Services	End users will wish to send and obtain data between EISs.
UIMS	Guidelines	Provides a description of the "flavor" of the EIS interface.
EIM	ECAD, ADM	End users are most interested in the objects and operations available to them in an EIS.

User Interface Management System (UIMS)

The UIMS is a set of design guidelines and computer programs that guide and support the development, execution, and maintenance of consistent user interfaces for designers, program managers, and administrators over a range of computer terminals. It is similar to a database management system (DBMS) in many respects. Just as a DBMS manages the "back-end" data storage and retrieval functions of a software tool, a UIMS manages its "front-end" capture and display of data on terminals and workstations. A DBMS provides application-data independence (the ability to change the structure of data without changing the code of applications that use the data), whereas a UIMS provides application-user interface independence (the ability to design, implement, and modify the structure and form of the user interface independent of the computation code of the application). Just as databases have design guidelines, user interfaces have guidelines for absorption of new tools and the accommodation of existing tools that supply their interfaces.

Table 5. Areas of Interest to EIS System Integrators

Component	Specific Topic	Interest
OMS	Schema Management	EIS system integrators will build site-specific schema using these functions.
	Interface to Function Application	System integrators will build scripts of functions to cause multiple-step processes to execute.
	Dynamics of Function Application	Provides system integrators with a basic understanding of EIS execution.
	Database and File Adapters	System integrators will wish to attach data servers to the EIS (those with existing design data, for example).
AOM	Application Object Model	The AOM is the EIS view of objects. All work the system integrator does to attach tools and data and to tailor the schema services will comply with this model.
EES	Control points and policies	This is the basic mechanism system integrators will use to implement site-specific policies and methodology enforcement.
	Configuration Management, change control, access control, audit trail, backup/archive, repositories	The basic objects and operations which a system integrator can use to build site-specific policies and management systems are described.
	Inter-EIS Exchange	The format and services through which the EIS interfaces to other EISs are specified.
UIMS	Guidelines, Interaction objects, UIMS languages	Interfaces to the EIS management facilities (EES) and new tools will use these services.
EIM	ECAD, ADM	System integrators need to understand how to do site-specific tailoring of the models and how existing tools, data, and processes fir with EIS-specified models.

User interfaces can be thought of as communication paths between users and computers. This user-computer dialog has three components: scripts, noninteractive core (NIC) functions, and interaction objects, as illustrated in Figure 12. In this example, the dialog solicits instructions from the user via a menu interaction object, invokes the DBMS NIC function to access a database, invokes a computational NIC function to calculate summaries, and then displays the result in a data box interaction object. Descriptions of each user-computer dialog component are provided to aid in understanding these interactions:

- Scripts describe dialogs and can be used to accomplish such tasks as:
  - Editing, whose steps are embodied in a single software package.
  - Letter writing, whose steps are supported by several software packages such as a text editor, an electronic mail system, printing routines, a text formatter, and library management routines.

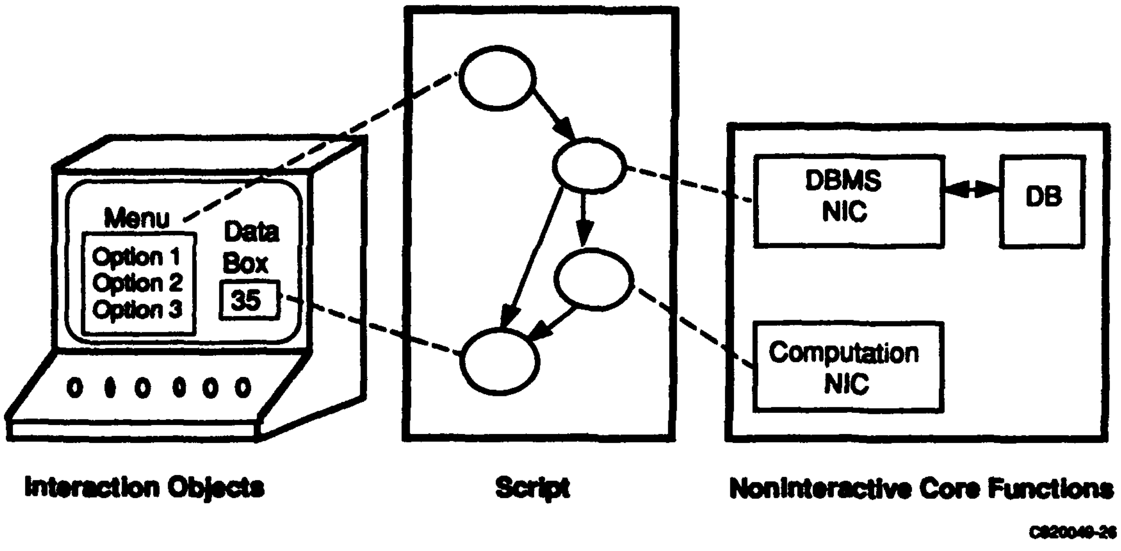


Figure 12. User-Computer System Dialog Components

- Login, whose steps invoke operating system functions rather than application software.

Scripts have several styles that reflect types of activities, ranging from proscriptive (requiring the user to go through steps in a predefined sequence) to flexible (in which the user, rather than the system, selects the sequence of steps).

Scripts represent dialogs as event-driven transition systems (EDTs) that comprise dialog-states and transitions. Each dialog-state corresponds to a task step. To perform the task step, the dialog-state may solicit user input; accept user input and verify that it meets range and type constraints; invoke NIC functions; and/or display results computed by NIC functions or warning and error messages when NIC functions cannot complete their computations.

Dialog designers write scripts by designing the behavior and appearance of each dialog-state and interconnecting the states. They can design each aspect in any order, shift among aspects, and interleave design activities as desired. The user interface development environment can support many possible methods for designing dialogs.

- NIC functions perform task steps. They require zero or more input parameter values and return one or more results or side effects at their processing completion.

- Interaction object attachment specifies the appearance of dialog-states. They accept user input and display the results and status of computations, prompts, or help/error messages. Some examples of interaction objects are menu, command-line, form, type-in box, and button.

Consistent user interfaces among applications in the EIS program are promoted by requiring all interfaces to use common interaction objects that adhere to the EIS user interface management system guidelines. This increases system interoperability—the ability to apply the user-system interaction techniques of one system to other systems. Interoperability also decreases the time required by a user to learn to use new systems. The reuse of common interaction objects and script fragments lowers the cost of developing interfaces.

Other features include a set of user interface guidelines, a library of reusable interaction objects and scripts, interactive script authoring and interacting object tailoring tools, and an approach for retrofitting existing applications with interaction objects and scripts.

The engineering information system UIMS offers many benefits to tool developers, interface designers, and end users:

- *Reducing Development and Modification Effort*—The effort that currently goes into building user interfaces can be significantly reduced through the use of a high-level user interface development language. Primitives of this language allow declarative specification of the interface details. Without this capability, interface details must be specified through multiple procedure calls as in a general-purpose programming language. For example, display details (the number of lines on a screen, screen position of an object, or syntax of a command being captured) can be specified as parameters to a display primitive.

- *Dialog Independence*—Dialog independence is the separation of tool user interface design and implementation from the design and implementation of its computational elements. Dialog independence permits prototyping of the user interface of a tool before or in parallel with development of its computational software. This promotes good interfaces by allowing designers to test their decisions with users. The decision can then be inexpensively and quickly modified until users and designers converge on a desirable interface. Dialog independence also permits separate modification of the user interface and the computational software during the lifetime of a tool. Complete dialog independence may never occur, but the engineering information system UIMS offers a high degree of independence.

- *Reusability*—There are three kinds of reuse. Language primitives that provide user interface functions reuse the design and code that implement them. Second, portions of the user interface of a particular tool that are not intertwined with the design or code of a computation may be reused. Finally, reuse of common functions can be achieved by populating the EIS with application building blocks that can be combined with a UIMS that accesses them in a scripted way to deliver complete applications.

- *Interface Standardization*—From the point of view of the user, one crucial measure of the usability of an environment is the look and feel of its user interface. A consistent appearance and interaction style with the system and all its component tools increases transfer of training, improves user performance, and reduces stress for users who must move among tools. Organizations using tools from various vendors desire those tools to have consistent interfaces. A UIMS promotes consistency with standards and reusable interface components.

Domain Schemas

Domain schemas are the primary vehicle for extending the EIS framework to support domain-specific tools and data. A schema was developed to model the information used by tools based on both EDIF and VHDL electronic computer-aided design.

The EIS electronic computer-aided design EIM was primarily derived from concepts in electronic hardware description languages, but additional input came from examining approximately 20 tools that are used throughout the electronic design life cycle. The ECAD engineering information model serves as a canonical form of the exchange of digital structure and physical layout information between electronic design tools. The current model contains nine conceptual areas:

- *Design*—Stores information required to support a design effort, including the design name, status, and team members.
- *Cell Instance Hierarchy*—Stores information about a design hierarchy, including details about how cells are hierarchically composed of other cells.
- *Cell Occurrence Tree*—Stores occurrence information that is specific to each cell.
- *Configurations*—Stores information that specifies how to associate a cell with one of many implementation alternatives.
- *Schematic Information*—Stores information relating to circuit schematics composed of many schematic pages, where each page contains symbols representing cells.
- *Layout Information*—Stores information about a circuit layout and its hierarchical decomposition into more primitive layouts and interconnection structures.
- *Geometric Shapes*—Stores information about shapes used to construct geometries used in schematic and layout information.
- *Library and Technology*—Stores information about the grouping of design information that shares common characteristics and information tied to the manufacturing of electronic designs.
- *Rules*—Stores information about a variety of constraints, requirements, and restrictions that apply to the parts of the design and design process.

Other projects are also developing domain models of various detail. For example, PDES is developing models for mechanical products and, more recently, electrical products. Much of the Product Process Organization model of the DARPA Initiative in Current Engineering also fits into this layer.

Engineering Environment Services

Engineering support services encompass a broad range of services of interest to application developers or end users. Topics include policy support, change and configuration management, design history, network communication, inter-EIS information exchange, audit trailing, access control, work-space management, and tool coordination. The Phase 2 prototype implemented policy support, network communication, inter-EIS information exchange, and a portion of work-space management.

The EIS control point/policy mechanisms provide system integrators and users a means for tailoring function behavior. A control point is an object that contains an on/off switch, a trigger condition, and a set of synchronous and asynchronous policies to be executed when the control point switch is on and its condition evaluates to true. A policy is an arbitrarily complex function that encodes some facet of conduct enforced by the system.

The EIS function application model partitions the execution of a function into three phases: an entry event, invocation of the implementation of the function, and an exit event. During a function's entry and exit events, the set of attached control points are evaluated and policies associated with control points that evaluate to true are invoked. Control points may be dynamically attached to a function by simply adding them to its entry and exit lists. An attached control point may also be activated or deactivated by setting its switch to true or false, respectively.

Control points and policies have been used extensively in the EIS integration experiments. They are also used to implement a very fine-grained access control model in the EIS prototype.

The EIS program defines a set of services for intertool and intersite communication. Network communication services provide an extensible set of primitives for establishing connections and transmitting data. Both TCP and UDP protocols are currently supported. The common exchange format (CEF) provides a data/protocol-independent, byte-stream (printable ASCII) form for representing data. Finally, the inter-EIS information exchange (IEIE) services provide a high-level protocol for exchanging data between EIS sites and for remote function application.

Object Management System

The OMS is responsible for providing access to objects and operations. In a sense, it is the common "bus" to which everything connects, including tools and data servers. By being on the same bus (sometimes via adapters), tools and their users more readily gain controlled access to data in the various servers.

The OMS is a collection of EIS functions that are responsible for managing the schema that describes all objects and operations available through an EIS and for managing and executing requests for EIS services. Major OMS subfunctions are session management, requests, views and access control, function application, control points and policies, and function implementations.

Session Management—The EIS schema defines all objects and operations accessible to clients at a given EIS installation. It describes the EIS configuration objects and the domain-specific data objects. In this way, the EIS is used to manage operation of the system. Schema definitions are interfaces used by EIS tool builders to register their tools, data, and data exchange formats. The goal is to provide common constructs for their use in defining the data objects used by their products, allowing them to be ported across EISs.

Requests—A request, which specifies a function and a set of arguments, is the means by which behaviors are invoked. A function may be requested by name or by its identifier (OID), and it may be "directed." The output arguments of a request are returned via the request object. A Request type forms the basis of the function application mechanism. A "directed" function request provides a clue to the resolution mechanism regarding how to view the type of the first argument. The Request type significantly simplifies the mechanics of argument passing and opens the door for many possible implementations (e.g., apply versus initiate) and capabilities (e.g., claim).

Views and Access Control—A view specifies a unit of visibility and is the mechanism by which access control is implemented. Access control is a specified policy or discipline whereby the ability of a client to apply functions to objects is limited or constrained based on the identity of any combination of client, function, and/or object.

A view provides visibility to a function group, a domain (object) group, and a range (object) group. Each group is defined in terms of a "membersTest" function. Each user has an associated "defaultRole" (a user group) that identifies an "assignedView." A function may have an "assignedView"; such an object is called a "registeredClient." A membersTest function takes a group and an object as input and returns a Boolean that indicates whether the given object is a member of the given group.

Function Application—Function application is the execution paradigm of the OMS. The function applier evaluates requests and produces results. Requests may be processed synchronously (apply) or asynchronously (initiate). Clients solicit services by requesting application of a function to arguments. The semantics of function application include mapping the request to the services that implement it, enforcing the appropriate control semantics, and handling executing events.

Mapping the request to the services that implement it involves selecting one implementation of the function, mapping the input arguments that are supplied with the function request to parameters expected by the selected function implementation, and invoking the selected implementation.

Enforcing the appropriate control semantics includes ensuring that function argument types are compatible with the signature of the function within the context of the view associated with the session of the requesting client; associating and enforcing transaction management semantics with the request as specified for the requested function; applying the entry function before initiating invocation of the selected function implementation; and applying the exit function upon completion of the selected function implementation.

Handling execution events involves handling the exceptions raised by the executing function implementation; posting outputs returned by the executing function implementation; and initiating transaction recovery and completion semantics upon request failure or successful completion, respectively.

Control Points and Policies—A control point is a conditional trigger for attaching policies to functions. A policy is a special function with a predefined signature that modifies system behavior. Taken together, control points and policies factor out administrative/methodological functionality from tools (i.e., functions). They also support the uniform enforcement of administrative/ methodological functionality across all tools.

A control point may be attached to the entry or exit events of one or more functions. It has a Boolean on/off switch (activated) and an optional condition (a Boolean-valued function that may enable/disable the control point). A control point has an associated set of synchronous policies and an associated set of asynchronous policies. If the synchronous policies are active and enabled, the control point applies each synchronous policy. If any policy returns false, the control point inhibits further application of the function to which it is attached. If the asynchronous policies are active and enabled and all synchronous policies return true, the control point initiates each asynchronous policy and does not wait for results. Application of the function to which the control point is attached then resumes.

Policies (and express functions in general) bypass the normal function application semantics and are hardwired to an implementation (i.e., no resolution and no control points). The signature of a policy consists of a name, an input parameter of type Request, an input parameter of type ControlPoint, and an output parameter of type Boolean.

Function Implementations—A function implementation represents an executable piece of code that implements some behavior. An implementation runs within the context of an "execution engine" provided by the invoke function. An implementation implements one or more functions. There may be kinds of implementations (i.e., specializations) that require different execution engines. An implementation may also serve as a tool adapter.

Application Object Model

The AOM is actually an extension of the OMS. The EIS application object model is a fairly typical model for describing and manipulating objects. It provides a fixed, high-level view that supports tool portability. The AOM intentionally shares many features with existing object models (e.g., Smalltalk-80, the Common LISP Object System [CLOS], and C++). The primary reason for the genericity of the AOM is that many different languages, some object oriented, will need to be integrated into an EIS.

AOM objects are described by types, and have three essential characteristics: they exhibit behavior, they have identity, and they have state.

- *Type*—An AOM type is very much like an abstract data type. It defines a set of functions that can be applied to its instances. AOM types can be arranged in an inheritance graph (a directed lattice), whereby one type can inherit the definition of one or more supertypes. In an EIS, types are themselves modeled and stored as objects, giving rise to the need for metatypes (i.e., types that describe types). Metatypes are also kept as objects in an EIS, but the recursion stops there; no further meta levels exist in the AOM view.

With instances, types, and metatypes being represented in a uniform way, EIS has significant flexibility, extensibility, and tailorability. In many cases, providing a new capability to a large collection of objects may involve simply adding a new function to a type somewhere toward the top of the type lattice.

- *Object Identity*—Each object has its own identifier, called an OID (object identifier), which is assigned at the time it is created. Once established, an OID always refers to the same object.

  An object handle is an indirect reference to an object (e.g., a query or a name). Suppose an application has two different handles (α and β). If α resolves to the same OID as β, then α and β are handles for the same object. Two objects are said to be equivalent if, and only if, their states are identical—that is, if the contents of the attributes and relationships of one match those of the other. Objects that are equivalent are not necessarily identical (the same object).

- *Object Behavior*—Object behaviors are modeled by functions and implementations. A function is an object that represents an interface to a behavior. The essential attributes of a function object are its signature and a set of implementations. The signature identifies the name of the function and its input and output parameters (and their types). An implementation is an object that represents a body of code that implements the behavior specified by the function.

  Behavior is elicited from an object by applying a function to it. A function request specifies the desired function (by name or object identifier) and a set of input arguments (objects) and identifies the number and type of expected output objects (results). By convention, the object given as the first argument in a function request is called the receiver.

- *Resolution*—Given a function request, there may be one or more functions specified for the type of a receiver (or one of its supertypes) from which the system may choose to serve the request. There may be one or more implementations that use the behavior specified by the function for any given function. Resolution is the process used to select a function and an implementation capable of satisfying a given function request. In an EIS, resolution is a two-phased process:

  - The first phase, corresponding to traditional (object-oriented) parameter-based resolution, uses the function name and arguments specified in the request to select the most specific function with a matching signature.

  - The second phase, called "environmental" resolution, considers factors such as host architecture and operating system to select an appropriate implementation for the selected function.

Once an implementation has been selected, it is invoked on the given arguments. A function implementation itself may make further requests for function application.

As an example, consider three hypothetical types: Design and its two subtypes, SoftwareDesign and HardwareDesign, with the following functions:

Design:
- edit(instance)—start a tool that can be used to edit the design
- simulate(instance, stimulus)—run a simulator on the design using the given stimuli

SoftwareDesign(inherits Component):
- edit(instance)—invokes an editor specific to software design
- simulate(instance, stimulus)—invokes a simulator on the software design with the given stimuli
- build(instance)—assemble the implementation of the design so it can be executed

HardwareDesign(inherits component):
- edit(instance)—invokes an editor (for an electrical or mechanical design)
- displayOn(instance, medium)—graphic display of the instance on the specified medium

In this example, Design is an "abstract" type that specifies that its instances will exhibit the `edit` behavior. Further, the `build` function can be executed on an instance of SoftwareDesign but not on an instance of HardwareDesign. Similarly, `displayOn` is specific to HardwareDesign. Note that SoftwareDesign and HardwareDesign both specialize the inherited `edit` function.

The user need not specify the precise function to apply; the system resolves the function request to the appropriate function and implementation based on the arguments given in the request. If an instance of HardwareDesign is passed as an argument to `displayOn`, the first phase of resolution (parameter-based) is trivial since only one function with a matching signature is available. Selecting an implementation may be more complex; it is likely that many implementations of the displayOn behavior will exist.

If an instance of SoftwareDesign is passed as a parameter to `edit`, the system will select the function defined for the SoftwareDesign type. Similarly, if an instance of HardwareDesign is passed, the function defined for HardwareDesign will be selected. If an instance of a different type is supplied that has no edit function, the system will reject the request and raise the "functionNotFound" exception (a compiler could catch this error). If the system cannot resolve a request to a single function, it raises the "functionResolutionConflict" exception (again, a compiler could catch this error). An EIS installation might choose to handle such exceptions by completing the resolution based on an ordering, such as supertype ordering or chronological ordering by registration time. A framework vendor might choose to provide a default exception handler that could be removed or replaced according to the needs of the site.

The edit function described above has a single parameter. To see how multiple parameters may be used in resolution, consider three different `displayOn` functions for another hypothetical type MechanicalDesign. One expects the `medium` parameter to identify a Sun workstation, another expects an Apollo workstation identity, and the third expects the logical name of a laser printer. Each function takes an instance of HardwareComponent and a format, as follows:

displayOn(aMechanicalDesign, sunHost)—displays an instance on a Sun workstation
displayOn(aMechanicalDesign, apolloHost)—displays an instance on an Apollo workstation
displayOn(aMechanicalDesign, lasername)—prints an instance on a laser printer

Upon receiving a request for `displayOn`, the system first examines the receiver (first argument). In this case, the type of the receiver narrows down the set of applicable functions to the above three. Examination of the second argument carries the resolution to a single function.

The AOM offers several ways to make function requests (e.g., by name or by function OID). A programmer's choice of one over another depends upon how much is known about the desired function at execution time.

Engineering Information Model

The EIM is an integrated conceptual model of electronic design information. It is a semantic model that documents the meaning of concepts that are held in common and makes the meaning of the concepts explicit. The overall scope of the EIM is focused on the design phase of the engineering life cycle.

While conceptual models generally evolve naturally and can be quite unconstrained, the EIS program requires a more formal approach. It cannot be assumed that all EIS users are from the same discipline and have worked closely; however, that level of understanding must be captured and shared with persons from various disciplines working in separate organizations and dispersed geographically. For this reason, the EIM uses formal information modeling constructs and is carefully organized to address specific well-defined topics.

The EIM is an important reference document for everyone involved in the EIS program. The EIM would be used to find common names and definitions for the information a CAD tool builder expects to process. A framework vendor would use the EIM to identify common operations and would consider building them into the framework. An administrator of a CAD environment would use the EIM to quickly understand the scope and organization of the EIS information to plan adaptations of the current system as the implementation of EIS concepts begins.

The EIM is a computer-processable form that is directly accessible to CAD systems and tools. The two major components of the EIM are the administrative domain model (ADM) and the ECAD domain model. The scope of these models is illustrated in Figure 13.

The EIM development approach is based on conventional information modeling techniques, with some critical improvisations and departures. The IDEF1X methodology, developed by DACOM, is the strongest single influence; however, departures from this methodology are significant.

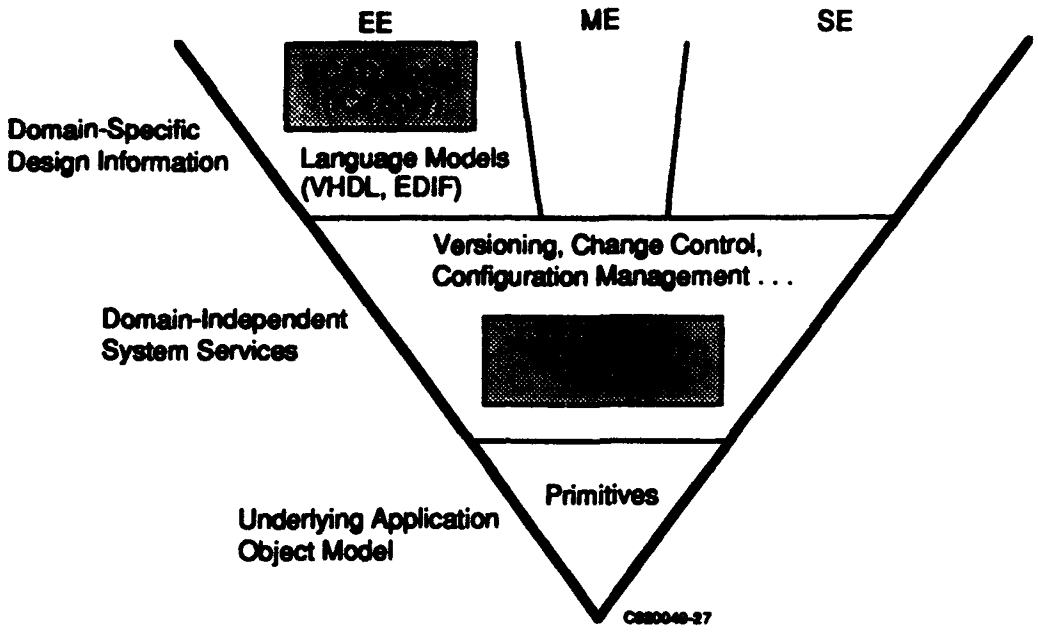

EE
ME
SE
Domain-Specific Design Information
Language Models (VHDL, EDIF)
Versioning, Change Control, Configuration Management . . .
Domain-Independent System Services
Primitives
Underlying Application Object Model
C880040-27

Figure 13. Scope of ADM and ECAD Models

One set of concerns relates to the object-oriented nature of the EIS. While object-oriented design offers potential and received emphasis in the EIS requirements, it does not yet follow a mature methodology. The graphic notation of the ADM reflects a compromise between the mature, widely used notation of IDEF1X and the object-oriented nature of the EIS requirements.

Another concern relates to the limited success traditional information modeling can claim for engineering domains. This concern had the greatest impact on the ECAD model. Because it was determined that a static model that contains only data would have been inadequate, operations were also modeled.

Administrative Domain Model—Domain-independent management and control topics are the focuses of the ADM—general capabilities that will be provided by an EIS framework. The capabilities are basically the same for electronic and mechanical design engineers (e.g., "versions, alternatives, and revisions" should have one consistent meaning and be implemented as a general capability regardless of the domain).

Discussions of scope in each topic of the ADM tend to focus on interpretation or reformulation of specific EIS requirements. The ADM topics addressed for the four user groups are:

- *System Implementor*—Data exchange formats, data exchange services, error handling, error management, parameter description, program networks, rule modeling, and tool invocation.

- *System Administrator*—Administrative data of a tool, backup, communication among hosts, concurrent data access, and recovery.

- *Data Administrator*—Archive, audit trail, change control, configuration, notification, policies, control points, and access control.
- *End User*—Decomposition and representation, methodology, rule processing and exchange, and version, alternative, and revision.

ECAD Domain Model—The EDM focus is on providing an integrated conceptual model of digital integrated circuit (IC) design information and is specific to the digital IC design domain; analog and board information are outside the scope of the engineering information system EDM model. There is an enormous amount of information for potential inclusion in the ECAD model.

The ECAD model is based primarily on the data and operations in the process model. An EDIF domain model and a VHDL domain model were also prepared to ensure that the language concepts receive adequate representation.

Lessons Learned

Many valuable lessons have been learned during the EIS program. Some of these are summarized below:

- The best standards are simple; most of the specification changes that were implemented during Phase 2 of the EIS program were simplifications.
- Object-oriented design is an evolving art.
- Inconsistencies were revealed during the EIS prototyping effort that aided in understanding tradeoffs.
- It is feasible to implement the EIS specifications using the existing framework software.
- Significant effort remains to formally couple type systems.
- Function distribution, state distribution, and distribution within a server all require significant additional effort.
- Some form of intertool message-passing capability is of significant benefit to fully exploiting the tools that fall in the loose portion of the integration spectrum.
- The integration process in the current PREIS implementation should be automated to allow integration with little or no coding.

Section 3
EIS Prototype Development/Demonstration

The prototyping phase of the EIS program addressed the implementation of more than 200 object classes. It validated the EIS specifications and demonstrated the feasibility of modifying framework software to meet the administrative and domain-specific portions of the specification. Honeywell's internal object-oriented framework (Gaia) was extended to achieve these goals. The result, PREIS, encompassed some of the EIS object management system and UIMS functionality, as well as major portions of the EES. It also includes schema management services that are used to install portions of the ECAD domain reference schema. The PREIS was developed in three incremental builds:

- Build 1 established a platform, using basic object management services, to serve primarily as a basis for the higher level functionality of builds 2 and 3. The services include AOM support, control points and policies, and OODL execution.

- Build 2 populated the OMS with types, instances, and operations in the areas of schema management and user interface. A browser and an inspector were built, and some C interfaces appeared.

- Build 3 developed services that facilitate tool interoperability, EIS environment tailoring, and attached tool incorporation. The C interfaces to the class managers and remote procedure call (RPC) module will also be completed in this build.

The PREIS was constructed by implementing various types defined in the EIS specification on top of the existing Gaia framework. Gaia was developed at Honeywell's Systems and Research Center under IR&D funding from 1986 to 1989. The basic architecture of Gaia is shown in Figure 14.

- The communications subsystem is based on the virtual operating system (VOS) layer and the RPC layer. The VOS layer provides a portability layer that insulates Gaia from the underlying operating system. The VOS interface closely resembles POSIX. The RPC layer provides a remote procedure call capability for client use in communicating with the object manager kernel.

- The object manager kernel primarily provides services for representing and manipulating engineering data as well as for data persistence and transaction management. The kernel is based on an object-oriented model that is similar to the EIS object model.

- The object manager interface provides clients with a programmatic interface to the primitive services provided by the kernel. This interface isolates clients from details of the RPC.

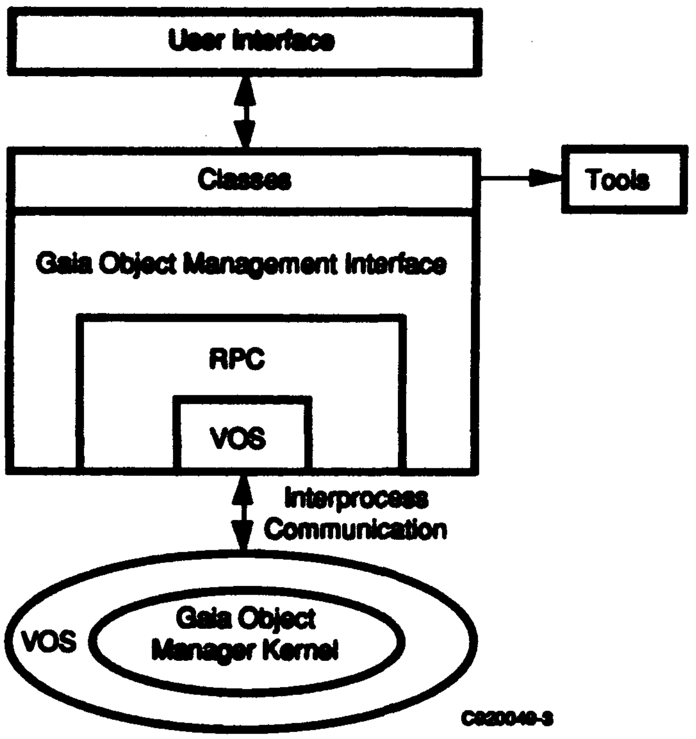

User Interface
Classes
Tools
Gaia Object Management Interface
RPC
VOS
Interprocess Communication
VOS
Gaia Object Manager Kernel
C920040-3

Figure 14. The Gaia Architecture

- In Gaia, classes function much like types in the EIS. A Gaia class represents a set of instances with like structure and behavior. Further, one class may inherit structure and behavior from other classes. Gaia provides a set of predefined classes that provide core functionality (e.g., creating instances, maintaining inverses, processing logins, applying inheritance, resolving names, checking types, and invoking operations). Given the predefined classes, one may create project-specific classes such as subclasses (much like EIS).

- Tools provide the implementations of the functional behaviors defined by classes. Tools may be implemented in terms of the functions defined by existing Gaia classes or they may be "adapters" that are used to attach preexisting tools such as commercially available editors, compilers, simulators, etc.

- The user interface (UI) serves as a window through which a user logs into Gaia and interacts with objects and tools. The UI is based on the notion of a "context," which is a container of objects (much like a traditional operating system directory that contains files). Each user has an associated default context, and after the user logs in, the UI displays the objects in the user's default context. Pull-down menus provide access to tools and allow navigation through the context space.

PREIS Architectural Overview

Figure 15 is a top-level view of the PREIS architecture. Taking this view, PREIS consists of the addition of an adapter and the PREIS types to the underlying Gaia system. The adapter serves as a wrapper around the Gaia system. Its primary purpose is to encapsulate the underlying Gaia system to yield a more EIS-like interface. The PREIS types implement the EIS services (schema management, function model, etc.). The underpinnings of this layer are provided by functionality provided by the underlying Gaia layer. For example, PREIS types are specializations of Gaia classes, PREIS functions are specializations of Gaia operations, etc.

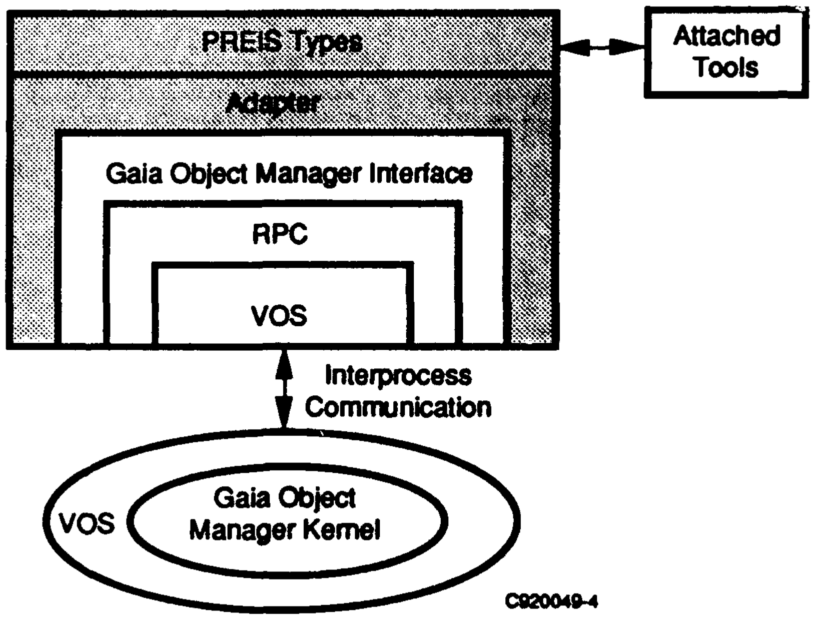

PREIS Types
Attached Tools
Adapter
Gaia Object Manager Interface
RPC
VOS
Interprocess Communication
VOS
Gaia Object Manager Kernel
C920049-4

Figure 15. Prototype EIS (PREIS) Architecture

According to the EIS specification, the Gaia object manager kernel may be loosely viewed as an attached data server. While the attachment is not complete with respect to the EIS specification, it does follow the spirit of the approach defined there.

In the PREIS, each type defines functions (zero or more); a function, in EIS terminology, corresponds to an interface. Each function may have any number of implementations. The architecture of PREIS is designed around these two concepts. Essentially, the function level provides a mapping to the implementation level. In EIS and PREIS, there are two types of functions: normal and express. A normal function is controlled by the function application mechanism; it performs name resolution, triggers control points, selects the correct implementation, and invokes it. An express function maps directly to an implementation and performs only type checking on its arguments.

The Gaia OMS provides services for representing, storing, and manipulating the engineering tools and data necessary to model a project. An object-oriented model (as opposed to relational or other models) was selected because it more directly and naturally models the objects dealt with by engineers. It also simplifies the task of the framework in presenting project objects in a uniform fashion that is easily understood by the end user.

Classes and Instances—All types of data objects known to the framework are defined as classes. Each class acts as a model of some real-world object (e.g., file, program, user). A class definition describes the attributes and relationships carried by its instances and the values those slots may hold. A class definition also describes operations that may be used to manipulate the class instances; class contains the set of its instances (objects that conform to the class definition).

Inheritance—New classes are defined in terms of existing classes (superclasses). Defining a new class in terms of a superclass means that the new class inherits characteristics (slots and operations) of its superclass. The new class is called a subclass (or derived class) and typically has additional slots and operations that distinguish it from its superclass. Reusing existing class definitions reduces the effort required to define new classes.

Built-in and Project Classes—There is a distinction between built-in classes (BCs) and project classes (PCs). BCs are predefined classes that come with the framework; they serve two purposes. First, they provide functionality for defining PCs. Project managers can define PCs that model project-specific objects and thereby create environments to meet specific project needs. Once a PC is defined, instances of the new class can be created and manipulated through the defined operations. Second, BCs capture system data used by the framework in performing its functions. For example, creating instances, maintaining integrity, processing logins, applying inheritance, resolving names, checking types, and invoking operations all require information stored in the BCs.

Object is the most general class in the system. All other classes are derived either directly or indirectly from Object. Slots common to all instances (e.g., name, unique id, etc.) are factored out and defined in the Object class.

Class is the class of all classes; that is, Class contains instances that are themselves classes (BCs and PCs). Class is even an instance of itself. This recursive definition allows the framework to store information on classes using the same capabilities used for all other instances. It also allows the framework to present an object-oriented view of the BCs to the project manager without having to implement additional functionality beyond that provided for all other instances.

OMS Data Representation—The internal representation used by the OMS is based on a single data type called entity. An entity represents an instance of a class that consists of a list of names slots. For example, an entity representing a file might have name, creation data, and text store slots. Each slot has a type and a value. For example, the name of the file might be "MEMO1," its creation data might be "7/1/87," and its text store might contain the text of a memorandum on programming guidelines.

OMS Services—Services for storing and manipulating objects are provided by four classes of operations that allow users to create, retrieve, decompose, and update objects in the OMS. The create operation establishes a new instance of a class and stores it in the OMS. The retrieve operations cover the instances of a given class. The decompose operations fetch the values of an instance's slots. Finally, the update operations change the values of the slots of an instance.

Another key feature of the OMS is its support of inverses; they provide the basis for dynamically maintained values, bidirectional relationships, and traceability. In defining a class, one defines the set of slots carried by its instances. Further, the OMS allows one slot to be defined as the inverse of another slot. Given such a definition, updating the value of the slot defined as the inverse causes the update of the inversed slot.

User Interface (UI)

The user interface serves as a window through which a user logs into the framework and interacts with the project objects and tools. The UI validates users logging onto the framework. A validated user is first displayed a set of default objects that are associated with the user. By clicking on any of these objects, the user can access a pull-down menu of tools that are applicable to that object. Another window is opened after the user selects a tool. It is within this window that the tool performs its task, directs any output, or requests additional information from the user. Additional objects may be displayed in this window; the same pattern is observed for interacting with these new objects. A user (or tool) closes a window when the work there is complete, and control returns to the "parent window" for the current window.

Tool Set Subsystem

The tool subsystem is not part of the framework proper. It results from and supports the integration of PCs. A collection of project-specific classes integrated into the framework represents an engineering environment. Thus, components of the tool set subsystem (class managers, tools, and tool interfaces) can be viewed as an environment instantiated on the framework (OMS and UI). Generators create the bulk of class managers, and in the future they will also create interface specifications for tools and tool interfaces from the CP information stored in the OMS. Although these components are not part of the framework, they are discussed here because they are necessary to the creation and support of an environment.

Class Managers (CMs)

Each class in the environment is associated with a CM that provides strongly typed, low-level access to the class instances. Essentially, each CM contains accessor functions that allow decomposition and update of a slot in an instance of the class. Thus, if there is a file class with name, creation data, and text store slots, the file CM would have the functions: get_name, get_creation_date, get_text_store, set_name, set_creation_date, and set_text_store. In addition, a CM provides a strongly typed function to create an instance of its class and to display all instances of the class.

Each tool accepts a strongly typed list of arguments and performs an operation on an instance of a class. For example, an edit tool might perform and editing operating on a file. Tools use the class manager component to access the individual slots of the instances they manipulate. A tool can be tightly integrated or can call an existing tool, passing it the appropriate parameters; tools can call other tools directly.

Tool Interface (TI)

The TI component provides an intermediate interface layer between the UI component and the tools component. Each tool has a corresponding tool interface that supports the extensibility of the environment and provides a strong typing between the tool, the UI, and the OMS. When a user wishes to invoke a tool, the UI invokes the tool interface of that tool and passes the object that it is to manipulate. The tool interface then prompts the user for any additional arguments and supplies any default values for arguments that its associated tool expects. Once it has the arguments, it performs any necessary type conversions, invokes the tool, and returns a message to the UI that contains the result objects to be displayed.

PREIS Demonstrations

Scenarios for using the EIS electronic computer-aided design EIM were designed to reflect use by integrated circuit designers. They do not reflect use of the EIM by groups such as tool builders, project managers, or design, system, and individual tool administrators, since it is assumed that these groups will access other portions of the EIS more often than they will access the ECAD engineering information model.

Scenario selection was motivated by the desire to demonstrate the utility of the ECAD engineering information model in sharing information otherwise expressed in terms of formal design languages (e.g., EDIF, VHDL) or in the specific format required by individual tools. The ECAD engineering information model is used to share information in a semantically representative fashion (in place of more semantically neutral formalisms). This approach allows application of additional administrative functions (e.g., version control, configuration management) to design data. It also allows the contents of the ECAD engineering information model to be more readily communicated to those familiar with design concepts but not familiar with specific expression of the concepts in an uninterpreted form.

Areas of the ECAD engineering information model addressed by the scenarios include:

- Hierarchical representations, cell instance hierarchy, geometric shapes;
- Use of the occurrence tree and its relationship to layout information;
- Technologies, technology rules.

Use of the ECAD engineering information model as an intermediate representation for exchanging information between EDIF and VHDL design information is reflected. As a result, some scenarios represent various combinations of the same or related design information expressed in terms of EDIF and VHDL in the same scenario.

Scenario Topics

Load design information from an EDIF schematic editor into the ECAD engineering information model. Retrieve netlist information from the ECAD engineering information model expressed in EDIF. This scenario addresses hierarchical representation and cell instance portions of the ECAD engineering information modeland geometric shapes associated with schematic representation. It demonstrates the capability of the ECAD engineering information modelto support a unified view of IC design information, allowing tools to exchange information shared between two views of a design (e.g., netlist and schematic).

Load information from an EDIF structure description into the ECAD engineering information model. Retrieve netlist information from the ECAD engineering information model and express it in VHDL. This scenario uses the instance hierarchy and functional models of cells stored in the ECAD engineering information model. It demonstrates using ECAD engineering information model to exchange information between two representations (EDIF and VHDL) widely used in the IC design community.

Retrieve hierarchical netlist stored in the ECAD engineering information moel and flatten it using an EDIF netlist expander. Load the flattened netlist description into the ECAD engineering information model. This demonstrates the ability of the ECAD engineering information modelto hold both hierarchical and flat design structures. It also provides a bridge between structural and layout design.

Retrieve the flattened netlist from the ECAD engineering information model and create the layout design using a tool with standard cell layout capability. Load the layout information expressed in EDIF into the ECAD engineering information model. This scenario addresses cell occurrence portions of the ECAD engineering information model, along with geometric shapes associated with layout representation. It also demonstrates the capability of ECAD engineering information modelto support the EDIF mask layout view.

Retrieve technology parameters and rules stored in the ECAD engineering information model and load them into the standard cell layout design tool that uses a declarative representation of rules. This scenario illustrates downloading of two rule sets into the tool for creating two version of a design (one for each technology). It also demonstrates the ability of the ECAD EIM to store information relating to design technologies and rules.

Extract parameters (e.g., net and component delay) from the layout design and update attributes of the netlist stored in the ECAD engineering information model. Retrieve this netlist and perform a VHDL simulation to get a more accurate picture of design behavior. This scenario demonstrates the process of back-annotation, which is critical to any IC design activity.

Section 4
EIS Application Demonstration

During the past 10 years, the DoD has experienced a dramatic increase in software spending due to the increase in software system size, complexity, and the critical nature of modern software systems. While the cost of software development and maintenance has risen, productivity has fallen behind the demand for new software. The same trends are perceivable throughout the software, in private businesses, and in government agencies alike. It is estimated that of all the software code written in 1983, probably less than 15% is unique, novel, and specific to individual applications. It is also estimated that, on the average, only about 5% of code is reused; thus, the reuse of existing software products may replace all or part of the remaining redundant 80% of software development.

Industry is also plagued with significant cost and schedule overruns in the hardware domain. Reuse of low-level components has long been an integral part of hardware development. The time and costs are overrun, however, when fundamental top-level design incompatibilities are discovered at a late stage in the system development life cycle. Part of the hardware system must then be redesigned and reimplemented. Therefore, it is essential that high-quality, proven, top-level hardware life-cycle components (e.g., specifications, requirements, designs) be reused along with the current standardized implementation-level components.

The EIS application demonstration (AD) program developed a reuse environment that integrated reuse methodologies into the system (hardware and software) development life cycle. It supports the necessary mechanism to make reuse a natural part of system development. The approach evolves around the incorporation of the domain analysis process and results in the system development methodology and reuse environment design. During the domain analysis process, the knowledge and experiences of domain experts are captured in a domain model. This generic model identifies the characteristics and functionality of multiple systems in this domain. A component classification mechanism is established that, in combination with the domain analysis data dictionary, can be used to define the reuse repository classification scheme.

Reuse Methodology Development

The EIS representative hardware and software development life-cycle processes were derived from interactions with government and industry hardware and software experts. The DoD 2167A software artifacts (requirements, designs, code, test cases, documentation) are mandated by the DoD as the format for all deliverable software; the VHDL, IEEE-1076, is emerging as the industry standard for digital hardware development; and the DoD has mandated that all deliverable ASIC digital hardware designs be documented during VHDL. Since government contractors are bound by the DoD-mandated hardware and software standards, the DoD 2167A software development life cycle and the VHDL hardware module development cycle are the software and hardware development processes to be represented and supported in our virtual system space.

In the course of development of the integrated hardware and software reuse methodology, existing literature was studied, there was interaction with hardware and software development experts, and there was participation in the JIAWG Reuse Committee meetings. From these government and industry interactions, the following key issues were identified:

- Reusability must be transparently integrated with all phases of the system development process to maximize the potential benefits of reuse.
- Connections between specification, design, implementations, and test levels must be maintained to enable a design that is reused to easily lead to implementation and test case reuse.
- Careful analysis of the application domain(s) must be performed to identify the basic objects and operations within the domain and to provide a generic domain model for applications within the domain(s).
- Reusable artifacts must be organized by the requirements they satisfy to reduce the probability of retrieving nonrelevant artifacts.
- System specifications and requirements must be defined in terms of the domain model and a set of modeling artifacts identified during the domain analysis process.
- Future reusability via abstraction, encapsulation, and documentation must be implemented in the design.

System Development Process

The objective of this effort is to encourage and support reuse from the earliest stages in the development life cycle to the final stage. When system specifications and requirements are reused to partially define a new system, the potential of reusing detailed design specifications, design requirements, implementations, and other related artifacts is significantly increased.

As shown in Figure 16, our reuse methodology has incorporated the JIAWG-defined general system development life-cycle stages.

The system development life cycle begins with the system specification, from which various requirements are derived and allocated to the top-level software and hardware components that comprise the system. The allocated system requirements are then used to derive requirements for their subcomponents. For example, at the system requirements analysis stage, the system analyst defines the system-segment specifications and a set of hardware and software requirements from the statement of work and/or the request for proposal and the possible user specifications. These system specifications and requirements are analyzed in the hardware and software requirements analysis stages by the hardware and software requirements analyses, respectively. They derive and refine the software and hardware engineering interface requirements and define the hardware and software component tests. This process of requirements allocation and refinement is repeated at each stage of the system development life cycle.

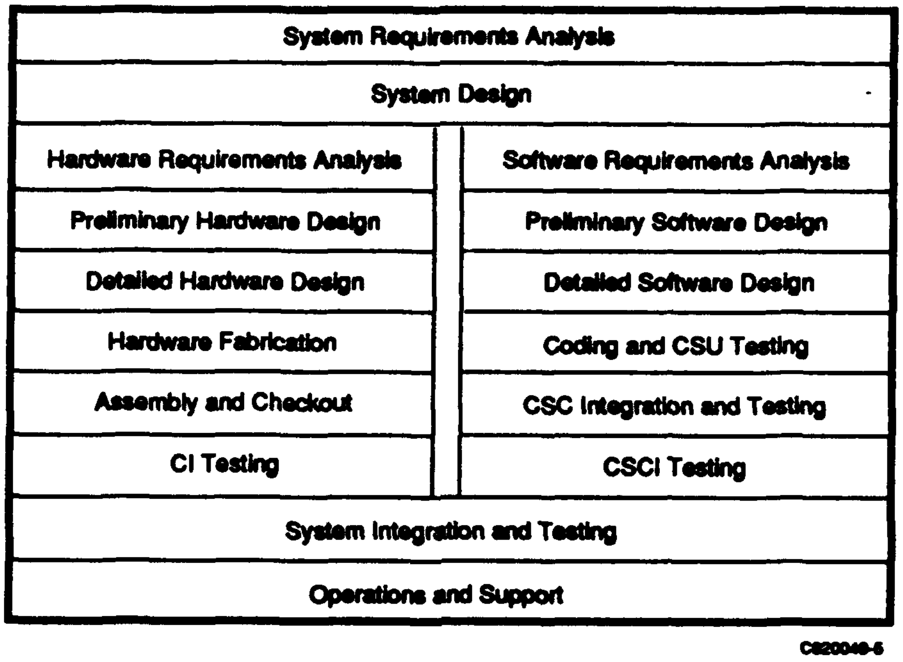

System Requirements Analysis
System Design
Hardware Requirements Analysis
Software Requirements Analysis
Preliminary Hardware Design
Preliminary Software Design
Detailed Hardware Design
Detailed Software Design
Hardware Fabrication
Coding and CSU Testing
Assembly and Checkout
CSC Integration and Testing
CI Testing
CSCI Testing
System Integration and Testing
Operations and Support
C920049-6

Figure 16. General System Development Process

At the design stage of the system development process, engineers develop artifacts that document, test, and implement the requirements of the various parts of the system. To support reuse at every level of the system development process, it is essential to organize the system according to the system development stage in which the artifacts were developed. This organization of artifacts, in combination with other reusable artifact classification criteria, will facilitate the identification of relevant reusable artifacts during the development of a new system.

The probability of artifact reuse can also be increased by establishing and maintaining connections between the system requirements and their derived hardware and software component requirements, documentation, test cases, and implementations. Subsequently, if a subset of an existing system can be reused in the development of a new system, it is very likely that hardware and/or software component requirements and their implementations will partially define the hardware and software component requirements of the new system. Maintaining these connections will facilitate the location of these reusable artifacts.

Identification and analysis of the system development process is only the first step in defining the EIS reuse methodology. Domain analysis is a key factor in the success of a reuse environment. The domain analysis step provides the building blocks for the EIS reuse methodology.

Domain Analysis

Domain analysis is the process of identifying the characteristics, system requirements, and functional classification of the artifacts within a specific domain. These domain artifacts are generalized into a domain model that defines the objects and operations of the domain, along with their relationships. The domain model is generic; it covers many specific applications within that domain. The domain model artifacts can be instantiated and tailored to define new systems within the domain. It is not only necessary to incorporate this process into the reuse methodology, but also to provide the environment support to store and manipulate the domain artifacts.

Robert Holibaugh, in his "JIAWG Domain Analysis Method" document, identified the following domain analysis outputs:

- Knowledge base and definition,
- Terminology and classification information,
- Domain model.

The knowledge base includes information on the domain and its major parts, the abstractions and features, and the relationships between them and other domain elements.

Terminology defines the terms for domain-related concepts, abstractions, services, and constraints to allow the reuser to understand the results of the domain analysis. The domain analysis process also provides classification information necessary to properly place the domain artifacts in a reuse repository. The domain model consists of scenarios of top-level services, subject and structure diagrams of objects and their relationships, and class and object specifications.

Life-cycle-specific domain requirements can be derived from the domain model. They can be used to guide the definition of the engineering development methodology. The domain requirements model allows the development engineers to define their general contract requirements in terms of the life-cycle-specific domain requirements, as shown in Figure 17. Through this process, unstructured contract requirements become life-cycle-specific contract requirements that can be used to define the engineering development methodology by identifying the requirements specific to every life-cycle stage in the development process.

Many variations/options and capabilities of the system(s) within the domain are captured in the domain mode. Definition of the domain model is an iterative process that refines and enhances the domain knowledge. The designer of a new system who is using reusable domain artifacts will encounter unexpected problems and gain insight into the solution of these problems. New reusable artifacts and other reuse development options come from new solutions. The domain knowledge base and the domain model must evolve with advancements in technology to be assured of the availability of the most recent reusable artifacts.

The development of a reuse methodology will require artifacts that are expected to be potentially reusable during requirements analysis to be designed, developed, tested, and delivered with reuse in mind. This can be accomplished by defining the system artifacts in terms of the artifacts defined in the domain model. When development of a new system is based on the artifacts of an existing domain model, the likelihood is increased that new system requirements, designs, and other related artifacts have been addressed in previous projects that built on the same domain model. Thus, reuse is changed from an ad hoc process to a more structured, well-defined process.

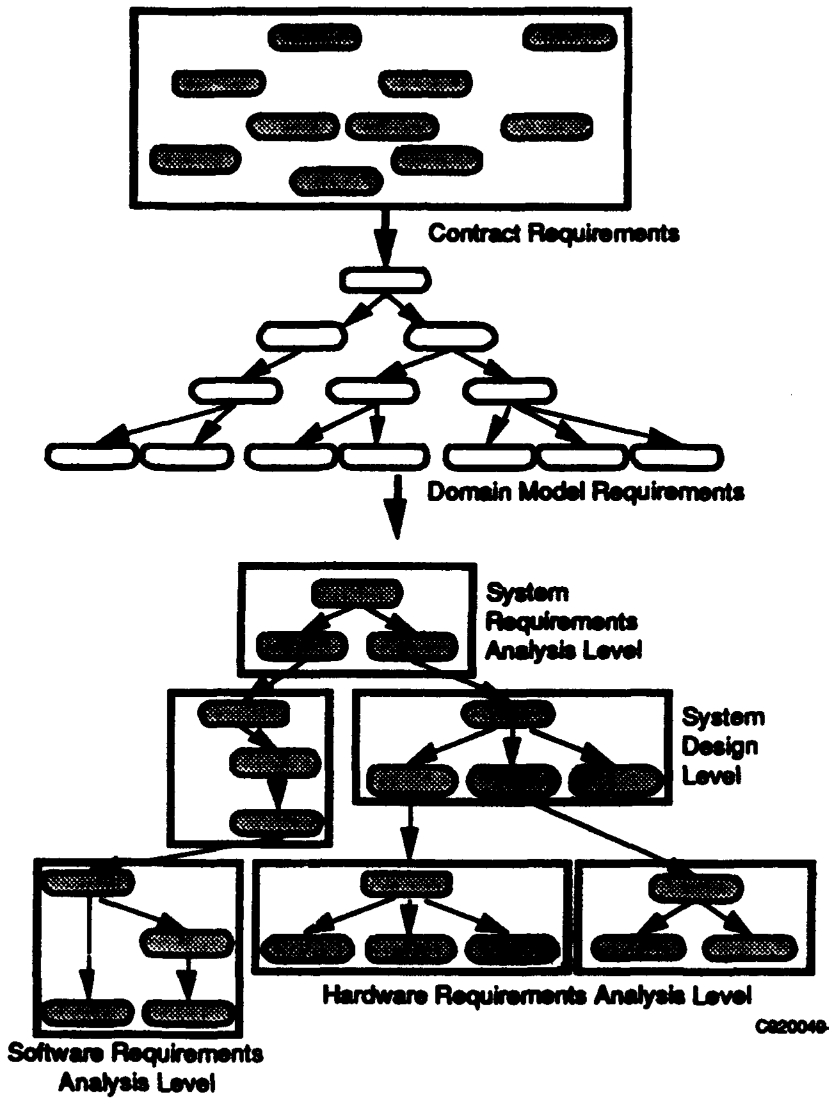

Contract Requirements
Domain Model Requirements
System Requirements Analysis Level
System Design Level
Hardware Requirements Analysis Level
C920049-6
Software Requirements Analysis Level

Figure 17. Definition of Contract Requirements in Terms of Domain Model Requirements

Reuse Environment Requirements

To encourage reuse during all stages of the system development life cycle, the environment must be integrated into the development methodology so that reuse becomes a natural, easy mode of development.

In analyzing the environment needs of several software and hardware development processes used in various Honeywell projects, and by studying current literature on existing software repositories and reuse environments, the following general reusability requirements for system development environments were isolated:

- Storage, control, and configuration management of requirements, designs, implementations, documentation, test cases, and their history is absolutely necessary.
- Development of domain-specific classification schemes for reusable system artifacts must be supported.
- A wide variety of information structures must be represented, since the reuse library may contain a wide variety of hardware and software artifacts.
- Connections between requirements, designs, test cases, documentation, and other related system artifacts must be represented.
- Abstract, generic information structures must be represented.
- Artifact transformation operations from reusable abstractions to concrete implementations must be supported.
- Artifact composition operations must be supported.

Since one of the essential problems in reusing software artifacts is locating and retrieving them from a large collection of reusable artifacts, information overload must be prevented by providing the engineer a development life-cycle-specific reuse catalog.

A fully developed environment management system will provide management and control of system development processes. The EIS supports these reuse environment requirements. It includes system life-cycle support for requirements tracking and validation, controlled access to tools, help facilities and automated dependency checks, and notifications of mandatory system artifacts. The EIS is a site-tailorable environment management system that addresses the varying reuse methodology support requirements across engineering projects, system architectures, and system domains. The EIS can capture hardware and software domain models by representing the generic domain models in an object-oriented type hierarchy, which can be instantiated and tailored to define new systems of subsystems within the domain. To support the evolution of the knowledge base, the EIS framework provides a strong change and configuration model.

To save the system engineer from the ordeal of searching a large collection of reusable system artifacts for the reusable artifacts that are relevant to a given system development life-cycle stage, a life-cycle-specific reuse catalog can be built on the EIS view mechanism. In the EIS, a user is represented by an entity, with a name and a password that is authorized for access to portions of the EIS. Users are members of one or more user groups. Authorization is based on the user groups associated view(s). A view defines the objects that are visible and accessible to those users who are included in this view. A user can change the view of changing user groups. The view mechanism currently used for access control can be used to build a user-group-specific reuse catalog.

When designing a new system using reusable system artifacts, the EIS framework will determine the "view" of the user and display the reusable artifact classes that are accessible to that view. For example, in the "JIAWG Software Engineering Environment Specification" document, the various user roles within the software development life cycle include the system analyst, The software requirements developer and analyst, and software designer, the programmer and CSU tester, the CSC integrator and tester, and the CSCI integrator and tester. Because each user is responsible for a certain portion of the system development life cycle, those reusable artifacts can be assigned classes relevant to a particular life-cycle stage to the "view" of the user group responsible for system development within that stage. Thus, the user who is a member of the systems analysis user group will be responsible for formulating the system requirements, formulating the system design, allocating the system components, and identifying the system-level tests. Since the system analyst is not responsible for implementing the hardware and software components of the system, there will be no presentation of reusable hardware and software implementation classes. Instead, only those reusable artifact classes relevant to a system analyst will be visible to this user. This will significantly reduce the reusable artifact classes requiring user search.

Potential Application Demonstrations

The EIS concept covers both domain-specific information modeling (electronic, mechanical, and software engineering) and domain-independent framework services (versioning, configuration management, object management, etc.), as shown in Figure 18. The objectives of the potential demonstrations discussed in this section are to apply the generic EIS framework technology to various applications and integrate the EIS information model with other domain-specific information models.

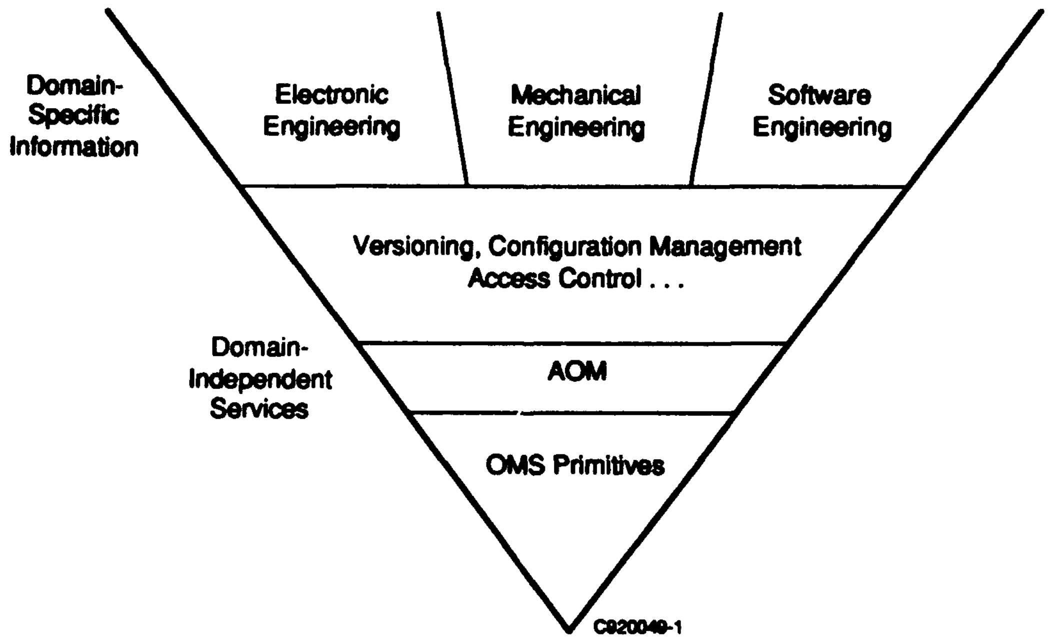


Figure 18. Scope of the EIS

The potential demonstration examples presented here illustrate how the various engineering domains can be integrated and how the variety of problem areas associated with each domain can be resolved. The EIS application areas and engineering domains are shown in Figure 19; the areas for demonstration are highlighted. Potential demonstrations include application and demonstration for the following environments:

- Electronic Computer-Aided Design (ECAD);
- Microelectronics Manufacturing Systems Technology (MMST);
- Product Data Exchange Standard (PDES) testbed;
- Microwave/Millimeter-Wave Monolithic Integrated Circuit (MIMIC);
- Advanced Tactical Fighter/Joint Industry Avionics Working Group (ATF/JIAWG).

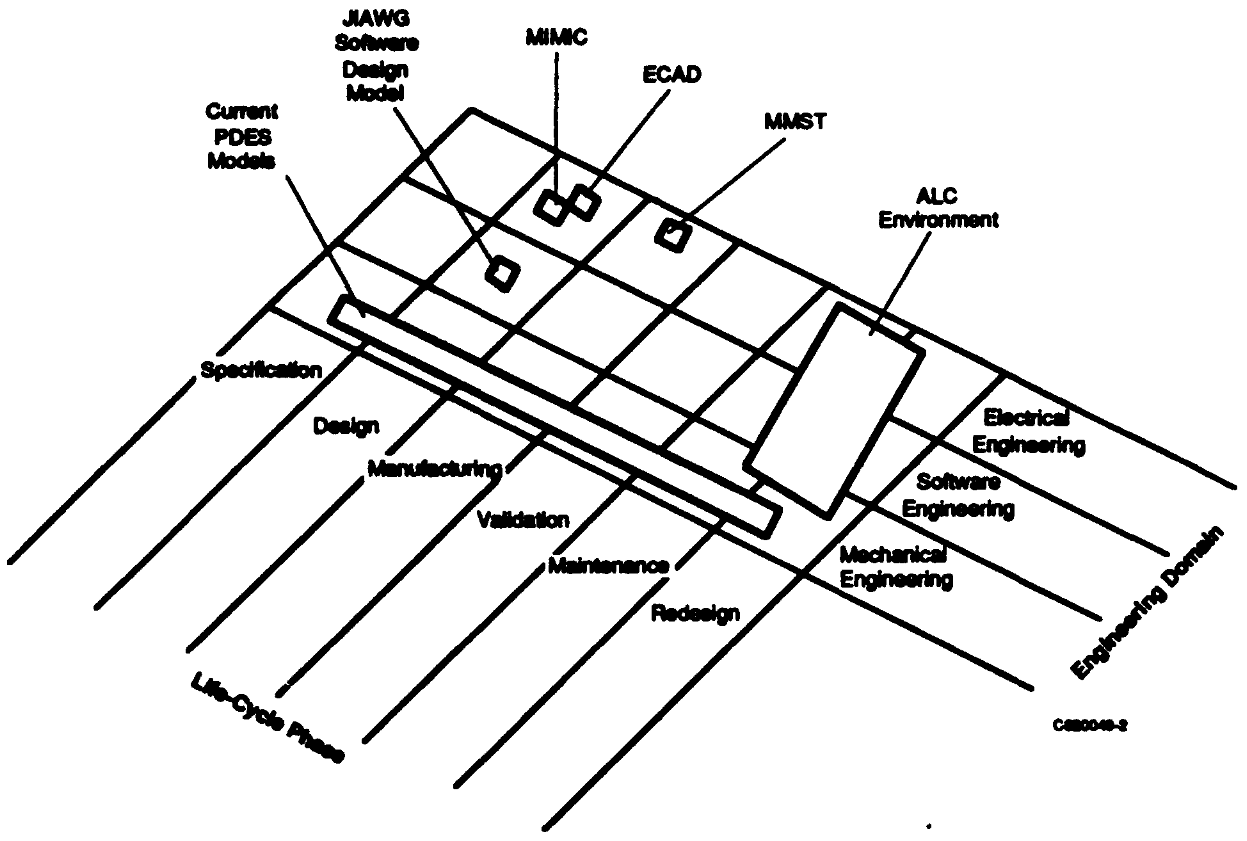


Figure 19. EIS Application to Engineering Domains and Life-Cycle Phases

In addition to the demonstration scenarios mentioned above, the Air Logistic Center (ALC) environment is examined (see Figure 19); however, it is not discussed in detail.

The EIS technologies include a framework technology and an information model technology. The framework technology is domain independent and deals with tool interoperability and information management. The information modeling technology is domain specific. The EIS program developed the ECAD model; many other ongoing activities deal with other domains. The EIS framework technology is designed to accommodate all of the information models to result in engineering environments that can span multiple domains and life-cycle phases. The EIS can be adapted to the needs of many different applications. Most adaptation is done at the information modeling level; however, the EIS framework can be modified, if desired, to meet the site-specific needs.

EIS Application and Demonstration for Electronic System Design

Several scenarios have been identified to use the EIS electronic computer-aided design model. They are designed to reflect the integrated circuit designers user group. Assuming that groups such as tool builders, design administrators, project managers, system administrators, and individual tool administrators will access other portions of the EIS more often than they will the ECAD model, they are not specifically reflected in the scenarios.

Areas of the ECAD model addressed by the scenarios include hierarchical representations, cell instance hierarchy, and geometric shapes; use of the occurrence tree and its relationship to layout information; and technologies and technology rules.

The selected scenarios would demonstrate the utility of the ECAD model in sharing information that is otherwise expressed in terms of formal design languages (e.g., EDIF and VHDL) or in the format required by individual tools. They are intended to demonstrate use of the semantically representative ECAD model to share information instead of formalisms that are more semantically neutral. Major advantages to this approach are that additional administrative functions could be applied to the design data (e.g., version control and configuration management) and that the ECAD model contents could be more readily communicated to those familiar with design concepts but not with the expression of the concepts in an uninterpreted form.

The use of the ECAD model as an intermediate representation for exchanging information between EDIF and VHDL design information would also be reflected in the scenarios. Some represent various combinations of manipulating the same or related design information expressed both in terms of EDIF and VHDL in the same scenario. The electronic system design and build activities are highlighted in Figure 20.

These scenarios would demonstrate the benefits of EIS to contractors and the DoD and would illustrate the viability of the EIS standards. As shown in Figure 21, the ECAD model would demonstrate multiple interrelated scenarios (e.g., customer, prime contractor, and subcontractor roles) and would illustrate that the EIS standards are key to the demonstration. The scenarios would demonstrate an IC design process from the VHDL RTL description to manufacturing data, using both proprietary and commercial CAD tools, and would illustrate the design interface and data exchange among various contractors. Finally, the model would generate the data items necessary to demonstrate the use of EIS standards for data interchange in the chip design environment.

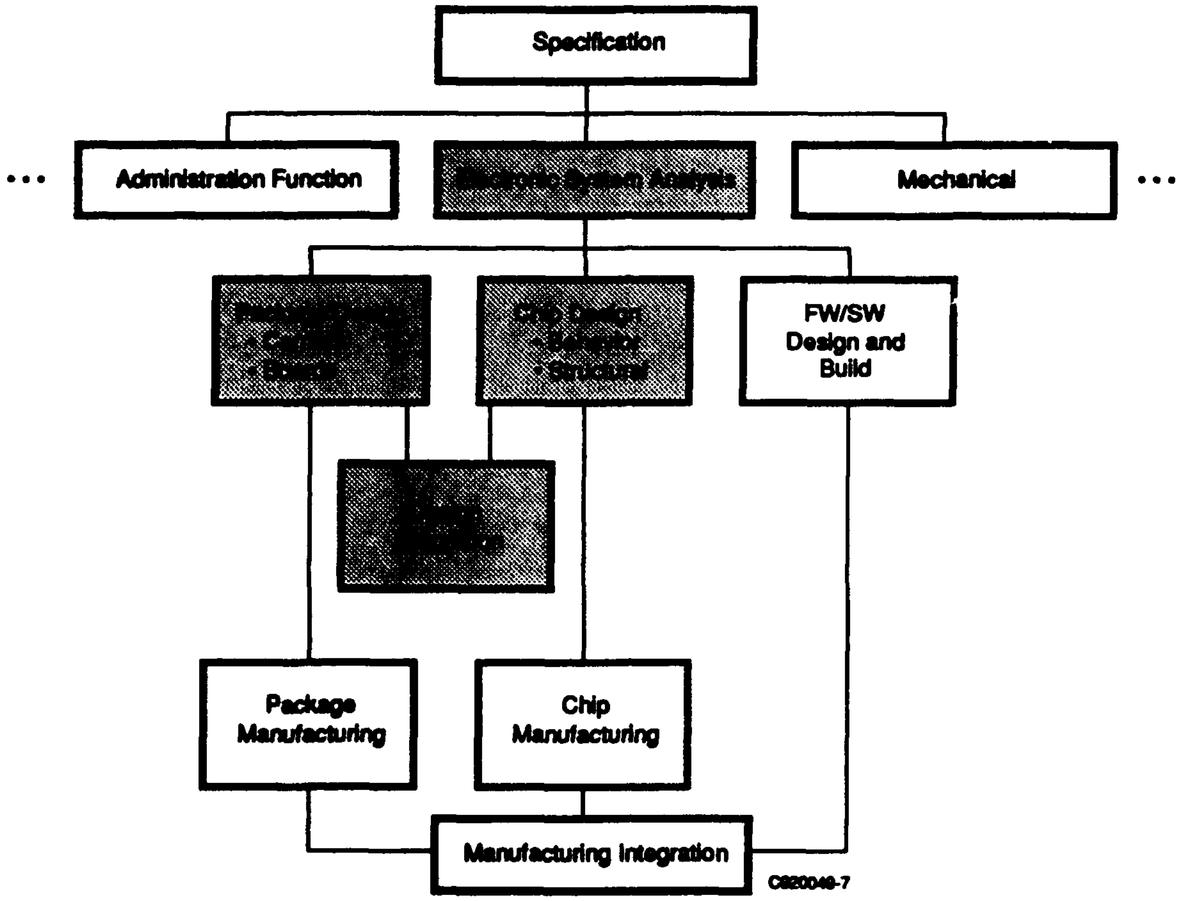


Figure 20. Electronic System Design and Build Activities

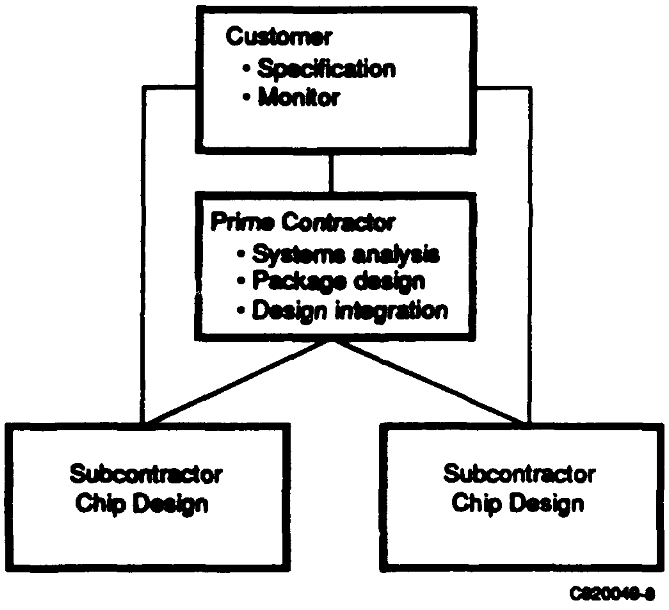


Figure 21. Electronic System Design Demonstration Scenario

EIS A/D for Microelectronics Manufacturing Systems Technology (MMST)

Integration of the ECAD model with the MMST models, and use of the EIS framework to support the MMST central information processor, is a possible demonstration scenario. Technologies of the EIS and MMST could be integrated to handle the design-to-manufacture transition of integrated circuits. This potential integrated design-to-manufacture demonstration environment is depicted in Figure 22.

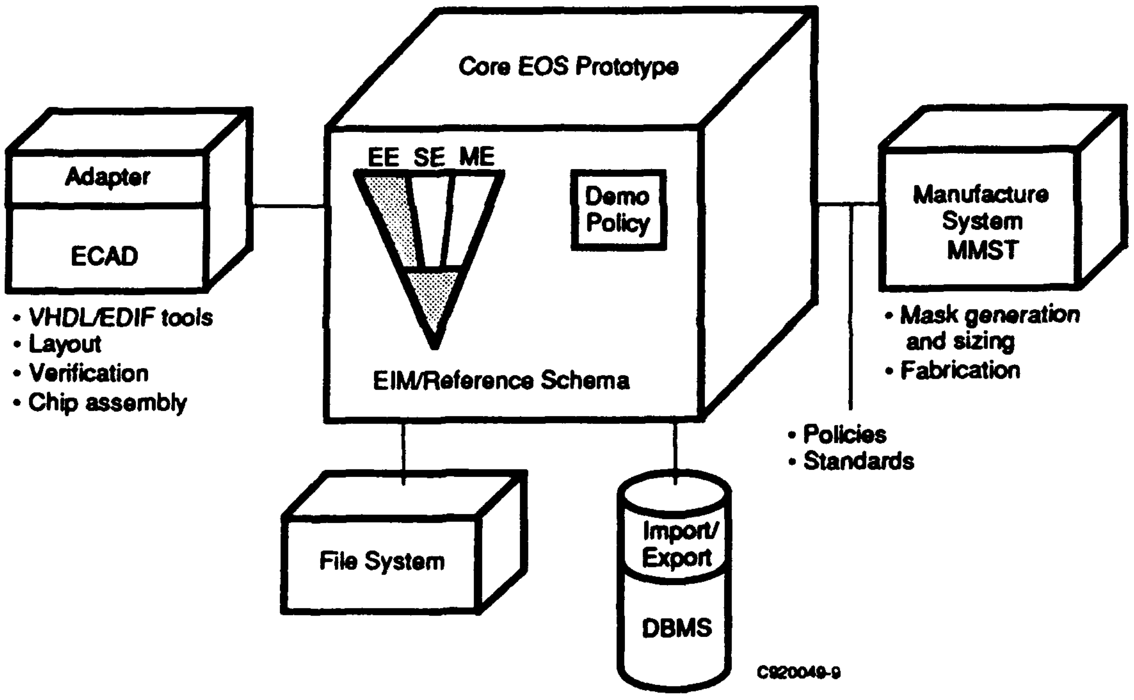


Figure 22. Potential Design-to-Manufacture Demonstration

The design-to-manufacture interface would include policies and standards. Policies refer to interface mechanisms that are to be established; standards relate to interface communication mechanisms. Verification approval would be included in the policies. The data format standards selected would be site-specific and would include standards for layout description, circuits, netlists, test vectors, and technology data standards for geometric design rules and electrical models.

In the integrated design-to-manufacture demonstration, ECAD tools and the manufacturing system would be attached to the EIS. The ECAD model and reference schema for the manufacturing domain and for domain integration would then be extended, and the extended ECAD model and reference schema would be validated.

The demonstration would illustrate a common interface mechanism between diverse design and manufacturing organizations, and ultimately would facilitate the development of candidate standards for the interface. The activities for this scenario are highlighted in Figure 23.

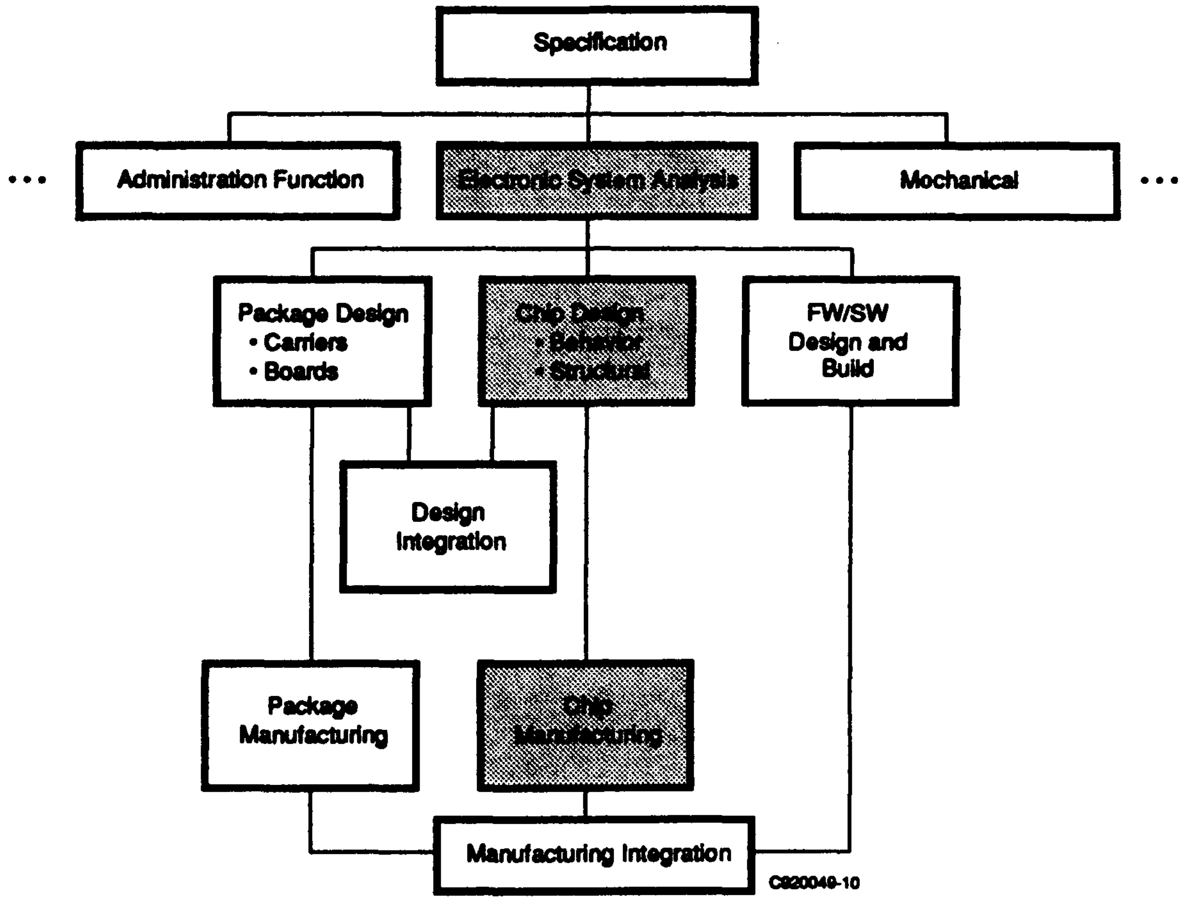


Figure 23. Design-to-Manufacture Design and Build Activities

Application and Demonstration for Product Data Exchange Standard (PDES)

A potential scenario would be insertion of the EIS into the PDES National Testbed at the National Institute of Standards and Technology (NIST). It would include PDES and engineering information system ECAD integration.

In the short term, EIS would be part of the PDES Network Testbed Headquarters at NIST and part of the Network Testbed at PDES member sites. The "centralized testbed" at NIST collects and controls the schema, data, and tools; performs verification and validation; and assists in use of the testbed. The EIS will facilitate and support these activities on the network.

In the long term, each site would become a user, contributor, and manager. The testbed becomes distributed, with each site collecting and controlling the schema, data, and tools; performing verification and validation, and assisting in use of the testbed. Propriety agreements would be enforced through access control. The EIS directly supports the exchange of schema fragments, and supports automatic notification between sites.

The PDES models represent a relatively static view of engineering environments that contain many tools (simulators, translators, etc.). The framework of the EIS addresses the behavioral side of this environment. In addition, PDES models represent agreement on information representation, which results in reducing the cost of information exchange. The EIS framework reduces the cost of integrating the tools that operate on that information. These scenarios would illustrate that the EIS can provide and augment a homogeneous environment for PDES testbed users, specifically in the areas of tool management, configuration management, and homogeneous user interface, and would facilitate integration of the PDES/ECAD models. The EIS prototype in the PDES testbed could lead to benefits for PDES development and validation in the areas of tool interoperability, data exchange at the context-driven information model (CIM) level, uniform application interface, and platform independence.

The hardware/software base at the NIST site testbed includes VAX/VMS, Sun/UNIX, IBM/PB-RT, TCP/IP, and Oracle DB. The tools include the Express Parser, Model Editor, and SQL Adapter. PDES models would be validated by loading them and testing them with query scripts that are developed through expert interviews.

Once the EIS is integrated with the testbed, there are several potential scenarios that can demonstrate the application of the EIS technology to the PDES environment. PDES Inc. is currently developing a programmer's application interface (PAI) to PDES. With the synergy between the PAI and the language bindings of the EIS framework specifications, it would be possible to use the EIS specifications as input to the PAI activity and use the EIS prototype to aid in development and validation of the PAI.

A second possibility is the potential for joint efforts to use the EIS prototype for CDIM development and validation environments and to enhance EXPRESS with object-oriented capabilities.

Since the EIS information models for ECAD, administrative domain model, and EES complement the PDES models, model integration is a third possible demonstration scenario. Both the EIS information models and the PDES use EXPRESS as the implementation language.

A fourth possible scenario can be found in using the EIS prototype to develop PDES prototypes for the electrical domain. The EIS team could aid PDES Inc. with evaluation of OODBMS.

The potential integrated EIS/PDES testbed demonstration environment for a specific instance is depicted in Figure 24.

Application and Demonstration for MIMIC

In the potential scenario, the EIS framework and ECAD model would be applied to MIMIC. ECAD applies to digital design; this scenario would create another ECAD-like analog model for MIMIC. The ECAD model for digital design and the ECAD-like analog model would then be integrated and placed on the EIS framework. The digital and analog design tools would then be attached to the EIS to demonstrate the multimode simulation capability.

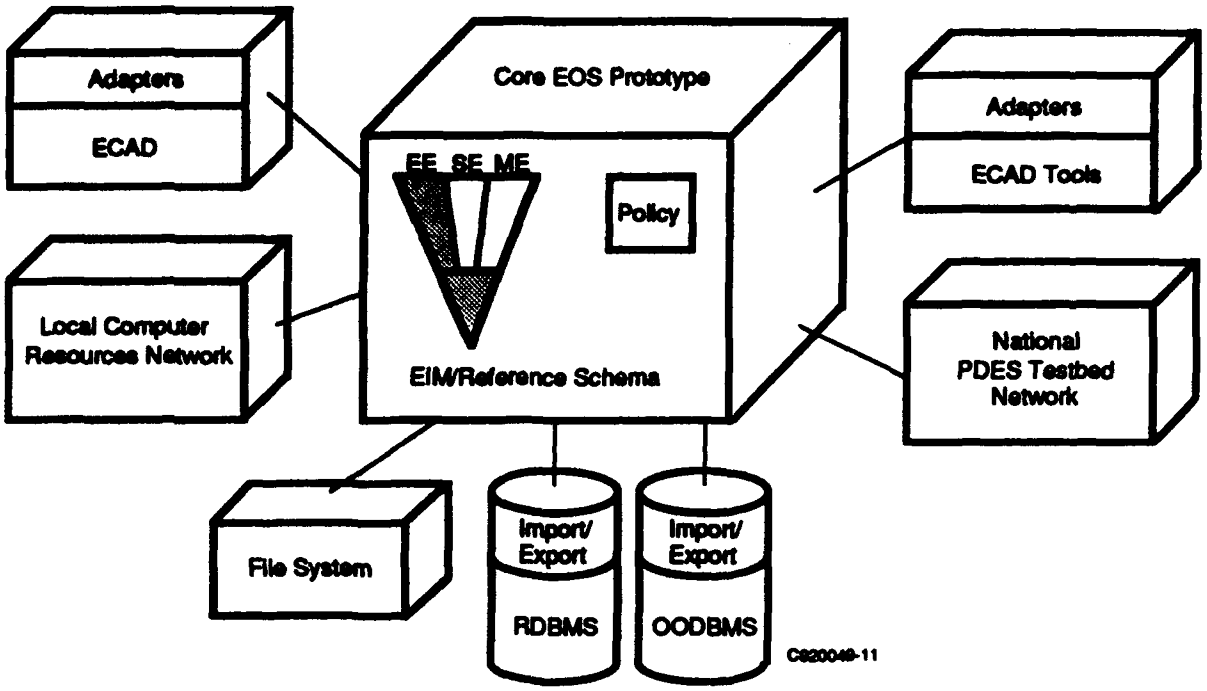


Figure 24. Specific Instance of Potential PDES Testbed Demonstration3

There is currently a desire, both within the DoD and industry, to automatically mix the digital and analog circuit designs. This scenario will demonstrate that the EIS will facilitate automation of mixed-mode design and simulation.

Application and Demonstration for[ATF]JIAWG

The[ATF]JIAWG has expressed a desire to perform concept demonstrations. It is their belief that significant pecuniary savings can be achieved through demonstrations on the actual build of[ATF] avionics systems by prime contractors. Some issues that stimulate this interest are:

- The ongoing Demonstration of Avionics Module Exchangeability via Simulation (DAMES) project has been shown to be highly successful in the concept demonstration.
- The Software Engineering Support Environment (SEE) laboratory is currently being developed; the Hardware Engineering Support Environment (HESE) laboratory is in the concept stage.
- Software/Hardware Integration Methodology (SHIM) is being developed.
- The EIS has technologies (framework/model) that can be applied to HESE and SHIM needs.

- Development of programs involving multidomain (electrical, software, and mechanical engineering) and multi-life-cycle phases (design, manufacturing, and support). The generic framework of the EIS can apply to all of these domains. The EIS information model can be extended to each domain, or can be integrated with information models that already exist in other domains.

- It is essential that the ATF programs convince prime contractors to reuse hardware and software modules across multiple programs (ATF, ATA, LHX, etc.) to enable the DoD to achieve very substantial benefits from modularity and reuse. The EIS program can aid this effort with a concept demonstration of reusability.

There are four primary objectives for the EIS application and demonstration to ATF JIAWG environment:

- Connect the EIS to DAMES.

- Use the EIS for the HESE concept demonstration.

- Use the EIS for the SEE/HESE integrated concept demonstration.

- Demonstrate the reusability of hardware and software modules through a methodology demonstration on the EIS.

Section 5
Future Directions for EIS Technologies and Their Applications

The EIS program achieved tremendous success for the DoD by catalyzing the defense and commercial communities (vendors and users) to adopt the framework and information model technologies. The EIS Specifications (developed in Phase 1 of this program) are a benchmark that can be used to evaluate commercial framework implementations. The PREIS software (developed in Phase 2 of this program) proved the viability of implementing the different facets of the EIS specifications.

EIS Technologies have potential to significantly improve the productivity of engineering users in many of the DoD/Air Force organizations such as Air Logistics Centers. There are a number of enabling technology activities that, if carried out, will enable the DoD/Air Force to maximize the benefits of such applications. These enabling technology activities are discussed in this section as recommendations to the DoD/Air Force for incorporation in appropriate future programs.

Data Storage Integration

One of the primary goals of a framework such as EIS is the integration of both tools and external datastores (e.g., filesystems, databases, etc.). In this way, frameworks provide flexibility in the design and implementation of environments by allowing an organization to select which type of datastores they wish to use. Further, integrating datastores provides a means of preserving "legacy" systems to which an organization may have committed enormous resources. A general-purpose method for integrating datastores must be defined. In particular, the notions of datastore access and transaction management across heterogeneous datastores must be addressed.

Data access and migration

EIS identified two basic approaches for hooking up to legacy databases: static and dynamic. The static approach takes advantage of detailed knowledge of the legacy database to construct an adaptor. The adaptor is hand-crafted, in a sense, to work explicitly with the structure and, to some degree, the content of the data as it appears in the legacy database. For example, adaptor code might embed code that issues a predetermined query to retrieve the "modificationDate" slot (a field) for a given "PWBDesign" (a row in a table).

The dynamic approach uses a layer of schema translation software to map the EIS object model into the data model of the legacy database. For example, dataslots are mapped onto columns, OIDs are mapped onto keys, types mapped onto tables. The translation layer then converts EIS data access requests into the appropriate query, and converts the results back into EIS objects.

While the static approach is fairly straightforward, it has its obvious limitations; it imposes significant maintenance costs if the structure of the legacy database changes often. The dynamic approach offers greater flexibility, at the cost of speed. Research into optimization and automatic code generation (permitting a combination of the two approaches, perhaps) is needed to clarify the trade-offs and offer specific implementation details.

Global transaction management

Most of today's database applications assume a centralized server for supplying and updating data. Transaction processing is effectively serialized by such a server, sometimes locking out other applications to avoid update conflict. Many schemes have been developed for enforcing transaction *atomicity* for a single database. Some research has also explored how to achieve transaction atomicity in a distributed database, but most approaches require specific functionality to be present at each data server node.

Research is needed into alternative transaction protocols. For example, strict atomicity may not be required for some long-duration engineering transactions. Approximate forms of roll-back may be sufficient for many situations. Methods for augmenting a server's native transaction processing need to be found. Further work in the area of cooperating transactions is also directly applicable to Engineering Information Systems.

Tool Integration Services

In a typical EIS, toolsets will be in an almost constant state of flux. New tools will be brought in and old ones retired (albeit at a slower rate). Each tool may introduce one or more new types of data. As tools become more open and programmable, ways of quickly constructing "super tools"—compositions of smaller tools—are needed to take full advantage of this flexibility. At even higher level, engineering process enactment comes into the picture.

Tool Registration Services

A set of services should be developed that would allow rapid and efficient integration of existing and new tools into EIS. These services would consist of an information model that would be used to describe the tools and an interface that would make it easy for a system administrator to add tools to the framework. Such tools would need to be tightly integrated with the type-definition, so that the registered tools can be closely associated with the kinds of objects/files they consume and produce.

The EIS specification currently contains some rudimentary services in this area. Higher level services are needed to make the framework interface more practical, as well as (standardized) user interface to make the services accessible to the end-user (for registering personal tools).

In addition to the typical monolithic tool, new types of tools are emerging that must be accommodated by EIS. These are the so-called "framework" tools, such as Cadre's *teamwork*™, FrameMaker™, Software-Through-Pictures™, etc. These tools are not used in a conventional invoke/terminate fashion; rather they are started in conjunction with a central server that coordinates requests and controls license tokens. EIS will need services for dealing with these types of tools.

Tool Composition and Engineering Process Support

The original EIS requirements document called out the notion of Automated Program Networks. These requirements overlapped substantially with those for the more basic control point and policy mechanisms, so the latter received more emphasis. More research is still needed on higher levels of services for process control and assistance. Supertools are, in a sense, encoded fragments of an engineering process, so the services will fall on a spectrum going from control points and policies to enterprise-wide functions. Some standardized user interfaces that enable the end-user to perform some custom tool composition are also needed.

Change and Configuration Management

A principal concern in engineering support technology is the management of change to a multifaceted product model. The objective of such changes is to focus a blurry representation into an optimum solution. The changes come from a multi-disciplinary team of designers, analysts and reviewers. The changes not only increase in frequency with the age of the project, but they are chaotic in nature; small decisions or adjustments can trigger a cascade of recomputations, constraint violations, cost impacts, etc., all involving other members of the team. Engineering support research is aimed at increasing and improving these input signals in order to lower the cost and raise the quality of a product throughout its lifecycle.

A critical means for achieving this is a comprehensive model of *change* coupled closely with the models of product, process, organization and resource. *Change* needs to be a first-class concept and must apply at all levels of structure, from the smallest informational atom to the largest design organism. Furthermore, the change model needs to bolster the instruments (tools and views) used by both designers and non-designers to probe the growing design for weaknesses. It also needs to organize the pathway for results of such inspections.

Economic forces are pushing industry toward agreements at finer grains of information structure. Together with a simultaneous increase in system complexity, this implies an explosion in data population and, more importantly, data relationships. This places a high demand on coherent and efficient representations not only for the product, but also for the process and the team organization. The primitive elements of the process are change and change notification.

Change is intrinsic to all design processes, especially ones that involve concurrent input from multiple perspectives. A formal model of change provides a calculus for defining critical aspects of the process by which a team and their tools work together: concurrency, state dependency, compatibility and notification. Our model is especially suited to the various information grain sizes prevalent in current and future engineering environments, from coarse files down to the entities, attributes and links of concept lattice fields.

Exception Handling

In any system, the appropriate management of exceptional behavior is a critical but often ignored issue. In fact, much of a system's complexity may arise from exception handling. In a framework-based environment consisting of an "integrated" set of tools, exception handling is complicated by the way the tools are integrated, both with the framework and with one another. In particular, tools are typically built as stand-alone functional units with no knowledge of frameworks or other tools. As a result, different tools may have widely different, perhaps incompatible, notions of exception handling and may not provide adequate "hooks" through which the environment can detect and/or control exceptional behavior. A method for modeling exceptions and exception handling behaviors must be defined. Methods for attaching such behaviors to integrated tools must also be defined.

Multi-Domain Engineering Environments

Integrating engineering environments across multiple domains (mechanical, electrical, software) offers significant payoff in concurrent product development and integration. The critical technology involved is information model integration across domain boundaries.

Schema Federation

Mechanical engineering shares many concepts with electronics. Both domains use shapes, for example. Should the shapes used by the electronics engineer be identical with those used by the mechanical engineer? Even though the processes used to create them are entirely different? Much remains to be done in the realm of data semantics and information modeling.

Until that problem is completely solved (a lifetime venture), there still remains the issue of coercing one representation of an object into another representation. EIS currently defines services that define placeholders for future solutions. For example, EIS defines a dictionary that can be queried by remote sites to determine foreign schema information, but more needs to be said about what can be done with that information.

Networking Support

Current computer networking will have to be greatly enhanced to meet the needs of cost-effective and concurrent engineering information access for GITIS and CITIS environments. The use of integrating information gateways will be key to achieving the levels of technical interchange needed among organizations and different engineering environments. EIS technologies (framework and information models) provide the foundation for building products for transparent information access across the information exchange networks. Future technical work is needed in the areas of common exchange formats (CEF), exchange of objects and schemas among multiple EIS sites, and information exchange between EIS and non-EIS sites.

Enterprise Integration

Enterprise-wide integrated information systems must integrate process-oriented tools and information management systems together across a distributed, heterogeneous computing environment. Application of such systems must be planned and managed to yield effective results. This entails obtaining a clear understanding of the problem to be solved by the intended systems, and appropriate application of technological resources to construct the information system.

Previous Air Force programs have developed methods and tools for analyzing and understanding enterprise activities and information flows (i.e., the IDEF methods).

Other programs, including EIS, IISS and IDS have developed many of the information system technologies required to implement integrated information systems. Future efforts (e.g., those intended to formulate methods for optimizing enterprises) will always be based on models. Other avenues for extending capabilities include dynamic (e.g., simulation) models and the addition of knowledge-based capabilities (e.g., expert systems).

Models of the Economics of Integration

The EIS contract was initiated with a set of requirements already in hand, but without an analysis of the costs and benefits of having a system that meets those requirements. Neither was this done during the contract. One of the main problems with performing such an analysis is the wide variety of engineering processes that may be served by the technology. The payoff analyses must then take the form of case studies. Even so, more needs to be done in the way of understanding the various parameters involved. An "economic model of integration" must be developed. In addition to providing general arguments for using the technology, a detailed economic model can be used by users, system administrators and integration specialists when making decisions about when to apply the technology and how.

One of the key parameters to be addressed in such a model is the granularity of data representation. The predominant grain size of current technology is the "file." Much research is being done (including the EIS program) to push this granularity to finer levels of detail, but little is known of the costs involved in doing this. Finer granularity clearly taxes both performance and storage capacity, but what is gained in return?

Models of Other Enterprise Domains

As much as engineers would like to believe that technology alone makes a business successful, other aspects of the enterprise weave their way into the engineering process. These aspects need to be modeled and related to the technology models (e.g., the ECAD information model).

Along with these models, a tool kit is needed to accelerate their implementation, minimizing the time between concept and realization. The tool kit needs to support not only implementation of the data and process models, but also embedding metrics instrumentation to close the loop.

Information Visualization

As engineering information systems grow to accommodate more domains and more complex processes (e.g., concurrent engineering), providing meaningful access to new types of data becomes a critical issue. Competitive pressures demand that productivity continue to rise, diminishing the value of raw numbers and strings. New ways must be found for automatically rendering aggregates of engineering information that exploit the human visual capacity.

Related to visualization is workspace (or context) management. As product complexity rises and engineering transactions lengthen, new ways are needed to capture and replay "trains of thought." Some related research is already going on under the heading of "design record capture," but it is so far largely domain-independent and not integrated with the rest of the engineering environment.

U.S. Government Printing Office 648-127