Electronic Delivery Cover Sheet

¶ ILL SID: 02519 IL

Beginning of record displayed.

ILL Pending 20020603 Record 8 of 8

CAN YOU SUPPLY ? ▶YES ▶NO ▶COND ▶FUTUREDATE ¶

▶ :ILL: 7561599 :Borrower: CIT :ReqDate: 20020603 :NeedBefore: 20020703
:Status: PENDING 20020603 :RecDate: :RenewalReq:
:OCLC: 44746368 :Source: OCLCILL :DueDate: :NewDueDate:
:Lender: *OVV,OVV ¶

▶ :CALLNO: ¶

▶ :TITLE: Encyclopedia of microcomputers / ¶

▶ :EDITION: COULD YOU PLEASE SUPPLY THIS ARTICLE, YOU ARE ONLY HOLDER. THANKS, CALTECH.****** ¶

▶ :IMPRINT: New York ; B # : Marcel Dekker, 2000. ¶

▶ :SERIES: Encyclopedia of microcomputers / Kent, A. ; Williams, J.G. ; 25, no. 4 ¶

▶ :ARTICLE: Biren Prasad and Subra Ganesan: Concurrent Engineering and Work-group Computing', ¶

▶ :VOL: Book Chapter :NO: :DATE: 2000
:PAGES: 73-95 ¶

▶ :VERIFIED: <TN:134448>OCLC ¶

▶ :PATRON: Prasad, Brian ¶

▶ :SHIP TO: CALTECH/ILL/MILLIKAN LIBRARY 1-32/PASADENA, CA ¶

▶ :BILL TO: SAME ¶

CONCURRENT ENGINEERING AND WORK-GROUP COMPUTING

INTRODUCTION

Design and development of a product is a complex undertaking. During a product design, development, and delivery (PD^3) life-cycle aspect, a product goes through a number of changes. Some changes may be present as specifications. Some of these specifications make it robust; others are required to test the product under extreme operating conditions. Initial design specifications must accommodate the diverse environments that a product is subjected to during its normal life. The specifications or characteristics associated with a particular environment of a product constitute a framework that dictates an initial phase of its design. A study of a complete product life-cycle reveals varying operating conditions that must be accounted for during a product realization—PD^3—process. Frameworks are conceptual models of the operating conditions. All of these frameworks together form an architecture. Because the environments depend on each other, so do the frameworks. The goal is to develop a series of "building blocks"—concurrent engineering (CE) frameworks—that would provide the CE work-group members in a large dispersed organization the same freedom of interaction and information transfer as enjoyed by a small collocated team (1).

CONCURRENT ENGINEERING

The concept of concurrent engineering was initially proposed as a potential means to minimize the product-development time. Since then, many interpretations of "concurrent engineering" have emerged in literature. Today, CE is much more encompassing. Expectations range from modest productivity improvement to a complete push-button-type automation, depending on the views expressed. CE is a paralleled approach—replacing the time-consuming linear process of serial engineering and expensive prove-outs. It is intended to elicit the product developers, form the outset, to consider the "total job" (including company's support functions). Some common definitions are as follows:

- Concurrent engineering is "a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support." "This approach is intended to cause the developers, from the outset, to consider all elements of the product life-cycle from conception through disposal, including quality, cost, schedule, and user requirements" (2).

$$\text{CE} \Leftrightarrow \text{Paralleling of (Life-Cycle Functions)} \qquad [1]$$

- Concurrent engineering is "a systematic approach to integrated product development that emphasizes the response to customer expectations. It embodies team values of cooperation, trust, and sharing in such a manner that decision

making proceeds with large intervals of parallel working by all life-cycle perspectives early in the process, synchronized by comparatively brief exchanges to produce consensus."

CE ⇔ Paralleling of (Life-Cycle Functions)
+ Consensus + Cooperation [2]

- The Computer-aided Acquisition and Logistics Support (CALS) Office definition of CE from Military Handbook-59 is "a systematic approach to creating a product design that considers all elements of the product life cycle from conception through disposal." In so doing, "CE defines simultaneously the product, its manufacturing process, and all other required life cycle processes, such as logistic support." "CE is not the arbitrary elimination of a phase of the existing, sequential, feed-forward engineering process, but rather the co-design of all downstream processes toward a more all encompassing, cost effective optimum. . . . Concurrent Engineering is an integrated design approach that takes into account all desired downstream characteristics during upstream phases to produce a more robust design that is tolerant of manufacturing and use variation, at less cost than sequential design."

CE ⇔ Cost-Effective Robust Design
(Conception Through Disposal)
+ Simultaneous Design-of
All Downstream Processes During Upstream Phases [3]

- Concurrent engineering is a goal-directed effort, where "ownership" is assigned mutually among the entire group on the "total job" to be completed, not just "pieces" of it, with the understanding that the team is empowered to make major design decisions along the way. This definition is more suited to a virtual CE enterprise, where each regional business unit is essentially its own separate business responsible for products sold in that region. Regional business units are empowered to make design decisions based on goals set with parent organization (with constancy of purpose).

CE ⇔ Minimization of Total Life-Cycle Time [4]

- Concurrent engineering is a product development methodology where up-front "X-abilities" (such as manufacturability, serviceability, quality, etc.) are considered part of the product design and development process. X-abilities are not merely for meeting the basic functionality or a set of limited strategies, but for defining a product, that meets all the customer requirements.

CE ⇔ Up-Front Consideration of All X-abilities [5]

- The narrow view of concurrent engineering is "integrating product and process design." The wide view is "integrating over the product life," and the wider view is "integrating over the enterprise".

CE ⇔ Integrating Product and Process Design over the Enterprise [6]

The most commonly referred to definition is that of Winner. Some experts recognize influencing agents of CE as forces of change. We have chosen to divide forces that influence the domain of CE into seven agents (called here as 7 Ts): talents, tasks, teams,

techniques, technology, time, and tools (see Figure 4.1 of Ref. 2). One of the primary team issues is the decomposition of tasks. The people's issue is the composition of teams. *Teams* are often used to cooperatively solve the problem. *Technology* issues arise due to drive for competitiveness. Examples of popular technologies in CE are soft prototyping, visualization, product data management, design for X-ability, multimedia, electronic data interchange (EDI), and so forth. *Tools* mean software, hardware, and networks that make CE practical in today's world of multinational corporations, multipartner projects, and virtual corporations. From the *time* point of view, CE is an initiative of the product-development community that has the goal of reducing the length of the product design and manufacturing cycle time by allowing teams of engineers to develop design modules concurrently from their perspectives (3). *Training* also plays an important role in CE. A popular word in the business press is reengineering, meaning, in short, revamping the process by which one satisfies customers needs. From a business angle, CE means reengineering the product-development process so that tasks are organized concurrently. The Department of Defense (DOD) and some aerospace companies refer to CE as integrated product development (IPD).

$$\text{IPD} \Leftrightarrow \text{Minimize Cycle Time} + \text{Paralleling of Life-Cycle Functions} \qquad [7]$$

DISTRIBUTED COMPUTING

Most traditional architectures are based on mainframes and minicomputers connected through a centralized relational database system (RDMS). Most of the RDMS (e.g., Ca-Ingres, Oracle, Sybase, DB2, etc.) provide a typical semblance of remote computing (see Fig. 1a). However, their centralized nature of installation causes performance problems, storage retrieval delay, and maintenance problems. Personal computers, on the other hand, provide better access for doing routine tasks but are not designed for use by a group of people (called work-groups). They usually lack computing power and are not well connected. Local-area network (LAN) or wide-area network (WAN) is usually added on an ad hoc or afterthought basis.

Distributed computing can provide a balanced solution for large-scale multidisciplinary computing if appropriately designed (see Fig. 1b). Workstations and servers shared by a group of teams (called work-groups) are the two main ingredients for building a distributed computing network environment for concurrent engineering. The smallest unit of this network is called the *work-group network*. A typical work-group network unit comprises of one or more clients, consisting of powerful desktop workstations, interacting with one or more servers that store and manipulate information. Relevant programs are stored on the servers so that they can be accessed from any workstation on the network. This enables an efficient and timely sharing of data and resources among all the members of a work-group connected to a network. A work-group network is equipped by design to provide the right computing resources (capacity and power) for a specific group of concurrent teams to do their job efficiently. Depending on the size of the work-group, one or more work-group networks may be needed and interlinked. A distributed computing environment consists of a chain of interconnected networks linking to work-groups at various levels in a CE organization like a tree structure. The connections between a CE unit and the rest or organization are established in the following way (see Fig. 2):

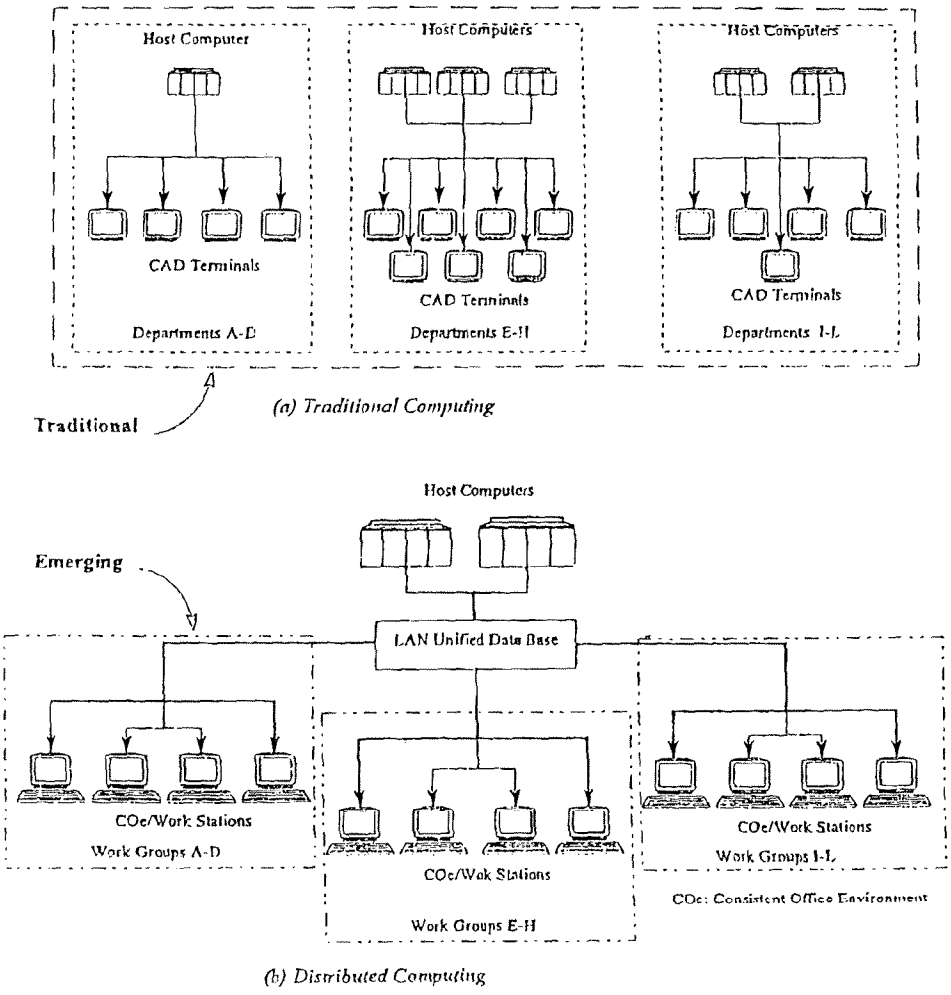

FIGURE 1 Information processing: (a) traditional computing and (b) distributed computing.

- A PDT unit through a "work-group network"
- A PDT's design unit through a "design center network"
- A PDT's manufacturing unit through a "manufacturing center network"
- The executives on top management board through an "enterprise network"

The networks are all a part of a network taxonomy for a distributed computing environment. A center network may consist of a *center/unit server* and one or more of these linked work-group networks. An enterprise network forms the corporate-level server connection down to the center/unit networks. It provides work-group access to organization-level applications such as business and office automation tools. A distrib-

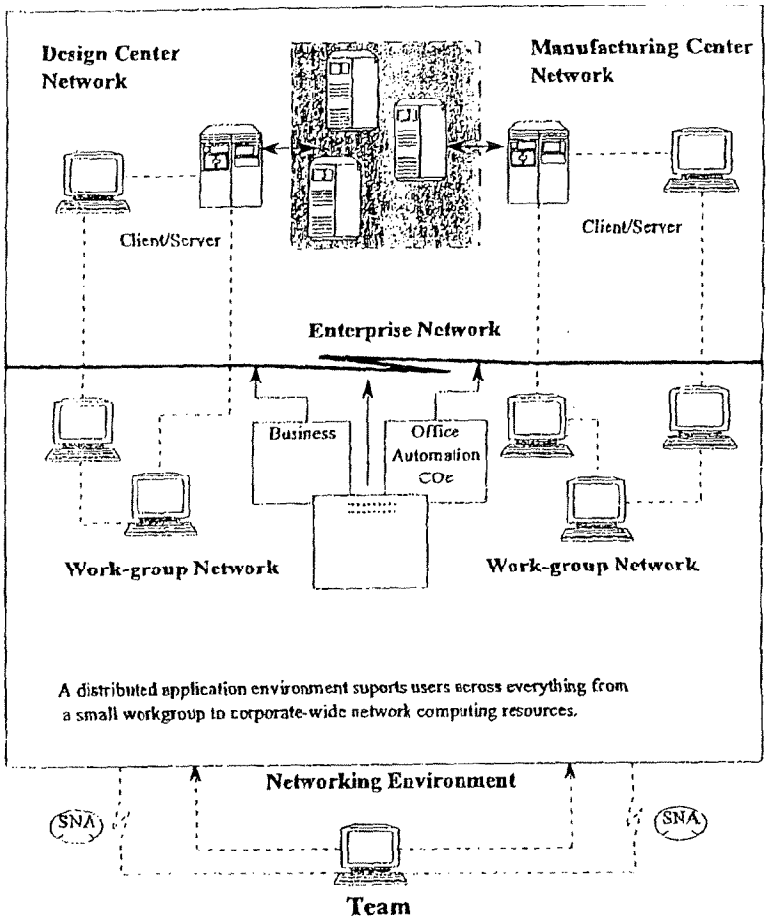

FIGURE 2 Distributed applications environment.

uted application environment is thus capable of supporting concurrent teams across every level from a small work group to corporate-wide computing resources. Others in the team who have need to access information can do so through the remote connection (e.g., SNA protocols) as shown at the bottom of Figure 2. Servers, at each center, are designed to provide the needed compute power and to be responsive to individual run-stream commands. Recent trends, therefore, are to replace the central database stored on mainframes and minis by distributed databases on server-based computers. CORBA (Common Object Request Broker Architecture) provides a distributed objects' capability—an ability to invoke objects residing on multiple platforms.

WORK-GROUP COMPUTING

In recent years, an architecture called "work-group computing" has emerged due to the efforts of many workstation vendors competing for CE market share (SUN, HP, DEC,

and IBM, among others). It provides a better integrated environment for CE compared to LAN-based PC networks. With "work-group computing," computing tasks are distributed between "clients," consisting of powerful desktop workstations and "servers" that store and manipulate information (see Fig. 3). Work groups are provided a view into the computing complex through a "window" created by the client workstation. They may, how-

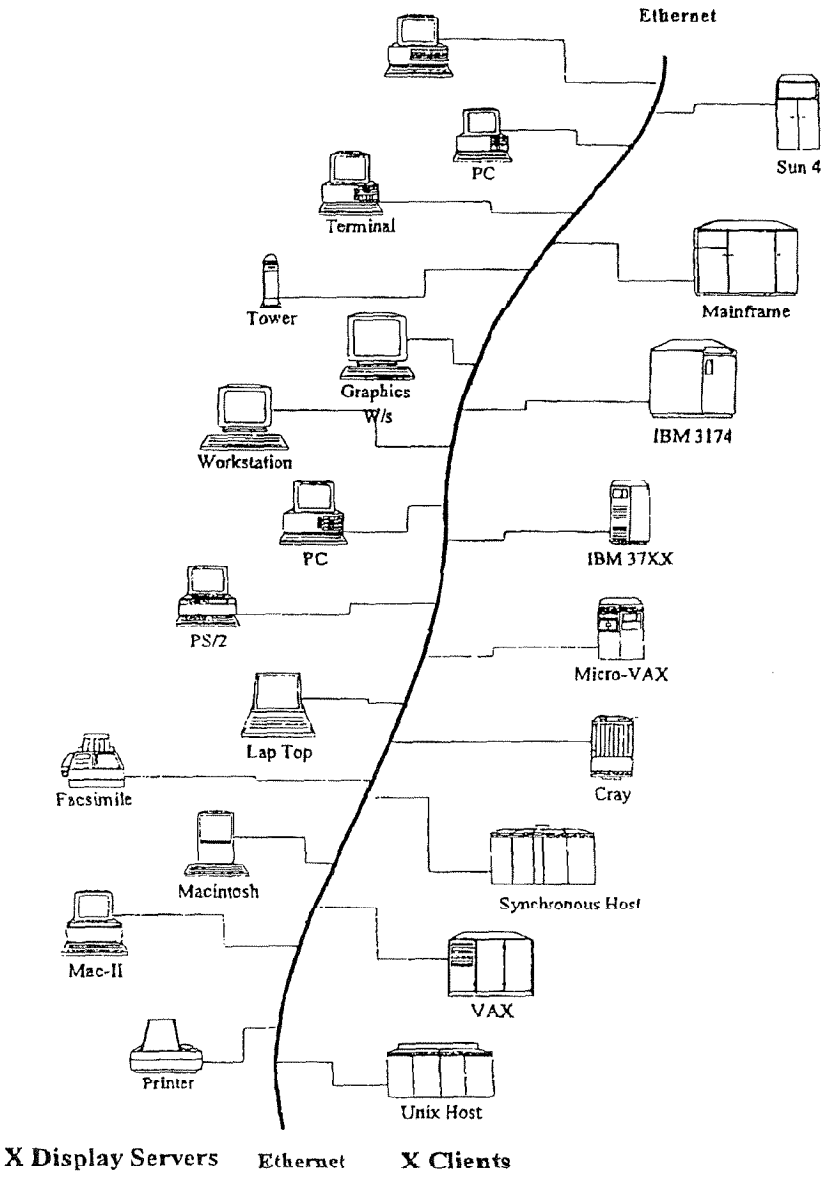

FIGURE 3 Work-group computing.

ever, be linked with virtually anyone else (see Fig. 3). The X-display servers can range from a printer, to a facsimile, to a workstation. Although mainframes, minicomputers and microcomputers suffice for general needs, there will always be a need for more specialized machines. That is why work-group computing is built around client–server architectures. In addition, modern databases can be distributed over many different machines, so work-groups can create and execute applications locally on their own computers. These applications could look and feel just like separate programs. Here, individual workstations—the clients—handle the local processing needs; the server has the power and capacities to access data from distributed databases beyond that of an individual workstation. They may provide heavy-duty number crunching, distributed databases, and links to outside resources. When a work-group runs the applications from any workstation, it draws on the resources of all the related applications and databases over the network no matter where they physically reside or are stored. This distributed concept maximizes the computer power needs of the work-groups with the least amount of investments.

Work-group computers are designed from the ground up. The intention is to help teams work together and automate group processes, including engineering, manufacturing, and other complex tasks. Because open systems are built around industry standards, they can be integrated easily with existing equipment from various vendors. Work-group computing provides the power to perform individual tasks with ease while opening up the possibilities of information sharing between parallel work-groups. Figure 4 illustrates the shift in key characteristics when moving from personal computing to work-group computing. The shift pulls everyone—teams, computers, networks—transforming disparate computers into one flexible, easy-to-use client–server-based system. The individual task-oriented environment of personal computing activities becomes a set of goal-oriented parallel work-group activities. The use of a distributed database over the network becomes the mainstream norm for file management as opposed to a local database resid-

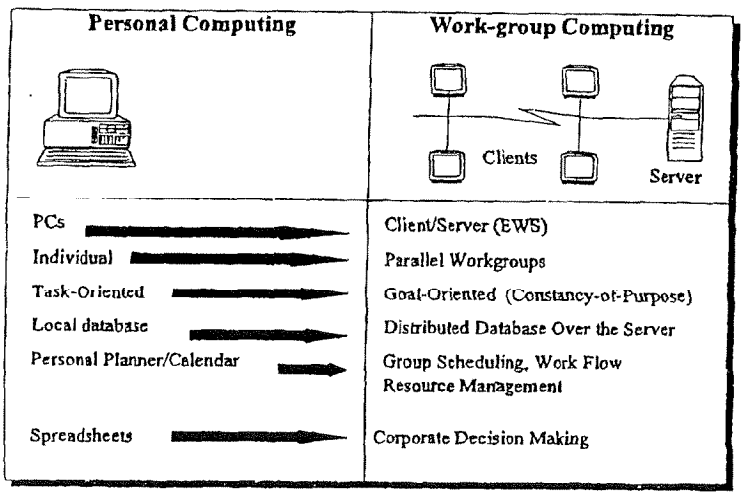

FIGURE 4 Shift from personal computing to work-group computing.

ing in one's own personal computers. The use of a personal planner and a calendar is replaced by a group scheduling system and electronic work-flow resource management system. With work-group computing, manufacturing organizations, small and large, can all benefit from concurrent engineering.

Parallel Work-Groups

While implementing CE, a major challenge an organization faces is to provide a seamless connection between parallel work-groups and computing machines. Work-group computing provides a basis of distributing the work into cohesive parallel teams working in close association with each other. An example of such a distribution is shown in Figure 5. Here, four concurrent work-groups—engineering, design, prototyping, and manufacturing—are shown to be working together, each with its own work-group computing system. The terminals represent the concurrent tasks that are being performed by a term within a work-group. For example, in a design work-group, different teams at any point may be working concurrently on tasks such as concept design, detail design, solid modeling, detailed analysis, drafting, and so forth. Scheduling of work-group tasks follows the integrated product development (IPD) methodology as discussed in Ref. 4. The division of tasks follows the hierarchy of the breakdown structure tress [work breakdown structures (WBS), product breakdown structures (PtBS), process breakdown structures (PsBS), and so forth] (2). Transparent communication and access to common databases provide a mode of constant communication and frequent interaction between the work-groups. The network is represented in Figure 5 by a thick horizontal wavy line. It runs continuously and crosses through the work-group partitions not shown in Figure 5. The vertical down-arrows connect the work-group workstations, terminals, personal computers, mainframe, minicomputers, and the corresponding database server to a network. The network ensures that the messages created by a work-group or a team member are passed on to the work-groups and that the changes that affect the design outcome are propagated throughout the CE organization.

CONCURRENT ENGINEERING ARCHITECTURE

A compendium of abstractions (called frameworks) leads to a CE architecture. These levels of abstractions are not progressive in nature (have no definite sequencing order) but are need based (see Fig. 6), hence they are called frameworks. These frameworks are as follows:

- Directional framework
- Conceptual framework
- Relation framework
- Logical framework
- Physical framework
- Application framework
- Instruction framework

The following sections describe each in more detail.

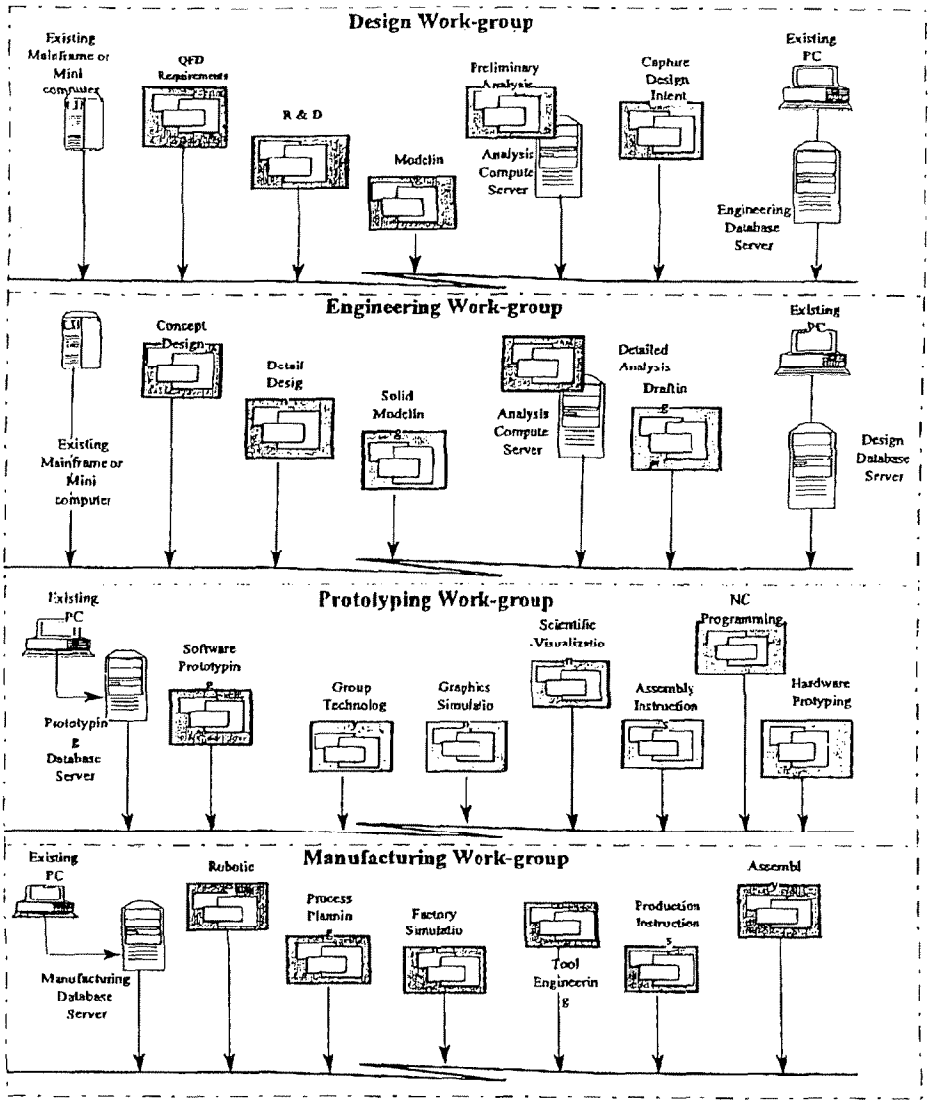

FIGURE 5 The parallel work-groups of concurrent engineering.

Directional Framework

The directional framework focuses on the enterprise's global needs (vision, mission, objectives, goals, etc.). A top-level abstraction for a CE architecture defines the product vision in relation to external (sometime referred to as uncontrolled) environments such as vendors, suppliers, field support, marketing, and customers. Examples of system frameworks are the Air Force's Integrated Computer-Aided Manufacturing (ICAM) program, CASA/SME (Society of Manufacturing Engineers), CAM-I, and many others. The

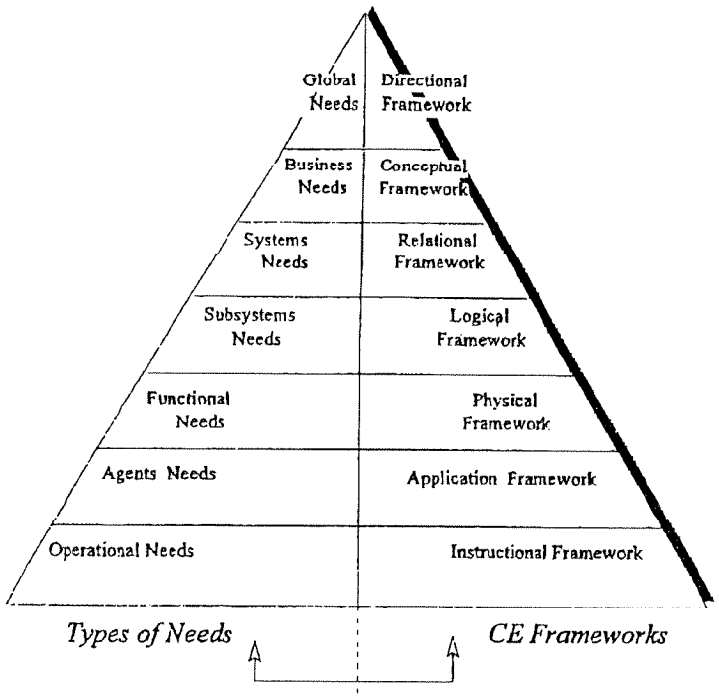

FIGURE 6 Type of needs and CE frameworks.

dominant means to meeting the challenges are global thinking, common tools, consistent standards, and uniform methodology applied across the enterprise.

CIMOSA Architecture

Very few methodologies cover the entire PD^3 development cycle—from analysis to operation to maintenance of a CIM system. CIMOSA (Open System Architecture for CIM) is one of the most complete methodologies in terms of life-cycle coverage (see Figure 7). It was developed by the AMICE consortium in 1986 under the auspices of EC ESPRINT (European Strategic Program for Research and Development in Information Technology) project funding. The AMICE (European CIM Architecture—in reverse) consortium now comprises 15 companies and research institutes from 8 European countries.

CIMOSA architecture is comprised of the following:

A Modeling Framework. This provides modeling structures—how CIM systems should be modeled. It organizes the CIMOSA reference architecture into a generic and partial modeling level, each one supporting different views on the particular enterprise model. CIMOSA has defined four modeling views: function, information, resource, and organization. This set of views may be extended as well. This concept of views allows teams to work with the subset of the model rather than the complete model. Users can view only what they are interested in viewing, hiding the complexity from the particular area of concern.

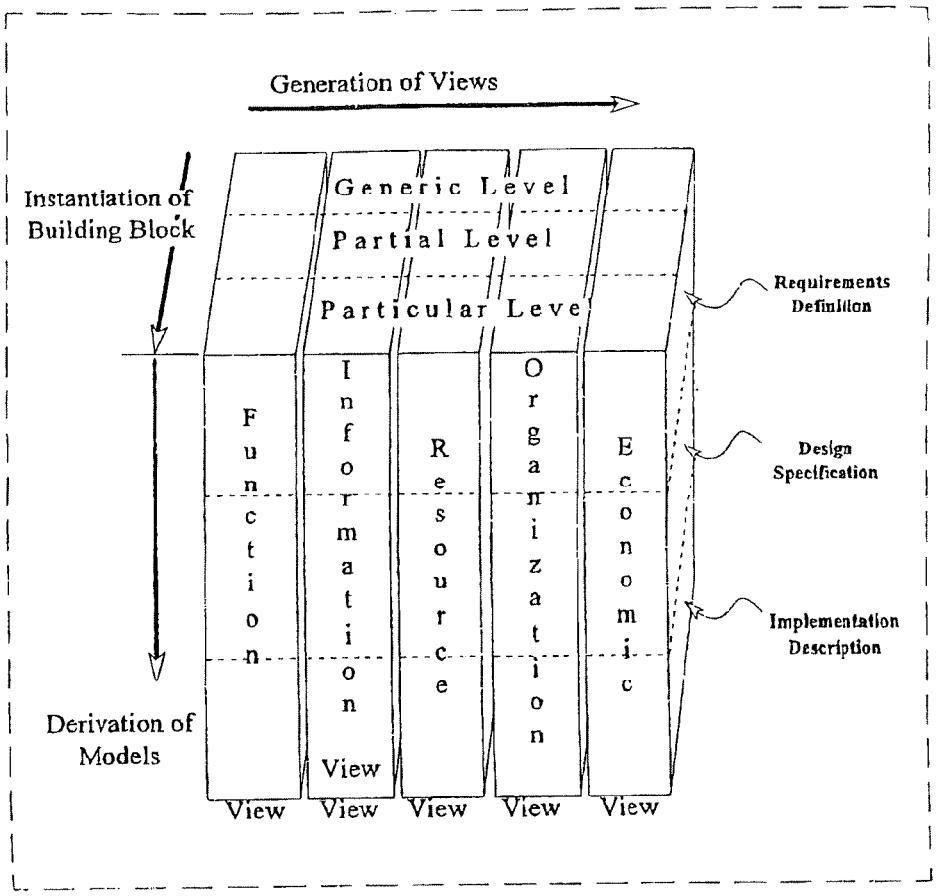

FIGURE 7 CIMOSA architecture.

System Life Cycle. This describes the operations or tasks to be used to generate CIMOSA models. The architecture supports modeling of three life-cycle operations: requirements definition, system specification definition, and implementation description. The sequence of modeling is optional (i.e., modeling may start at any of the life-cycle phases and may be iterative as well).

An Integration Infrastructure. This provides a set of generic (system wide) information technology (IT) service entities and resources to support the execution of CIMOSA models in a heterogeneous environment. The types of service include management entity, business entity, common entity, information entity , presentation entity; resources include information technology resources and manufacturing resources. Control on the execution of the implementation description model is provided by the "business entity," which receives the events and creates occurrences of the domain process and all of its contents.

These three together form a three-dimensional framework for the models, as shown in Figure 7, also known as the "CIMOSA CUBE." CIMOSA provides the basic framework for evolutionary enterprise modeling. CIMOSA is based on the object-oriented concepts of inheritance (i.e., structuring its constructs in recursive sets of object classes).

Conceptual Framework

The conceptual framework addresses high-level CE business perspectives (e.g., strategies, objectives/goals) for developing a PD[3] system (1). Examples and tools that fall in this category are multidisciplinary or cross-functional team, work-groups, quality function deployment (QFD) and its replacement, concurrent function deployment (CFD), total quality management (TQM) and its replacement, total value management (TVM), and so forth.

Relational Framework

The relational framework expands conceptual perspectives into system needs and then forms the next level of abstraction. As the name suggests, it describes the relationships between processes and programs. The Air Force ICAM program and its successors have generated extensive relational models of manufacturing organization (5). Many changes have come about in this architecture from the time it was initially introduced. The growing PDES/Express specification is a good example of a relational model for product-definition databases, although Express is also being used as a data-definition mechanism. Less formally, the Commission of the European Communities (CEC)—ESPRIT (European Strategic Planning for Research in Information Technology)/CIM program has generated a book-sized flowchart description of a typical product-development organization (6). The key to meeting the CIM challenge is enterprisewide system thinking on the relational framework, with standards being the glue to make it all stick.

Logical Framework

The logical framework defines subsystem needs. It depicts logical design information with respect to a relational framework. It provides symbolic descriptions of the processes and programs to define the logical view of the system. The types of symbols that identify the structures of the framework are CAD, CAM, CAE, object-oriented databases, and so forth.

Physical Framework

The physical framework defines the functional needs for logical and relational frameworks so that they can work together. This level of abstraction defines the following:

- A computer system
- A network that connects a collection of computers
- A data representation [e.g., data-definition language (DDL), structured query language (SQL), etc.]
- A communication interface like Ethernet
- A database that stores the underlying information

An example of a typical physical framework for a computer system is shown in

Figure 8. The operating system and utilities are shown to be the gatekeepers for I/O transfers to CRT terminals, peripherals, and other computers.

Application Framework

The application framework defines the agent's view. This defines the specifications of the physical framework components in relation to both hardware and software needs such as operating systems (DOS, UNIX, etc.), basic programming languages (PL1, UNIX, C, C++, FORTRAN, etc.), and hardware platforms (IBM RT's, SUN, HP, etc.).

Instruction Framework

This is the lowest level of abstraction and provides the basic instruction sets for computer programs or systems. This may include data encoding, character coding, data conversion, and program formats (real, alphanumeric, or floating point). Although instruction frameworks have considerable impact on physical applications, their lack of standardization is a well-known annoyance.

In the aforementioned section, CE architecture was classified into seven levels of abstractions. Figure 9 summarizes the salient features of each level with examples and tools in a tabular form. The examples and tools are only representative samples to show

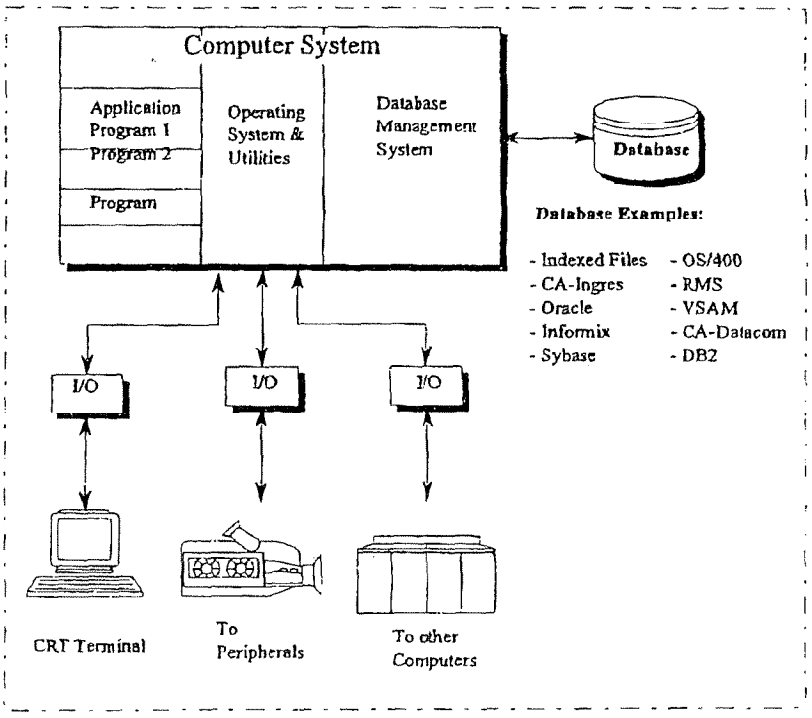

FIGURE 8 Examples of a physical framework.

Type	Description	Examples and Tools
Directional	Enterprise global needs and directions (Vision, Mission, Objectives, Goals, etc.) High level reference models	Air Force ICAM Program, CIM-OSA, IBM CIM, CASA / SME, CAM-I, etc.
Conceptual	The high-level requirements or taxonomy for developing a product or a system. It addresses the major business needs	QFD, Multi disciplinary teams, Work-group, TQM, etc.
Relational	Relational descriptions of processes or programs (not executable). It addresses the major Systems needs	IDEF, PDES / Express, ESPRINT/Amice E-R models process, etc.
Logical	Symbolic descriptions of processes or programs to define the system. It defines the major subsystem needs.	Design, Object data bases, CAPP, CAE, CAD, CAM, etc.
Physical	It defines the functional needs for Relational and Logical frameworks to work together.	DDL, SQL, Ethernet, TCP/IP, and other communication Interfaces.
Application	Specifications of the physical framework components – hardwares and Softwares	DOS, Unix,... ,etc. PL/I, C, C++, Fortran, etc. IBM RT, SUN4, HP, etc.
Instructional	Basic instructional set – Data descriptions, Information flow and formats	CISC / RISC Instruction sets, character codes, formats.

FIGURE 9 Salient features of CE architectures.

the main distinctions among these different levels. The boundary between two consecutive frameworks such as application framework and instructional framework, or physical framework and application framework, is rather fuzzy. The first four frameworks, directional, conceptual, relational and logical, are more distinct than the rest. Their models are more formally defined than the rest and, unlike the others, they are not directly executable by computer programs or systems. Logical and physical frameworks are closely related. Although not executable in the same environment, logical and physical frameworks share the same goal. They represent a major contribution to the success of the product life cycle and, together, they constitute a "computational environment" for a CE-based PD^3 system.

CE COMPUTATIONAL ARCHITECTURE

Computational architecture (CA) is a 3-in-1 deal—a fusion of three framework components: relational, logical, and physical. Because of their relevance to the CE field, this article focuses on computational architecture.

A CE developmental environment is actually an implementation of relational and logical objects into a particular physical architecture. An object is a particular design method, too, or an advisor that is executed from within a computational environment. The object may be "extensible" (i.e., other tools are accessible as native resources for an

application) or "open" (i.e., the facilities of that object are made available to other tools). A server is an object that provides facilities to other objects without a direct user interface of its own. An extensible environment that provides teams access to the facilities of one or more servers, work-groups, and foreign codes is called an "open environment."

Requirements for a Computational Architecture

In Ref. 2, the requirements for concurrency between different work-group configurations and their degree of involvements were presented in a matrix form (see Figure 4.13 of Ref. 2). Although significant portions of any computational framework meeting these requirements must be included, concurrent engineering systems must also operate under other kinds of constraints related to infrastructure (see Figure 6.13 of Ref. 2). A computational architecture for CE must exhibit the following qualities:

- *Accommodate distributed computing environment.* Computational architecture must support a distributed computing environment based on an open-networking context to allow teams to work collaboratively, unhindered by types of computer platforms, operating systems, network protocols, and so forth. The CA should have seamless communication among work-groups (with appropriate level of security).
- *Existing applications and product-development procedures.* Because of large prior investment in tools and commitments to ongoing operations, most organizations will not be able to start with a clean slate. The CA should work temporarily with the legacy system without interrupting operations significantly for any reason while a new framework is being designed, developed, and installed.
- *Emerging frameworks and applications architectures.* Vendors and organizations developing software and hardware will continue to produce new frameworks/applications. This might dictate changes to the data or configurations of the computational framework. The computational architecture should allow all such tools to be integrated so that minimal effort is required to move data from one tool/application to the next.
- *Efficiency.* Methods, tools, and advisors within the framework should run at speeds comparable to their independent execution. System overhead must be commensurate with the benefits accruing from its use.

CE Developmental Environment

One of the purposes of computational architecture is to easily build new end-user applications that not only provide a new set of CE functionality but are integrated with the rest of CE applications developed earlier. As a development platform, the computational framework for CE can be specified in terms of the following nine-layer toolkit: user interfaces, command language, high-level application language, programming language, geometry engine, data and memory management, the interface engine, communication and networks, and, finally, standards. The specifications apply to the syntax, semantics, and representations of each of the above nine basic functional layers. Computational architectures range in complexity from time-sharing operating systems and conventional languages (FORTRAN-like) to the newer object-oriented languages and databases defined by organizations like the CAD Framework Initiative (CFI) and some of the system vendors. Today, many computational architectures (as a developmental toolkit) seem to converge on some variation of the following nine-layer configuration (Fig. 10):

Facility	Description
1. User Interface	Mechanism for user to activate and monitor application facilities (Window to the world)
2. Command Language	Language for defining user interfaces and capturing command sequences
3. Higher Level Application Language	Symbolic representations of attributes inputs/Outputs and other design and manufacturing parameters
4. Programming Language	Languages for expressing, developing and writing application functions
5. Geometry Engine	Languages, facility, library and utilities for modelling geometry of parts, family of designs and defining topology
6. Data and Memory Management	Languages, facilities, Library and utilities for managing information, sending messages, changing files, records and objects
7. Interface Engine	Chaining a set of instructions to infer results or an outcome of an event (values of an attribute)
8. Communications and Network	Communication protocols facilities, library and utilities for communication amongst heterogenous h/w & s/w environments
9. Standards	Introduction of standardized neutral forms, languages and interchange formats

FIGURE 10 A generic CE developmental environment (nine-layer configuration).

- *User interface.* The user interface (UI) is often considered the "window to the world"—an external mechanism whereby a team or a work-group can execute system functions and graphically view the results. The common interface structure allows the team members to move from one computer application tool to another, with minimal learning. The same look and feel aspect of UI allows the teams to concentrate on the tool and not on the interface. A simple example of UI is the pull-down menu in any window-based environment. The current emerging standard appears to be some variation of the X-Windows graphic windowing system with OSF/MOTIF libraries.

- *Command language.* This enables team members to issue interactive commands with programmatic modification capabilities in object-oriented setting. The definition and interpretation of computational framework are done with the help of a command language. Using the command language, one can execute sequences of functions provided by an application. The Macintosh Hypercard language Hypertalk is a representative of this class of languages.
- *High-level application languages.* As application languages evolve further, programming languages support more English-like syntax. At the 4GL level, one merely tells the application what is to be done, and the program automatically figures out how to do it. The term 4GL encompasses not only the particular programming languages but also the entire set of software-development tools that are associated with these languages. Such tools include the following:

  Debuggers
  Text database editors
  Application and menu generators
  Forms and report writers
  Screen layout
  Presentation systems
  Data manipulation languages
  Data dictionary

- *Programming language.* This includes the language and compiler (usually) for defining the functions needed by the application. Although FORTRAN is still a popular language in engineering design, C language is dominating many new applications and C++ object-oriented extension to C appears to be the choice of many electronic CAD system vendors.
- *Geometry engine.* The geometry engine has five principal elements: 3D wireframe, constructive solid modeling, 3D solids, rendering and scientific visualization, and interactive photorealistic rendering (see Figure 7.7 of Ref. 2). Solid modeling provides a more natural understanding of proposed designs and makes it easier to discover the relationships among systems, structures, materials, and processes. Moreover, a solid model offers an unambiguous definition of geometry and topology, simplifies computation of physical properties, enables detection of interference, and supports determination of dimensions and tolerances.

  Associativity helps to create 2D drawings from 3D solid models. Essentially, 2D views are taken off a 3D solid model and automatically placed in a 2D drawing. Small changes in the 3D model can be automatically reflected in a 2D drawing. A common math-based representation for solids can communicate seamlessly between various modules. For instance, a unified solid modeler can communicate with modules for production drafting, 3D modeling (including wireframe, surface, and solid modeling), FEA, and NC programming.
- *Data and memory management.* This includes utilities and libraries for managing collections of objects, files, and records in memory, for use by the application languages, or on permanent storage (disk). It can add functions, such as predrawn symbols, for other CAD/CAM packages. Relational databases have been standardized by ANSI. Object-oriented disk databases (the disk-based equivalent of the memory management systems mentioned in Fig. 1b) com-

bined with ionic desktop user interfaces seem to be the most likely technique for management of files in the future. A number of new database products are emerging and existing relational database vendors are extending their products with object-oriented facilities such as clustering and rules. However, an early resolution in this area seems unlikely.

Gaining in popularity are object-oriented database extensions of memory object managers of languages like C++. Soon, system vendors may provide language-independent object management systems and shared access to common object pools by independent applications. Some sort of run-time object manager will also be required because most existing object managers do not save enough class information for run-time dynamic objects. Object-oriented databases such as ROSE (7) can provide such layers. Because of considerable technical complexity, the development of standards in this area will take some time.

- *Inference engine.* Inference engine allows teams to define rules for capturing design intent and processing it in a "demand-driven" mode. Conventional artificial intelligence (AI) tools emphasize symbolic processing and nonalgorithmic inferences. In order for the resulting distributed intelligent environment to be powerful, it is imperative that the inference engine has abilities to conduct both heuristic and algorithmic reasoning.
- *Communications and networks.* It must facilitate information sharing and enable organizations to coordinate their teams' activities and resources. The idea of an enterprise integration network is to provide electronic exchange of information and services both internally among work-groups, teams, and departments of a company, and externally with customers, suppliers, and strategic partners. It must provide electronic access of information to the CE teams of technology, standards, methodology, and 3Ps. Teams should find meeting on the network useful to collaboratively guide and develop the necessary product ideas aimed toward a global product realization strategy.
- *Standards.* Defining common semantics and schema for the objects are difficult propositions. With a number of viable options emerging, such as the PDES/Express language, some standardization in this area seems likely in the next few years.

A CE developmental environment must permit any application tool to access any product data in the PIM and process it at a workstation—whether or not any of these elements are inherently compatible. This is the underlying role of the CA framework: A tool can be modified or updated any number of times, or replaced with a similar tool from another vendor, and it must still function without special preparation or custom programming. It must support standard protocols (e.g., DDL for windows that will make them "plug compatible"), allowing teams to request information and services from one another. The possible examples tools for various layers of this developmental environment are shown in Figure 11.

Models of Computational Architecture

In the last few years, many models of computational architecture have emerged. Each defines a consistent interface that has the same "look and feel" for all applications, thus allowing CE work-group members to devote their creativity to perform PD^3 functions rather than learning the infrastructure. An overview of an approach, which meets these

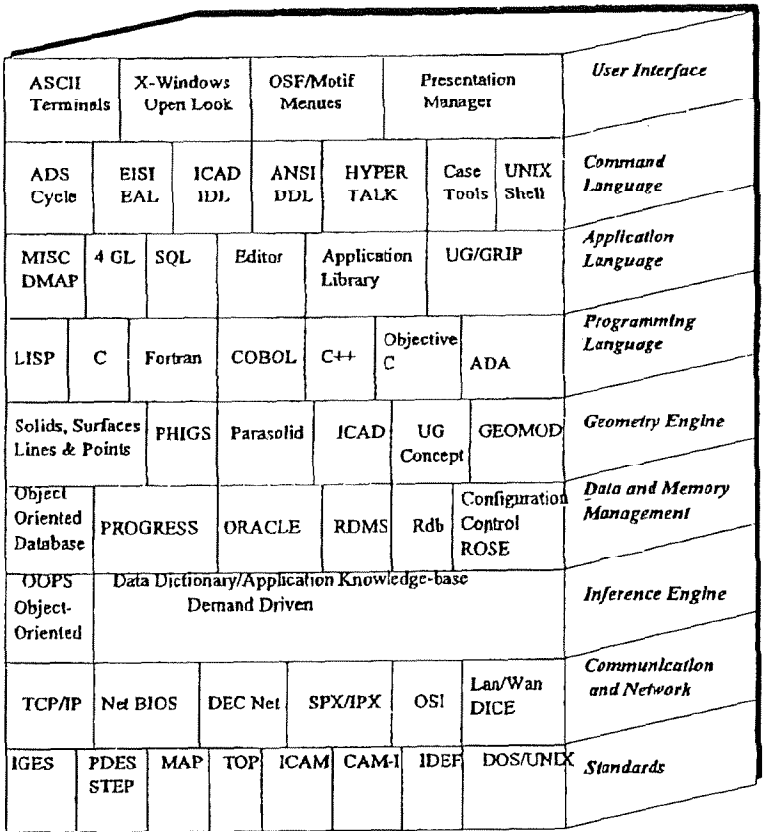

FIGURE 11 Possible example tools for a CE developmental environment.

constraints, is proposed by Lewis (8). The basic idea is to implement most product-development methods, tools, and advisors as independent servers or libraries. User interface or data management calls are not included as part of these libraries. The work-groups and data interfaces are implemented through a collection of open environments based on one or two application frameworks described earlier. Consequently, individual applications will not require extensive modifications for integration into the rest of the open environments. Applications are thus insulated from changes in the frameworks, other included applications, or the user interface. Some significant examples of computational architectures developed for CE are CASE (9), DICE (10), Design Fusion (11) and Next-Cut (12). They are discussed in the appropriate references cited.

CE SUBARCHITECTURE OR A FRAMEWORK

The goal of a framework for CE is to provide a flexible application development environment that shields end-user applications from possible downstream changes. The CE framework is organized into a three-layered system. Figure 12 shows a logical view of

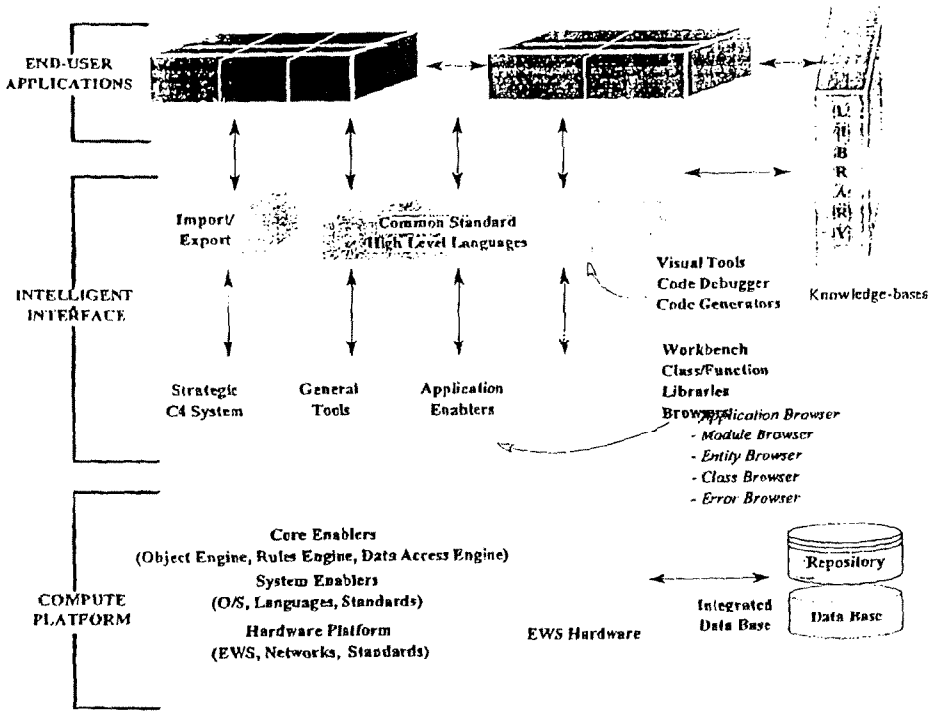

FIGURE 12 A logical framework for CE.

this CE subarchitecture, which forms the basis for the flexible CE environment described in this article. The lowest layer is the computing platform. The second-layer—intelligent interface—provides the primary programming interface to application developers. The top layer consists of end-user applications communicating among themselves (horizontally) and to the intelligent interface (vertically):

$$\int [\text{Compute Platform} \oplus \text{Intelligent Interface}] * \Delta\text{Standards} \Rightarrow \text{Long Life of End-User Applications.} \quad [8]$$

When computing platforms with intelligent interface are integrated over the applicable standards, this results in a long life of the end-user applications developed on the top.

Computing Platform

This is the bottom layer of a CE subarchitecture. It consists of actual hardware platform [such as engineering workstations (EWS), networks, and standards], system enablers (such as operating system, languages, and standards), and core enablers. The core enablers are the core tools that enable transparent access to database and other system enablers. Core enablers have three engines:

Object engine. This is used for managing objects supporting complex hierarchies of objects arranged into classes, subclasses, and instances.

Rule engine. Because different engineering, manufacturing, or business problems require different kinds of reasoning, a versatile rule engine is required that gives the work-group members a choice of various rule-based reasoning techniques:

- Forward chaining for data-driven reasoning
- Backward chaining for diagnostic-style reasoning
- Backtracking for intelligent searching and the management of iteration and looping

Data access engine. For mapping data from a relational database onto objects.

Intelligent Interface

This layer consists of strategic C4 (CAD/CAM/CAE/CIM) system, general tools, and application enablers (see Fig. 12). The application enablers create and foster the adoption of a base set of enabling features, functions, and interfaces (called engines). The enabling base provides an environment for building applications utilizing the engines. The intelligent interface layer includes the following:

Strategic C4 system. This is a set of C4 utilities—a set of graphical end-user interface construction facilities. It is employed to construct a common knowledge-base model. The common knowledge-base model is a common part/feature-based model. This enables a view of the design that is common across all CE disciplines.

General tools. Interactive C or C++ environment facilitates development of object-oriented (C or C++) source codes.

Application enablers. This is a developer interface for developing object bases, rule sets, and database mapping in a graphical environment. Examples include workbench, class/function libraries, browsers, and so forth. Browsers, for example, can further be broken down as application browser, module browser, entity browser, class browser, and error browser. Application enablers can be used to build model drivers. Examples of model drivers are equation solvers, optimization tools, and visualization tools

The intelligent interface contains import/export facilities, common programming standards, and high-level language processors. High-level languages help in capturing business decision-making processes. Examples of such processes may include visual tools, code generators (CASE tools), and a code debugger. Business logic, navigation, and database operations are triggered by visual control. The application language combines rule, analysis, and procedure-based descriptions, along with sophisticated pattern-matching capability. The application language is tightly integrated with the intelligent interface system. Access to the common knowledge-base model is achieved through the application language that allows the work-group members to query the model in their own terms. The application language can be used in conjunction with, or even as an alternative to, the C++ language to write procedures for object system methods and monitors, rules, or even a command language. The language itself may be either interpreted or compiled in a single phase during development, thus providing one language, with a single syntax, for a variety of uses.

An intelligent interface with high-level language provides a full application pro-

gramming interface (API) to the various engines. This allows an application developer to function in any role within a broader application architecture, including back-end server, intelligent front end, and run-time library.

END-USER APPLICATIONS

The top layer contains the production-user interface that integrates all the functional layers into the so-called end-user applications. This layer is equipped with a knowledge-base library. While building end-user applications, the knowledge base and library functions can be called directly from command language expressions. The language compiler manages the passing of bindings, whether to the C++ library or to other knowledge library functions. High-level language, along with the facility for graphical construction, can be used to depict the status of important events or actions. With interactive graphics, the work-group users can see and directly manipulate images representing their business operations and decisions.

CONCLUDING REMARKS

This article described a distributed environment and a logical framework for performing concurrent engineering. Creating a three-layer logical framework for concurrent engineering has the following benefits:

- *Interoperability.* It provides the applications better interoperability in a multi-vendor environment.
- Protecting investments. It shields the investments in applications from being lost due to downstream variations in platform hardware and software configurations, such as the "winning brand name or standard of the month."
- *No danger of obsolescence.* It prevents the applications that are developed today from becoming the "legacy" of tomorrow.
- *Intelligent interface layer.* It allows quick generation of the applications, because development kits, or utilities are placed on the common service layer (intelligent interface layer) separated from the computing platform and shared across multiple applications.
- *Less programming.* It insulates the application developer from platform and network variations.

REFERENCES

1. D. S. Margolias and M. H. O'Connell, "Part 3—Implementation of a Concurrent Engineering Architecture," in *WESTEC'90*, March 1990, Paper No. MS90-205.
2. B. Prasad, *Concurrent Engineering Fundamentals: Integrated Product and Process Organization, Volume 1*, Prentice-Hall PTR, Inc., Upper Saddle River, NJ, 1996.
3. Pennell and Slusarczwk 1989.
4. B. Prasad, *Concurrent Engineering Fundamentals: Integrated Product Development, Volume 2*, Prentice-Hall PTR, Inc., Upper Saddle River, NJ, 1997.
5. U.S. Air Force, *ICAM Project Reports*, Air Force Wright Patterson Laboratory (AFWAL), Dayton, OH, 1982.

6. R. W. Yeomans, A. Choudry, and P. J. W. Ten Hagen (eds.), *Design Rules for a CIM System*, North-Holland, Elsevier Science, Amsterdam, 1985.
7. M. Hardwick, et al., "ROSE: A Database System for Concurrent Engineering Applications," in *Proceedings of the Second National Symposium on Concurrent Engineering*, Concurrent Engineering Research Center, Morgantown, WV, 1990, pp. 33–65.
8. J. W. Lewis, "An Approach to Applications Integration for Concurrent Engineering," in *Proceedings of the Second National Symposium on Concurrent Engineering*, Concurrent Engineering Research Center, Morgantown, WV, 1990, pp. 141–154.
9. M. Sapossnek, S. Talukdar, A. Elfes, S. Sedas, M. Eisenberger, and L. Hou, "Design Critics in the Computer-Aided Simultaneous Engineering (CASE) Project," in *Concurrent Product and Process Design*, Chao and Lu (eds.), ASME, New York, 1989, pp. 137–141; *Proceedings of the Winter Annual Meeting of the American Society of Mechanical Engineers (ASME)*, December 1989.
10. DICE Initiative in Concurrent Engineering-DARPA, "Red Book of Functional Specifications for the DICE Architecture," Technical Report, Concurrent Engineering Research Center, West Virginia University, Morgantown, WV (February 1989).
11. D. Navinchandra, M. S. Fox, and E. S. Gardner, "Constraint Management in Design Fusion," in *Concurrent Engineering: Methodology & Applications*, P. Gu and A. Kusiak (eds.), Elsevier Science, Amsterdam, 1993, pp. 1–30.
12. M. R. Cutkosky and J. M. Tenebaum, "Providing Computational Support for Concurrent Engineering," *Int. J. Syst. Autom.: Res. Applic.*, *1*(3), 239–261 (1991).

BIBLIOGRAPHY

I. Turino, Managing Concurrent Engineering, Van Nostrand Reinhold, New York, 1992.

U.S. Department of Defense, Draft Military Standard for Systems Engineering, MIL-STD-499B, Draft, 1992.

R. I. Winner, J. P. Pennell, H. E. Bertrend, and M. M. G. Slusarczuk, "The Role of Concurrent Engineering in Weapon Systems Acquisition," IDA Report R-338, Institute for Defense Analysis, Alexandria, VA, December 1988.

S. G. Shin, "Concurrent Engineering and Design for Manufacture of Electronic Products," Van Nostrant Reinhold, New York, 1991.

S. Finger, "Concurrent Engineering," Presentation Materials, ICED '93, The Hague, August, 1993.

K. J. Cleetus, "Definition of Concurrent Engineering," CERC Technical Report Series, Research Notes, CERC-TR-RN-92-003, Concurrent Engineering Research Center, West Virginia University, Morgantown, pp. 1–5, 1992.

D. L. Carter and B. S. Baker, Concurrent, The Product Development Environment for the 1000s, Addison-Wesley Publishing Company, Reading, MA, 1992.

J. Pennell and M. Slasarczuk, "An Annotated Reading List for Concurrent Engineering," Technical Report HQ 9-034130, Institute for Defense Analyses, Alexandria, VA, 1989.

SUBRA GANESAN

BIREN PRASAD