Applying a hierarchical multiprocessing architecture to computer integrated manufacturing
by Roen Stephen Hogg

A thesis submitted in partial fulfillment of the requirements for the degree 'of Master of Science in Computer Science
Montana State University

Abstract:
A hierarchical multiprocessing architecture is an organizational paradigm that can be applied to the class of problems that involve organizing asynchronous processes via abstraction and localization of information. Within the domain of Computer Integrated Manufacturing (CIM), one such problem is the integration of manufacturing constraints into the product design process. Though most companies perform some degree of design optimization, it is often achieved via an ad hoc system of people and computers. This thesis discusses the application of a hierarchical multiprocessing architecture to this ad hoc system in an attempt to provide the design engineer with timely estimates of the economic impact of various design alternatives. In particular, a hierarchical organizational system, based on Factor and Gelernter's [1988] process lattice, was developed and tested.

Applying a Hierarchical Multiprocessing Architecture to Computer Integrated Manufacturing

by

Roen Stephen Hogg

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

September 1989

APPROVAL

of a thesis submitted by

Roen Stephen Hogg

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

NOVEMBER 13, 1989
Date

Gary J. Hoskin
Chairperson, Graduate Committee

Approved for the Major Department

Nov. 30th, 89
Date

J. Dubigh Stanley
Head, Major Department

Approved for the College of Graduate Studies

December 7, 1989
Date

Henry L. Parsons
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made.

Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Rae Hogg

Date 18 Oct 1989

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

ABSTRACT

A hierarchical multiprocessing architecture is an organizational paradigm that can be applied to the class of problems that involve organizing asynchronous processes via abstraction and localization of information. Within the domain of Computer Integrated Manufacturing (CIM), one such problem is the integration of manufacturing constraints into the product design process. Though most companies perform some degree of design optimization, it is often achieved via an ad hoc system of people and computers. This thesis discusses the application of a hierarchical multiprocessing architecture to this ad hoc system in an attempt to provide the design engineer with timely estimates of the economic impact of various design alternatives. In particular, a hierarchical organizational system, based on Factor and Gelernter's [1988] process lattice, was developed and tested.

INTRODUCTION

A recent trend in Computer Integrated Manufacturing (CIM) research has been the development of process and data management systems. These systems are intended to manage the entire engineering release system of a product. Though such systems are useful, management of the product development process is not sufficient. A product is more cost effective when manufacturing constraints, such as material and machining cost, are considered while the product is being designed. Though most companies perform some degree of design optimization, it is often achieved via an ad hoc system of people and computers. This thesis discusses the development of a Constraint Monitor that provides the design engineer with timely estimates of the economic impact of various design alternatives by imposing an organizational paradigm on this ad hoc system. The driving force behind the design of the Constraint Monitor is the recognition that the problem domain, since it directly interacts with the physical world by the continual input of new information from a variety of sources, is an open system. Any design for an integrated product development system, therefore, needs to address the properties of open systems. A paradigm that is well-suited for an open system is Factor and Gelernter's [1988] process lattice since it provides a flexible method to organize the flow of information within a system that consists of a diverse set of functions. This hierarchical organizational paradigm is used to develop the Constraint Monitor.

Though the Constraint Monitor developed in this thesis does not demonstrate actual manufacturing constraints, it does illustrate how different constraints could be organized in a hierarchical fashion based on a process lattice. In order to apply this concept to an actual product development system, a thorough analysis of the different manufacturing constraints and their relationships must be performed.

This thesis first presents an overview of current CIM systems. Following this overview, the need for a Constraint Monitor that integrates manufacturing constraints into the design process is discussed. An argument is then made that a hierarchical multiprocessing architecture is a well-suited organizational paradigm for the development of a Constraint Monitor. This is followed by a specification of the particular architecture used to design the Constraint Monitor, namely the process lattice paradigm. This discussion of the design and implementation of the Constraint Monitor is then followed by an overview of related areas of future research.

COMPUTER INTEGRATED MANUFACTURING

In this chapter an overview of Computer Integrated Manufacturing (CIM) is presented followed by a brief review of several existing systems that provide data and process management. The functional components of these systems are abstracted and the limitations of such a functional design are discussed. A case is made for CIM systems that incorporate manufacturing constraints into the design process.

CIM Overview

Automation of the manufacturing process has been a research concern for a number of years. In 1974, Harrington defined "Computer Integrated Manufacturing" (CIM) in his book by the same name. Since then, the concept of what exactly constitutes a CIM system has been disputed. In particular, the terms "manufacturing" and "integration" remain controversial. This section discusses the various interpretations of these two terms and their relevance to this thesis.

Manufacturing

Manufacturing systems, as noted by Chang and Wysk [1985], can be classified into two major categories: continuous process manufacturing and discrete parts manufacturing. The former involves the production of a continuous product; for instance, the process of refining crude oil into gasoline.

The latter involves the production of a product that undergoes a finite number of production operations. CIM research has generally focused on this latter case. In particular, it has focused on the manufacturing of machined parts which is also the focus of this thesis.

Integration

As illustrated in Figure 1, integration can differ in terms of type and degree, which produces confusion regarding what constitutes an integrated system.

Integration		
Type	Control limited to actual production activities	Incorporation of preproduction activities
Degree	Loosely coupled	Tightly coupled

Figure 1. Types and Degrees of Integration.

Research in integrating the process of manufacturing machined parts can be delineated into two areas with respect to type: (1) the control of actual production activities, i.e. the actual manufacturing of the product, and (2) the incorporation and synchronization of preproduction activities such as product design and analysis. The former limits manufacturing in the strict sense of the word. Such a view is unnecessarily limiting; since other aspects of the product development process must be considered to achieve true integration.

The degree of integration is a continuum defined by the amount of coupling. For instance, a loosely coupled system (as illustrated in Figure 2) might encompass the interface and coordination of information that occurs throughout the entire organization.

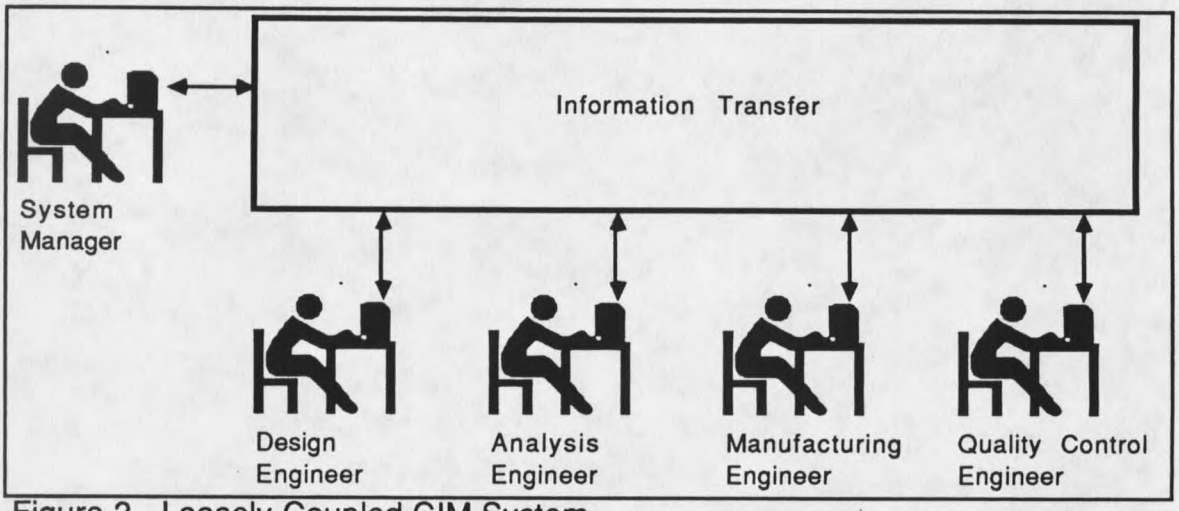

Figure 2. Loosely Coupled CIM System.

The solution requires integration of organizations that are currently "islands of automation". Though this solution has technical consequences, it is less a technical problem than political. As noted by Appleton [1985] and Savage [1985], policies need to be defined by upper management which will bring members of the engineering release system into a closer relationship.

A more tightly coupled system (Figure 3) extends this concept of information integration to include, for instance, the data and process management of the engineering release system of a product. Such a CIM system provides procedures, methodologies, and application programs to aid people in developing a product. This includes providing authorization control (controlling who can use the system) and process control (controlling when the

next process in the product development process can be executed). This ensures an effective and accountable product development process (also referred to in the literature as a product's life cycle).

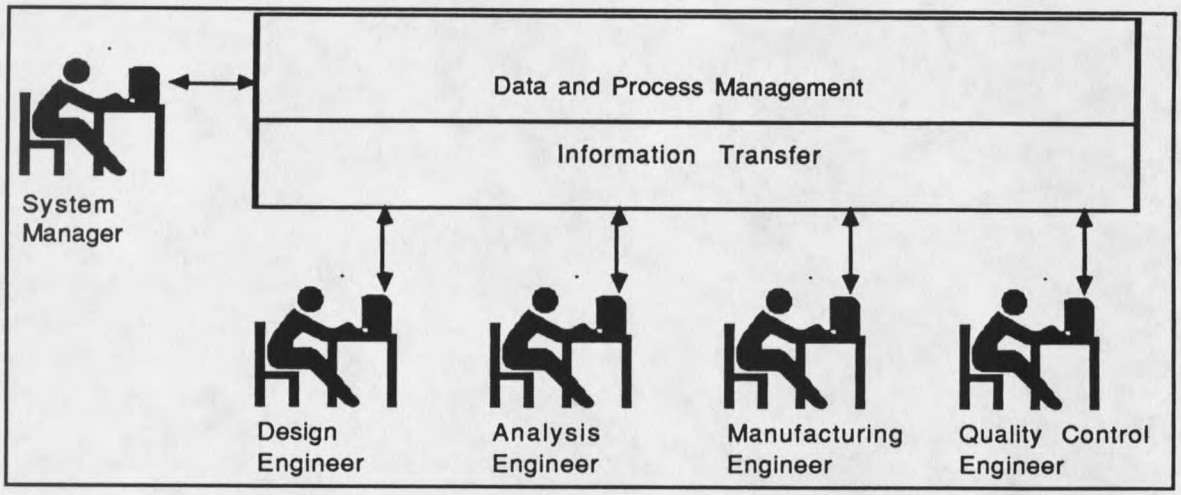

Figure 3. Tightly Coupled CIM System.

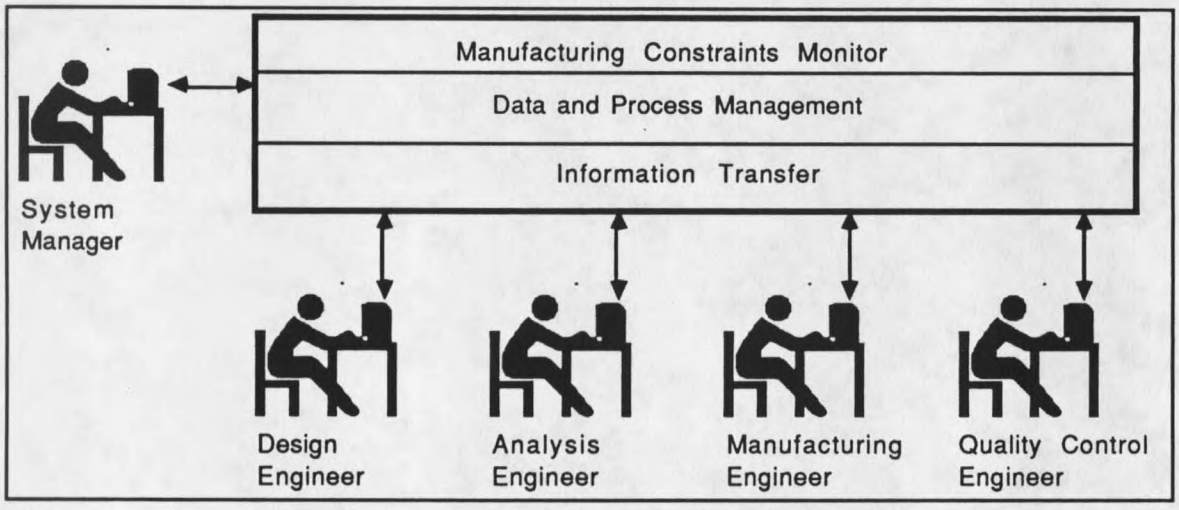

Figure 4. More Tightly Coupled CIM System.

A more tightly coupled system (Figure 4) extends this concept of data and process management to include, for instance, the integration of manufacturing constraints during the design process.

This research considers developing such a tightly coupled system. Methods are investigated to organize the preproduction tasks so that the product is manufactured at the optimal production rate and cost. Before this research approach is discussed, it will be helpful to describe current CIM data and process management systems.

Current CIM Data and Process Management Systems

Developing a CIM system that performs data and process management is not technically difficult. In fact, several such systems that provide some degree of data and process management currently exist: (1) Sherpa's DMS, (2) SDRC's (Structural Dynamics Research Corporation) DMCS, and (3) IBM's DCS. These three products can be categorized into two distinct groups: Those that offer an integrated system and those that offer a collection of software tools. Sherpa and SDRC are of the former, IBM of the latter. In other words, Sherpa and SDRC offer an integrated system that has predetermined data structures and predetermined functions that operate on this data. IBM offers a set of software tools which the user must integrate into a unified system.

There are advantages and disadvantages to each approach. The advantage of the integrated system approach is that it reduces the cost to install a CIM system. The disadvantage is that because it is an off-the-shelf system, it is difficult to tailor the system to match the user's particular needs and ways of doing business. The software tools approach is flexible and can be tailored to

the user's particular requirements. However, more time (and money) is required to integrate these flexible software tools into a unified system. Based upon this categorical distinction, the following sections examine in more detail the three products.

Sherpa DMS

Sherpa DMS is a software system that runs on the VAX series of computers. This system manages both data and the development process for engineering organizations. Once the user gathers the design process data, Sherpa offers a Macintosh-style interface that allows one to enter this data into DMS. Once this data is entered, the system provides several predetermined functions that facilitate management of the design process. In addition, DMS allows users to tailor these functions to match their particular needs and way of doing business. This is done by using DMS's alert facility to define alerts. The alerts can be used to notify users of database activity or to activate a computer process based on a database activity. The capability to define alerts enhances the flexibility of the system. For the alerts enable the user to customize the system to perform those actions deemed necessary. In addition to allowing the user to tailor the system functions to match his needs, DMS offers two other useful features: distributed file storage and distributed user access. Distributed file storage allows files to be stored on any node in the network, but still be controlled by DMS from a central location. Distributed user access provides users with transparent access to DMS from any node in the network.

SDRC DMCS

SDRC DMCS is in many ways similar to Sherpa DMS. DMCS, like DMS, is a software system that runs on the VAX series of computers. Once the user gathers and inputs the design process data, DMCS keeps track of product data creation, revision, and movement. DMCS, like DMS, supports distributed file storage. In fact, in DMCS, file storage is based on user rules that allow files to be stored on any node in the network. And like DMS, DMCS provides several predefined functions that facilitate management of the design process. However, unlike DMS, DMCS does not allow users to tailor these functions. That is, DMCS is an application, not an operating system. Thus it is designed to be used as delivered. Though the user can change data values, the user cannot add additional functions or modify existing functions. To do this, the application code itself must be modified. As such, DMCS does not offer the flexibility of DMS alerts.

IBM DCS

IBM DCS, unlike DMS and DMCS, is not a software system. Instead, IBM offers several software tools which the user would then integrate into a software system to support engineering design and manufacturing. However, this integration is only the first step since these tools must be tailored to meet the user's specific needs. In particular, the user must define and write ISPF panels, define data elements for DCS (Data Communication System) and for CDF (Consolidate Design File), write EXEC's to perform necessary functions for each panel, and implement user security.

Though these three systems provide some degree of data and process management, none of these systems is a turn-key system. Before such a system can be used, a company must determine the method by which it develops a product. This method should include the phases through which the product passes, the conditions and approvals that are required, and the personnel that are involved. Furthermore, a company must tailor any purchased CIM system to match its particular needs and ways of doing business.

Functional Description of CIM Data and Process Management Systems

Though the three products differ in particular details, functionally they all provide some degree of data and process management (in particular, they all support authorization control, process control, life-cycle definition, and message notification). In Figure 5 these CIM data and process management systems are functionally abstracted.

In this CIM model, User Control Knowledge is used to verify each requested user action. If the user has authorization to perform the action, this request is transferred to the Action Controller which sequences all requested actions. This list of sequenced actions is then sent to an Execute Action module which executes each requested action. In addition to user requested actions, there are system initiated actions. The Determine CIM System Action module uses CIM System Control Knowledge to determine whether any CIM action should be initiated based on information in the Product Definition Data Base. Such system initiated actions include the automatic notification of individuals based on an event within the CIM system as well as automatic execution of

processes defined within the CIM system based on a set of conditions, such as the automatic promotion of the product through its life cycle.

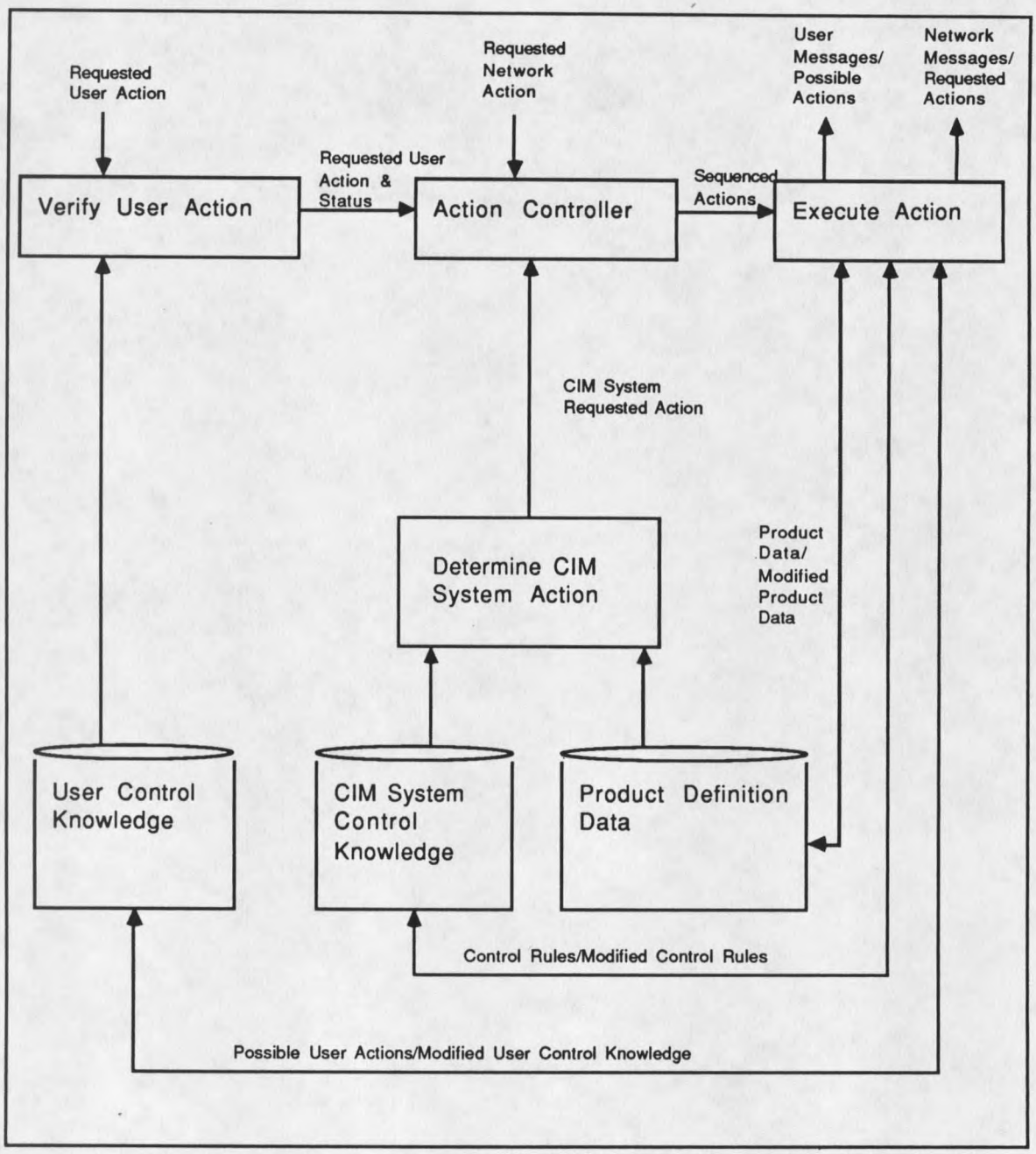

Figure 5. CIM Data and Process Management System.

Each of these functions is discussed in more detail in the appendix. In particular, the appendix describes the inputs, outputs, and driving requirements for each function.

Though these functions enable the system to provide data and process management, they do not integrate manufacturing constraints into the design process. The following section examines the need for such constraint integration.

Limitations of CIM Data and Process Management Systems

Though the three CIM data and process management systems described above support product development management, none incorporates manufacturing constraints into the design process. The incorporation of such constraints -- as noted by Fleischer and Khoshnevis [1986], Compton and Gjostein [1986], Suri [1988], Tipnis [1988], and Whitney et al. [1988] -- can result in substantial savings in product development cost. Such savings can be obtained since the design intent is often achievable in various ways. For instance, material type and tolerances can often be modified without impacting the design intent.

To achieve savings in the development of a product, the design engineer needs to be aware of the relationship between his design and the manufacturing process. This includes information on manufacturing constraints such as material cost, material attributes, and the production qualities of different machining processes.

The goal of this thesis is to propose an architecture that supports a formal interaction between design and manufacturing such that manufacturing

constraints are considered during the design process. In particular, this thesis discusses the development of a Constraint Monitor, as illustrated in Figure 6.

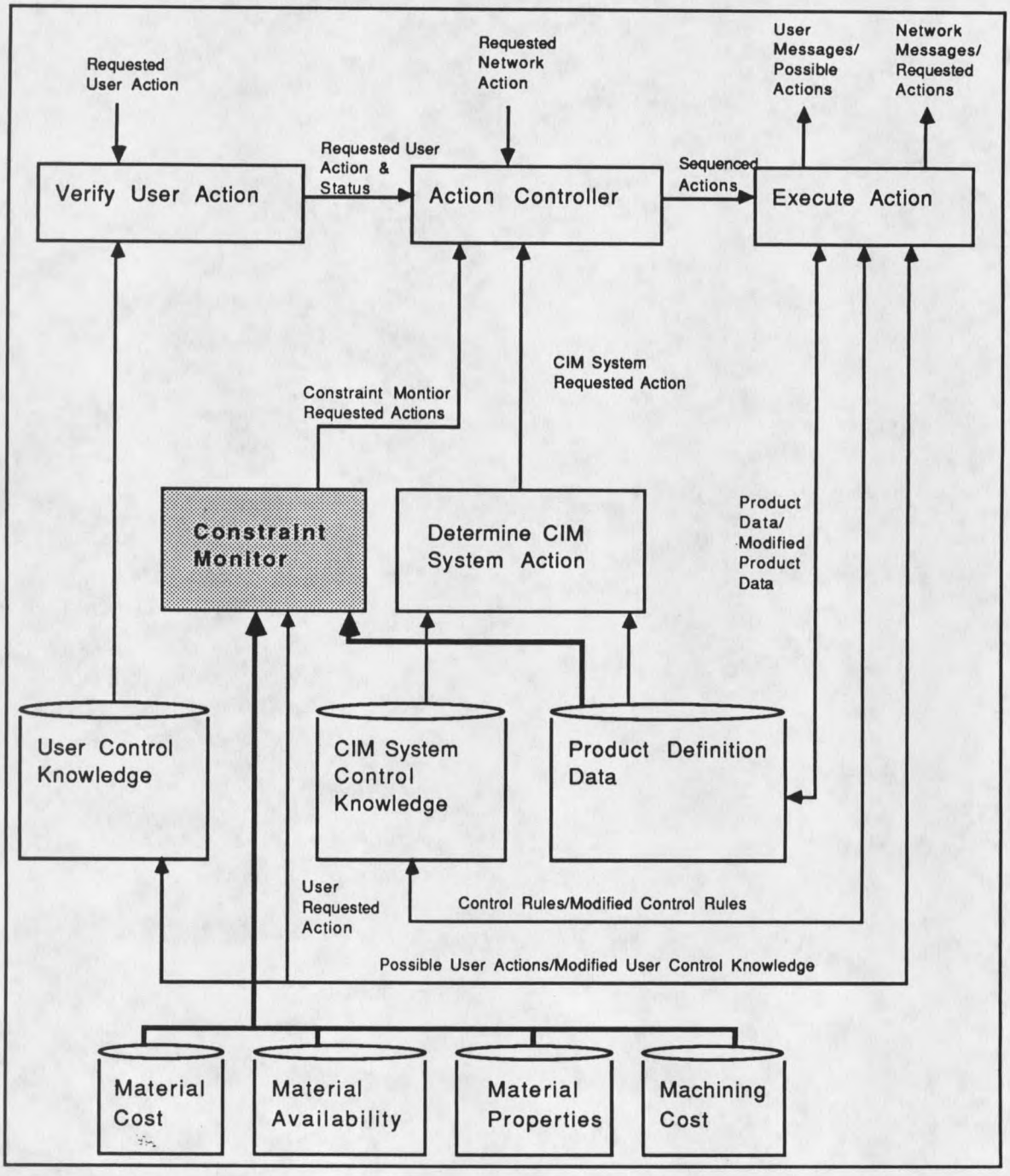

Figure 6. CIM Constraint Integration System.

This Constraint Monitor would provide, based on the state of the product definition data as well as the state of material and machining data, constraint information to the design engineer. For instance, it could monitor a material-type field in the product definition data base. Once this field is specified it could generate a message and send it to the manufacturing engineer for a material cost analysis. Or the design engineer could issue a query to the Constraint Monitor, before he specifies a particular material, asking for a cost analysis of several different materials. The Constraint Monitor would then automatically route the query to those relevant material cost analysis functions. These functions could be either analysis programs, expert systems, or human experts.

In summary, the Constraint Monitor formalizes the interaction between the various functions involved in developing a product. Such an organizational paradigm would be useful in reducing product development cost. The following chapter discusses a possible design approach for the Constraint Monitor.

HIERARCHICAL MULTIPROCESSING ARCHITECTURES

The previous chapter discussed the need for a Constraint Monitor that integrates manufacturing constraints into the design process. This chapter outlines a possible design approach for the Constraint Monitor. Since a product development system that directly interacts with the physical world by the continual input of new information is an open system, this chapter first enumerates the properties of open systems. Then it proposes that a hierarchical multiprocessing architecture is a well-suited organizational paradigm for the open system discussed in this thesis, namely the development of a Constraint Monitor. General properties of such hierarchical architectures are then mentioned. This is followed by a specification of the particular hierarchical architecture used to design a prototype Constraint Monitor, namely the process lattice paradigm. The following chapter then discusses the implementation of a prototype Constraint Monitor.

Open Systems

Hewitt [1986] posits several fundamental characteristics of open systems. The characteristics most pertinent to the development of a Constraint Monitor concern: (1) concurrency, (2) asynchrony, and (3) decentralized control. Other characteristics, such as the problems of inconsistent information and the need for continuous operation, will be addressed in the chapter on related research areas.

Concurrency arises since open systems consist of numerous components that must process information concurrently since each component can simultaneously receive new information. In terms of developing a Constraint Monitor, an organizational scheme is needed that integrates a variety of components that concurrently monitor manufacturing constraints -- whether based on dynamic data such as material and machining cost, or based on static data such as the results of material stress analysis functions.

As for asynchrony, Hewitt notes it occurs in open systems in two ways. First, new information may enter the system at any time, requiring it to operate asynchronously. Second, the components are physically separated. This prevents them from acting synchronously in an efficient manner. Both of these cases are true for a product development system that integrates manufacturing constraints into the design process. New information regarding material cost, inventory, and so forth could enter the system at any time. Furthermore, the results of manufacturing constraint analysis -- whether performed by a human expert, an expert system, or traditional analysis program -- could at any time enter the system. Moreover, the different analysis components in such a system are physically separated. Indeed, some of these components most likely will be human analysis engineers. Subsequently, an organizational scheme is needed that is able to integrate these asynchronous processes.

As for decentralized control, it arises in an open system since a centralized decision maker would become a bottleneck. Furthermore, Hewitt notes that a centralized decision maker could never have complete and up-to-date information on the state of the system due to communication asynchrony and unreliability. Though a distributed system will not provide up-to-date data

at every node, such a system naturally supports localizing decisions to where they are needed in the system.

The process lattice paradigm, discussed later in this chapter, provides such a distributed architecture. Furthermore, it naturally supports concurrency and asynchrony. Thus it is an architecture worth investigating as a organizational paradigm for a Constraint Monitor. In essence, the process lattice is a hierarchical multiprocessing architecture. The following section discusses general characteristics of such architectures.

Hierarchical Multiprocessing Architectures

A hierarchical multiprocessing architecture provides an organizational scheme that is well suited for an open system. This section outlines general characteristics of such architectures and shows how these traits naturally support concurrency, asynchrony, and decentralized control.

This section first describes the different control and data mechanisms of such architectures. Of the different mechanisms, the combination used for the Constraint Monitor is of the data flow type. The salient characteristics of such a network topology are then summarized. Following this overview, the particular multiprocessing architecture used for the Constraint Monitor, namely the process lattice paradigm, is discussed.

Control and Data Mechanism

Treleaven [1982] notes that there are two mechanisms fundamental to any model of computation: (1) the control mechanism and (2) the data mechanism.

Control Mechanism	Data Mechanism: by value	Data Mechanism: by reference
by availability	data flow	control flow
by need	string reduction	graph reduction

Figure 7. Model of Computation. [Treleaven, 1982]

As for the control mechanism, it defines how one function causes the execution of another. Execution can be caused in two ways. First, it can be caused by availability. In this case the control mechanism signals an argument's availability and a function executes when all the arguments are available. Second, execution can be caused by need. In this case the control mechanism signals the need for an argument and a function executes when one of its output arguments is required by the invoking computation.

As for the data mechanism, it defines how a particular argument is used by a group of functions. The data mechanism can be defined in two ways. First, it can be defined by value. In this case an argument is shared by giving a separate copy to each accessing computation. Second, the data mechanism can be defined by reference. In this case an argument is shared by having a reference to it stored in each accessing computation.

A control mechanism by availability and a data mechanism by value (as in the data flow type) is well suited for the Constraint Monitor which, as discussed in the previous chapter, needs to support asynchronous communication and distributed control.

Numerous variances of this architecture exist. For instance, this architecture is used in a slightly modified form in neural networks. [Hecht-Nielsen, 1988] It is also used in Petri Nets [Sunshine, 1982; Peterson, 1977], and in the Graph Model of Behavior (GMB) [Razouk and Estrin, 1982]. The particular architecture used to define the Constraint Monitor is based on the process lattice concept developed by Factor and Gelernter [1988]. But regardless of the particular variances, such architectures share a similar network topology, as discussed in the following section.

Network Topology

A multiprocessing architecture can be depicted by a graph in which nodes represent processing elements and arcs represent communication links. In general, each processing element can have many input values but usually is defined as having only one output value. The output value, however, can be distributed along many pathways to provide input to other processing elements. These pathways, as illustrated in Figure 8, connect the processing elements into a network.

Each processing element's output value is defined as a function of its input values. In the most general sense, how this function is implemented is irrelevant. The function itself could be executed by an algorithm, an expert system, or even a human expert. The only constraint is that the function be self-

contained and that it transform the given input values into an output value. Furthermore, there is not necessarily any limitation to the functional abstraction of these functions. Indeed, these functions are often defined hierarchically such that processing elements situated at higher levels in the network perform correspondingly more abstract data transformations.

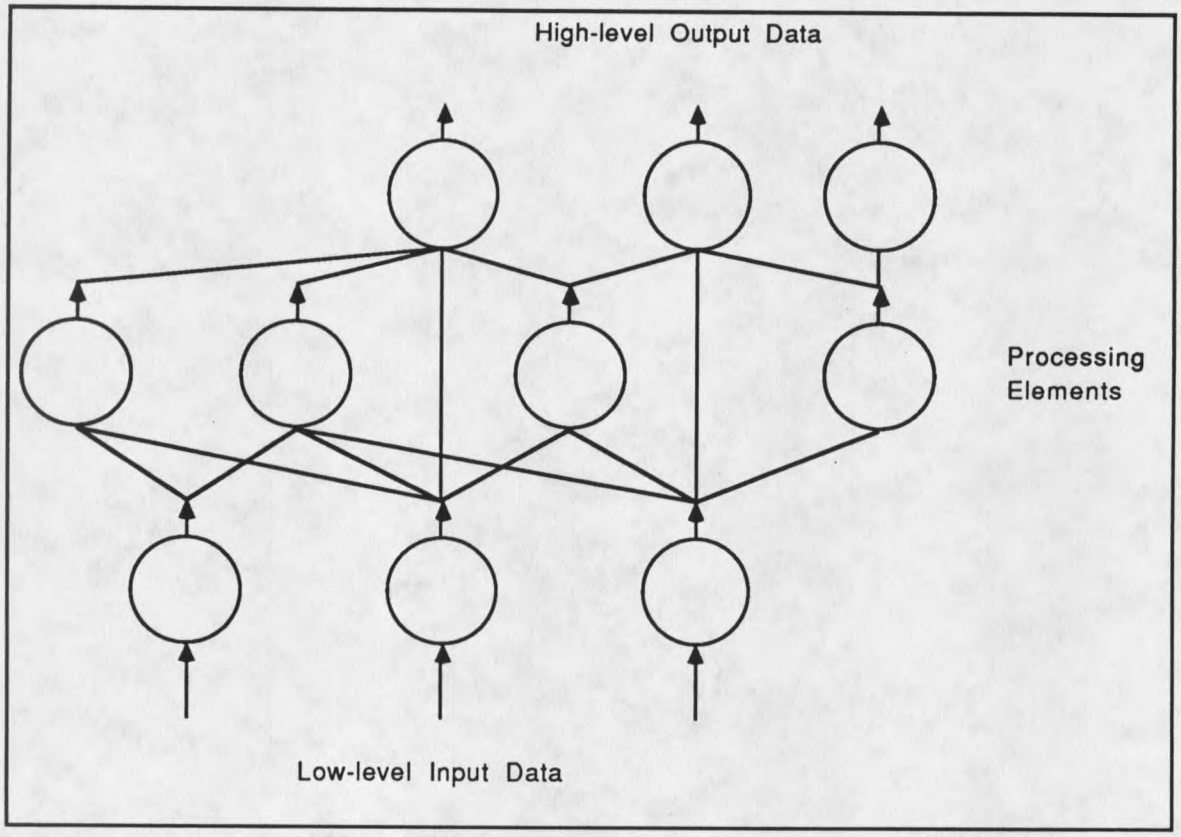

Figure 8. Network Topology.

This is the underlying concept of the process lattice paradigm. It organizes the numerous processing elements into a hierarchy and then restricts how these elements can interact. The following section discusses this paradigm in more detail.

Process Lattice Paradigm

The process lattice paradigm was developed by Factor and Gelernter [1988]. It is a network of concurrent processes arranged according to a simple organizational principle. In particular, the process lattice is arranged as a series of ranks or layers.

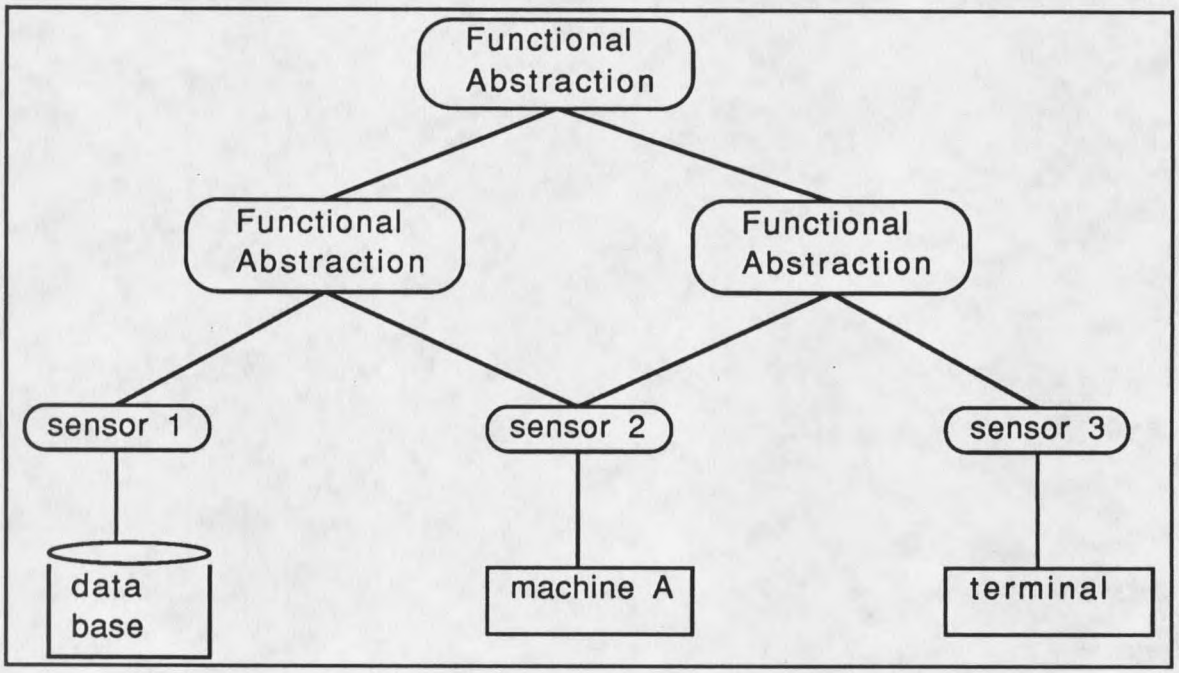

Figure 9. Process Lattice Structure.

Figure 9 is an example of a process lattice structure. Each node in the bottom rank corresponds to some external data source. Nodes in higher ranks correspond to increasingly abstract or general decision procedures. Data values flow upward through the lattice, and higher-level nodes respond to the data by changing their own states. In particular, higher-level nodes perform

some data transformation based on their inputs from the lower-level nodes. This process lattice framework allows many different functional methodologies to coexist since each node can support a unique functional paradigm. As such, this lattice paradigm is well suited for CIM with its multifarious constraint functions.

Factor and Gelernter refer to this structure as a "lattice" because each bottom-rank node corresponds to some primitive element and each higher-rank node corresponds to some set of primitive elements. The process lattice can be more precisely defined as follows:

> The lattice contains a collection of nodes, and each one defines a mapping from a set of input states (the states of the inferior nodes) to an output state (its own state). We posit that a node's state always reflect [sic] (or be in transition to) the value yielded by applying its own state function to the current values of its input states and its own state. It follows that, whenever some value changes, the effect of the change ripples upwards through the lattice. A node's state may [sic] "undefined" as well, or may be "pending", which is an intermediate condition; each node's state function determines explicitly when the local state value is defined, undefined and pending. Concretely, a value will be defined when "enough" inferior values are defined to allow the state function to be computed. Nodes with the null state function (depending on no inferiors, relying on externally-supplied values) have [sic] "undefined" state in the absence of an external data signal. [Factor and Gelernter, 1988, p. 3]

Factor and Gelernter define two types of "logic probe" that operate in opposite directions on the lattice: an "inject value" probe and a "read value" probe. An inject-value probe sets the state of any selected node to any value. Nodes at the bottom of the lattice have a permanently-attached inject-value

probe which inputs new values into the system. A read-value probe reads any node's current state. When the current state of the queried node is undefined, the read-value query propagates downward to each of its inferiors. When new data values are returned, the current state is computed and the read-value query is completed (see Figure 10).

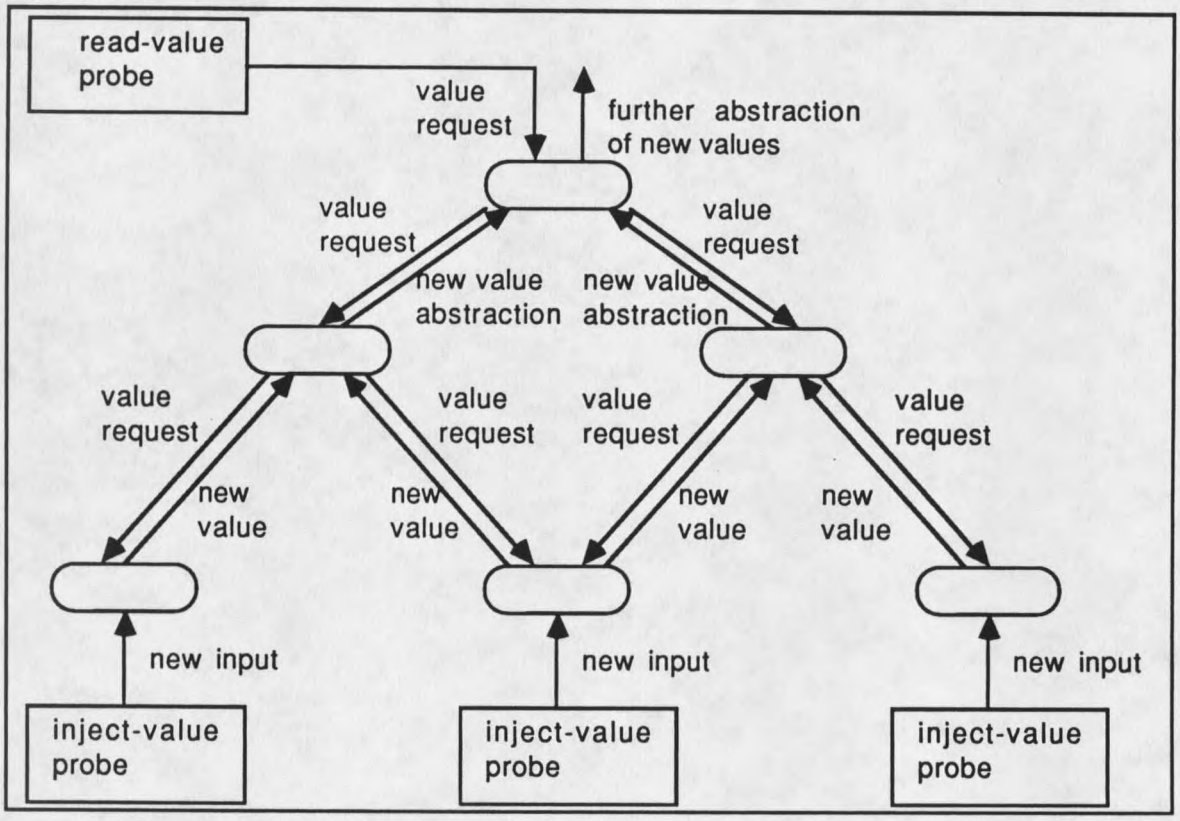

Figure 10. Process Lattice Flow of Control and Data.

The process lattice shares two characteristics of a data flow graph. First, the process lattice does not introduce sequencing constraints other than the ones imposed by data dependencies specified in the internode relationships. Second, it is directly translatable into a graph whose nodes represent functions and whose internode relationships represent a data path. In this case, nodes

group functional information into one conceptual unit while internode relationships define a multitude of dependencies between nodes.

As noted by Factor and Gelernter the process-lattice structure is attractive for several reasons: (1) it represents a flexible organizing structure superimposed on a diverse set of functions, (2) it provides a simple and comprehensible organization of the overall system and information-flow within the system, (3) it preserves a simple mapping between the logical structure of the domain and the structure of the system, and (4) it naturally supports parallelism and "bi-directional" operations.

The hierarchical multiprocessing architecture for the Constraint Monitor is based on this process lattice. The following chapter discusses the implementation of the Constraint Monitor using the process lattice paradigm.

A HIERARCHICAL ORGANIZATIONAL PARADIGM APPLIED TO CIM

This chapter describes an application of a hierarchical multiprocessing architecture to CIM. In particular, it discusses the implementation of a Constraint Monitor that notifies the design engineer of manufacturing constraints during the design process. In order to demonstrate the concept of the Constraint Monitor, a Design Assistant system was developed which incorporates not only the Constraint Monitor, but related subsystems and data bases. As illustrated in Figure 11, the Design Assistant sends user generated product design data to an asynchronous process lattice. Constraint analysis is then performed by the lattice and constraint information is sent to the Constraint Monitor. This lattice, as described in the previous chapter, is a collection of processing nodes connected by a unified communication protocol. Furthermore, this lattice is functionally hierarchical. That is, functions associated with higher-level nodes correspond to increasingly more abstract data transformation functions.

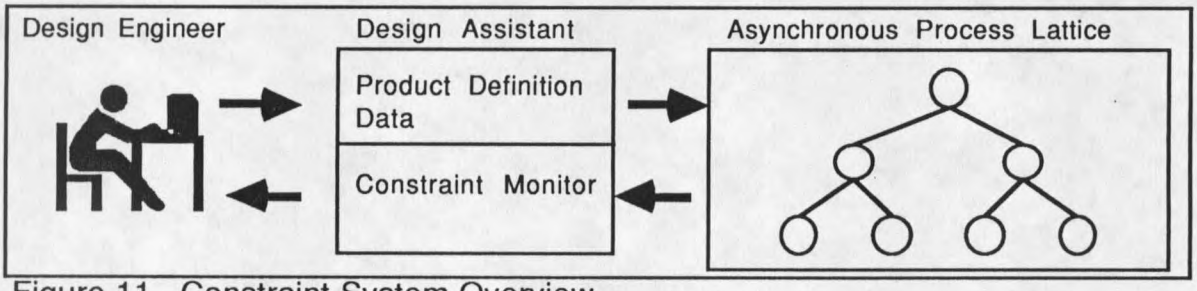

Figure 11. Constraint System Overview.

The prototype Design Assistant system was written using HyperCard on the Macintosh. This object-oriented programming language provides an environment for rapid development of the system and is well suited for designing the Design Assistant user interface as well as implementing and simulating to some degree the process lattice. In particular, this lattice is implemented as a collection of interrelated cards. In other words, each node in the lattice is represented as a card object in HyperCard. Input and output ports are represented by links between cards. A hierarchical multiprocessing architecture is simulated via HyperTalk, a subsection of HyperCard. This simulator sends a message to each node to execute its particular data transformation function. These functions transform the node's input values into an output value and then route this value to all specified nodes in the lattice.

This chapter elaborates on this cursory overview of the Design Assistant system. First the implementation language (i.e. HyperCard/HyperTalk) as well as the implementation of the process lattice simulator is mentioned. Then the Design Assistant system is discussed in more detail. In short, this chapter consists of the following sections: (1) Implementation Language, (2) Process Lattice Simulator, (3) Design Assistant System Design.

But first, however, a caveat is in order. The prototype Constraint Monitor discussed in this chapter is intended to demonstrate proof of concept. It is not intended to demonstrate actual manufacturing constraints but instead to illustrate how different constraints could be organized in a hierarchical fashion based on a process lattice paradigm. In order to apply this concept to an actual product development system, a thorough analysis of the different manufacturing constraints and their relationships must be performed.

Implementation Language

HyperCard was used to develop the Design Assistant since it is well-suited for designing the user interface as well as implementing and simulating the process lattice. The implementation of the process lattice is discussed in the following section. This section discusses not only HyperCard, but issues pertaining to object oriented programming languages that assist in the development of highly interactive graphical user interfaces.

In such a user interface, information communicated between the user and the system is presented in a graphical format. Moreover, information is not only outputted graphically, but inputted graphically as well. These graphical user interfaces enhance the user's ability to understand data and relationships between data. Programs that assist in the development of these graphical user interfaces are motivated in part by problems that plague traditional user interface development: the user interfaces are time consuming to build and are not easily modified or reused.

Object-oriented programs, such as HyperCard, attempt to resolve these problems by enabling the programmer to avoid programming when possible. Instead of programming the interface directly, HyperCard provides a kit of useful building blocks which can be assembled in flexible ways to create a particular interface. HyperCard, however, is more that just a user interface development tool. It also consists of an object-oriented programming language called HyperTalk. In essence, each graphical object in HyperCard (viz. a button, a

field, a card, a background, or a stack) can have an associated script. HyperTalk is the language used to create and modify these scripts.

These scripts were used to implement the Design Assistant. In particular, they execute all the user initiated commands as well as the process lattice simulator. The particular user commands are discussed later in this chapter. The following section discusses in more detail the use of HyperTalk message handlers to implement the process lattice simulator.

Process Lattice Simulator

This sections consists of two parts. The first part describes the particular implementation of the process lattice simulator used in the prototype Design Assistant to incorporate manufacturing constraints into the design process. The second part discusses more general design issues related to message-passing systems. As mentioned previously, this lattice consists of interrelated nodes that communicate via message passing.

Implementation of the Simulator

The parallel process simulator automatically executes the data transformation function associated with each node in the parallel process lattice when any change is made to the product data, e.g. the product's size or material type is modified. The simulator executes a node in the lattice by sending it a message to execute its associated function. This function performs some data transformation on the input values and then sends the output value to its associated nodes. To ensure data consistency, execution messages are sent to

nodes in a hierarchical fashion, i.e. those nodes at the bottom of the lattice are executed first. Figure 12 is the HyperTalk script for the simulator.

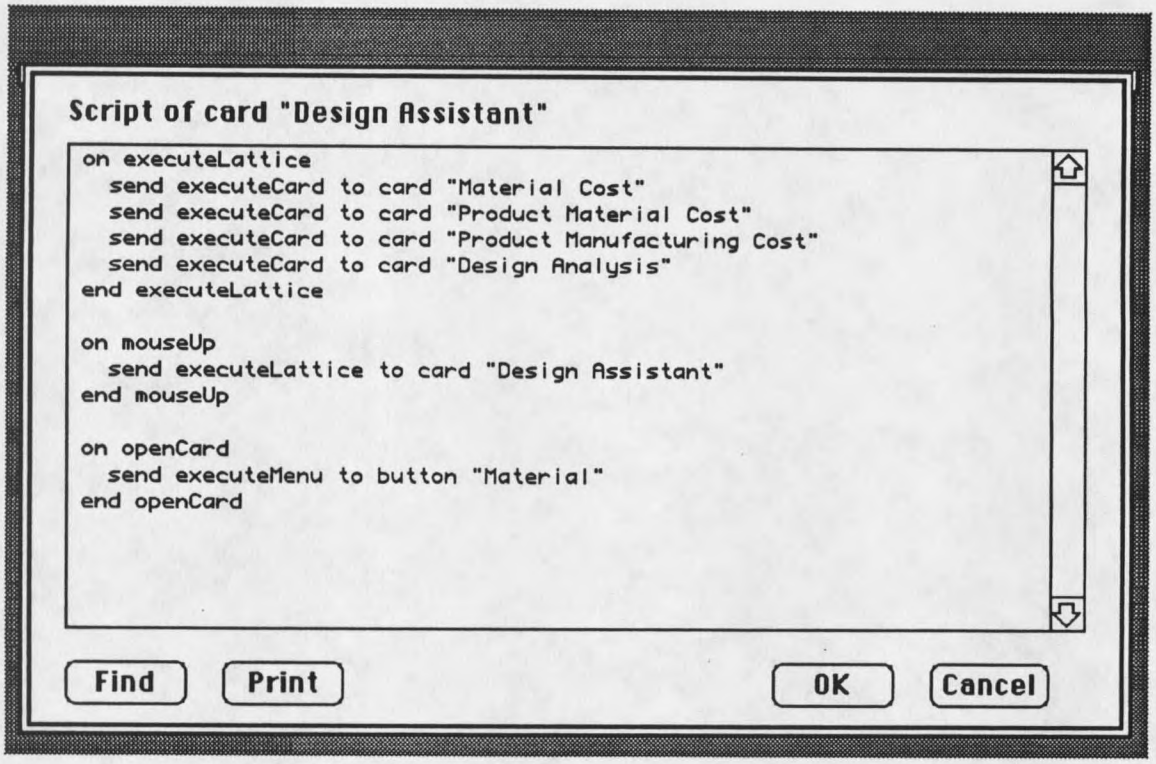

Figure 12. HyperTalk Script for Simulator.

Each node in the lattice has a unique *executeCard* function. The only constraint on this function is that its inputs are restricted to its input ports, and it must send its output to its output ports. Figure 13 illustrates a simple executeCard function for the node Product Material Cost.

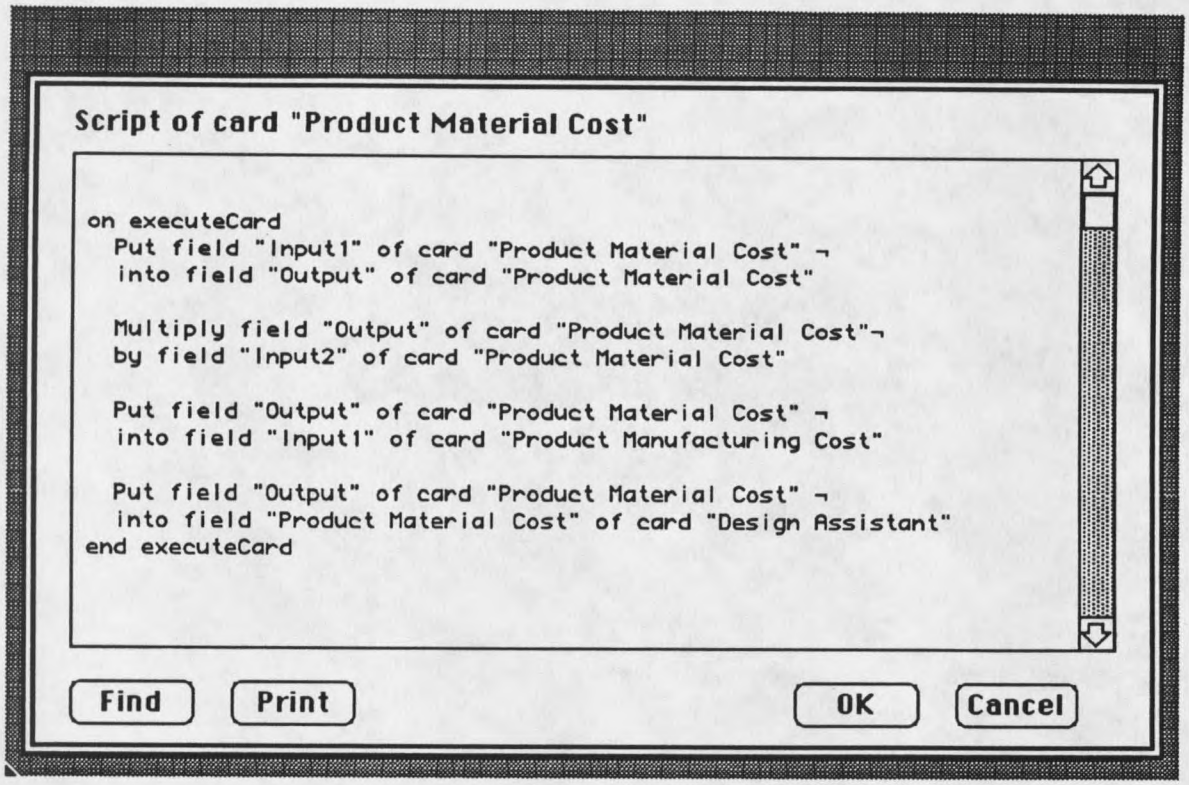

Figure 13. Function for Product Material Cost Node.

Message-Passing Systems

This section examines some of the salient traits of message- passing systems and discusses their relationship to the process lattice simulator used by the Constraint Monitor.

The hierarchical multiprocessing architecture for the Constraint Monitor is based on the process lattice paradigm developed by Factor and Gelernter [1988]. However, the Constraint Monitor's architecture differs with respect to interprocess communication. The Constraint Monitor uses a message-passing scheme to communicate between processes. The developers of the process lattice, on the other hand, developed their prototype using the parallel language Linda which is based on the tuple space model of parallel programming.

Instead of passing a message, a process communicates with other processes by generating a new data object (called a tuple) which is accessible to all other processes [Factor and Gelernter 1988, Carriero and Gelernter 1989].

The Constraint Monitor uses message-based communication between nodes because its problem domain, viz. the integration of manufacturing constraints into the design process, is more suited for a distributed system rather than a single parallel computer. Factor and Gelernter used the lattice paradigm to develop a prototype system for use in a post-operative cardiac Intensive Care Unit (ICU). In the case of the ICU prototype, a single parallel computer is well suited for this application since the problem domain -- the monitoring of the health of a single patient -- is physically localized. The Constraint Monitor must monitor a mechanical product as well as monitor manufacturing constraint information that is physically decentralized. Unlike the case of the human patient, the necessary information resides most often in a complex system consisting of numerous interconnected components of hardware and software as well as human analysis experts. A message-passing system for internode communication was selected since it facilitates the support of such a distributed product development system.

All message systems, as noted by Whiddett [1987], provide at least two basic operations of the general form:

(1) **send** <message> {**to** <destination>}

(2) **receive** <message> {**from** <source>} {**about** <subject>}

Multiple processes in the system use these two operations to communicate by sending and receiving messages. However, message- passing systems, as discussed by Quinn [1987] differ in three respects: First, whether they utilize

direct versus global naming; second, whether they utilize static versus dynamic links; and third, whether they are synchronous or asynchronous. The message-passing system used for the Constraint Monitor, as illustrated in Figure 14, utilizes direct naming and static links, and is asynchronous.

Message-Passing Approaches		
1	**Direct Naming**	Global Naming
2	**Static Links**	Dynamic Links
3	Synchronous	**Asynchronous**

Figure 14. Message-Passing Approaches.

In a direct naming scheme, the source and destination designators are the names of processes. In a global naming scheme, global variables or mailboxes are used for interprocess communication. The Constraint Monitor's process lattice simulator uses the first approach since such a system avoids the need for global variables which are a potential single-point of failure within a distributed system.

Second, message-passing systems differ in whether they use static or dynamic links. One approach is to have static links between nodes. In this case, the links between nodes are fixed at compile time. The second approach is to have dynamic links between nodes. In this case, the specification of the links between nodes is delayed until run time. The Constraint Monitor, as designed, uses the first approach. Though this limits the flexibility of the system,

such a limitation is not detrimental since static links are sufficient to demonstrate proof of concept for this prototype system. There is nothing inherent in the design, however, that restricts message passing to static links.

Third, message-passing systems differ in whether they are synchronous or asynchronous. Synchronous message passing will delay further execution of the sending process until its message is received. Asynchronous message passing has no such waiting protocols. Asynchronous message passing allows the sending process to get arbitrarily far ahead of the receiving process. The Constraint Monitor's design is based on asynchronous message passing; for it needs to monitor real-time constraint data such as material cost and availability. Subsequently, once a message is sent the sending process is unrestricted in its execution.

Design Assistant System Design

The Design Assistant is a system that supports product analysis and design methods. It consists of the integration of some product definition tool, such as a CAD tool, with a Constraint Monitor. This Constraint Monitor, as mentioned in the previous chapters, assists the design engineer in incorporating manufacturing constraints into the design process. The Design Assistant represents one possible use of the Constraint Monitor. In this case, relevant product design changes result in automatic constraint analysis. The results of this analysis, available at any level of abstraction, are displayed to the design engineer. The Constraint Monitor, however, could be used in other ways. For instance, instead of sending all results to the design engineer, results

could be sent to appropriate product design reviewers (whether human experts, expert systems, or traditional analysis functions) as well as to a centralized data base that records the development history of the product. In short, the hierarchical multiprocessing architecture of the Constraint Monitor allows for a flexible system.

A system that resembles the Design Assistant is the Analyst [Stephens and Whitehead, 1986]. Though this system also provides analysis and design support, it differs in the type of support. The Analyst is a collection of expert systems that perform some degree of product analysis. The Design Assistant, on the other hand, is a research approach to organize product analysis functions irrespective of the particulars of any analysis method. Indeed, as noted by the developers of Analyst, there is a need to incorporate a diverse range of analysis methods since many if not most of the products of analysis and design are documents which are not directly generated by some function. The process lattice paradigm used by the Design Assistant allows for the integration of such diverse functional methodologies.

The Design Assistant, as illustrated in Figure 15, consists of the following components: (1) Design Assistant user interface, (2) Constraint Monitor, (3) Process Lattice, (4) Material Cost data base, (5) Computer Aided Design (CAD) system, (6) Product Definition data base, and (7) Design Recommendations data base.

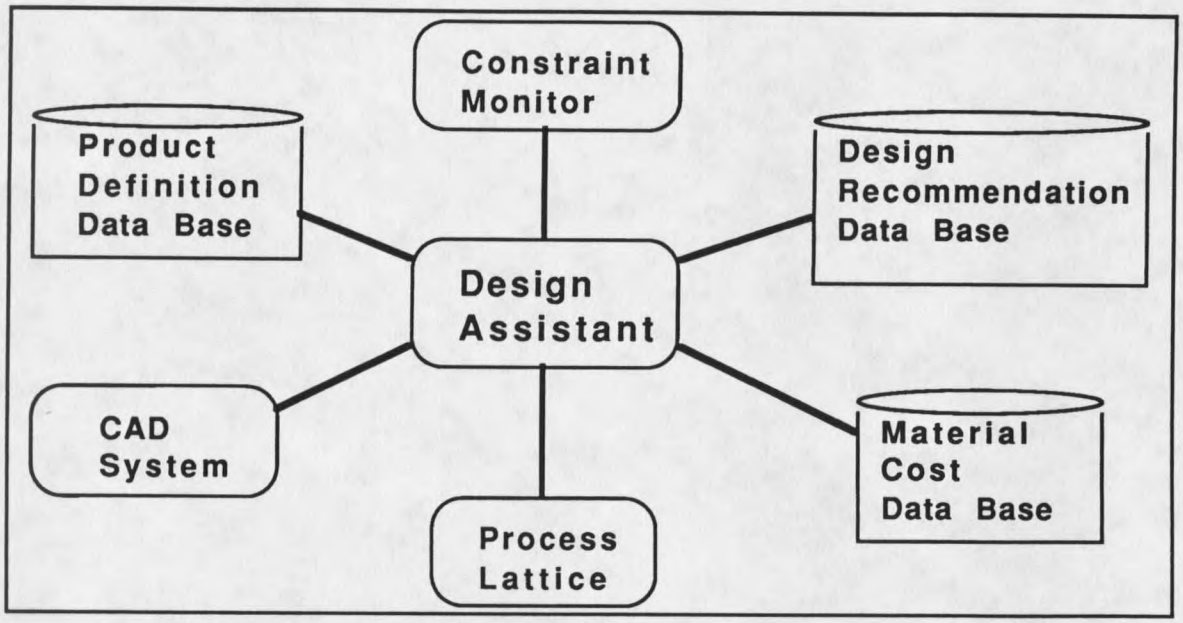

Figure 15. Design Assistant Components.

The following sections discuss in detail each of these system components.

Design Assistant User Interface

The Design Assistant user interface (see Figure 16) allows the design engineer to receive real-time feedback on manufacturing constraints pertaining to the product design.

This user interface consists of three major sections: (1) the header, (2) the product definition data, and (3) the Constraint Monitor. The header section contains, in addition to the name of the system, icon buttons that allow the user to move between various components of the system. Buttons in the Design Assistant user interface provide the mechanism to perform a variety of functions. Figure 17 describes the functionality of the different buttons.

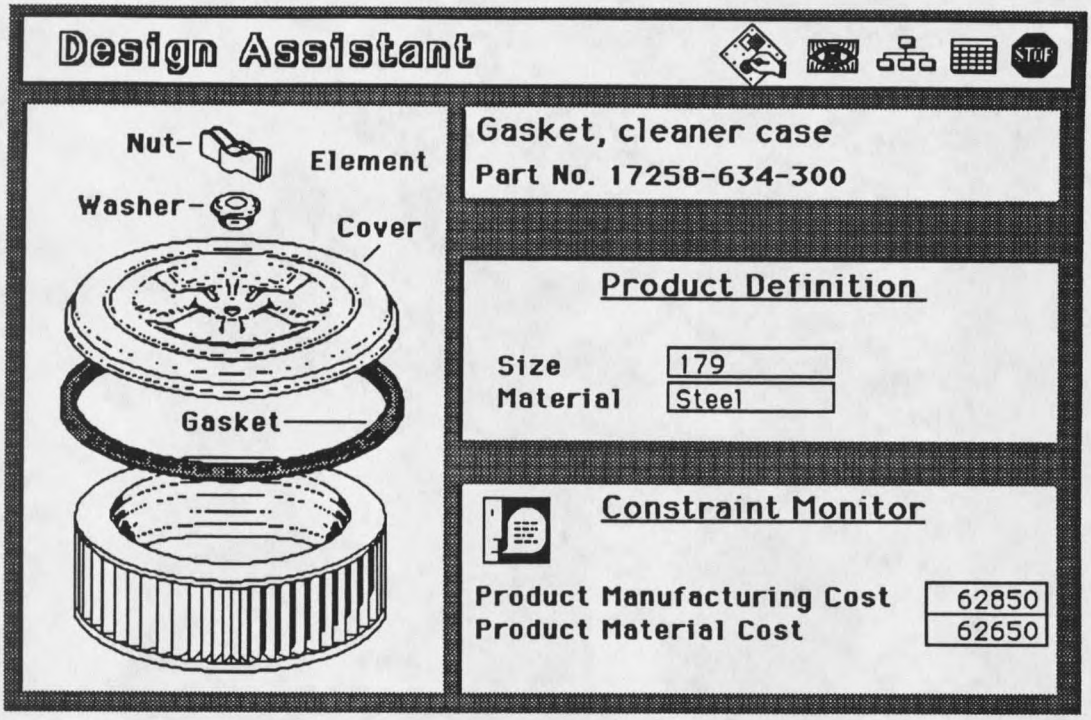

Figure 16. Design Assistant User Interface.

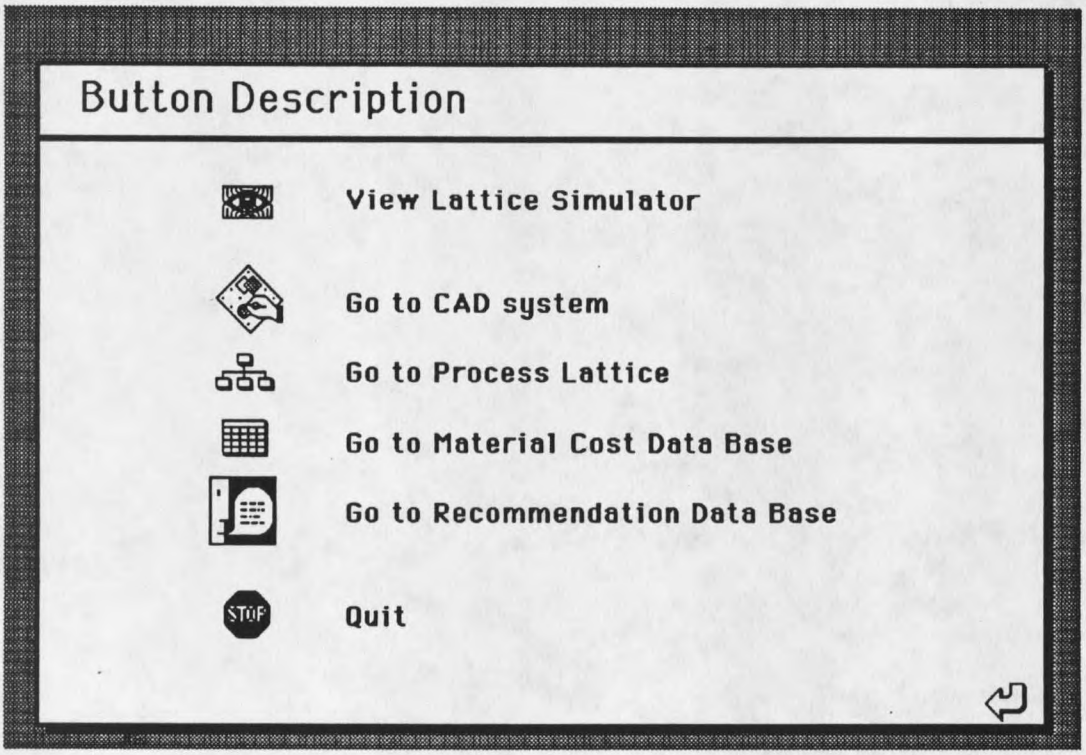

Figure 17. Functional Description of Buttons.

The product definition data section contains user-entered product information. In this prototype, these values are entered directly by the user. In a real CIM environment, these values would be generated by some CAD system. The Constraint Monitor section displays the results of constraint analyses based on constraint information represented in the lattice.

Constraint Monitor

The functionality of the Constraint Monitor was discussed in the previous chapter. In essence, it notifies the design engineer of relevant manufacturing constraints. In terms of the prototype Design Assistant, the Constraint Monitor monitors two cost constraints: product manufacturing cost and product material cost. These two constraints were selected solely for demonstrational purposes. Furthermore, the Constraint Monitor allows the design engineer to query any manufacturing constraint recommendations, whether they are generated by an expert system or a human expert.

Process Lattice

Figure 18 illustrates the lattice used in the prototype. The lattice consists of functions arranged in a hierarchy. For instance, the function for the node *Product Manufacturing Costs* is dependent upon input from more primitive functions associated with the nodes *Product Machining Cost* and *Product Material Cost.* Each node in this lattice is mouse sensitive. When a particular node is selected, a detailed description, as illustrated in Figure 19, is displayed.

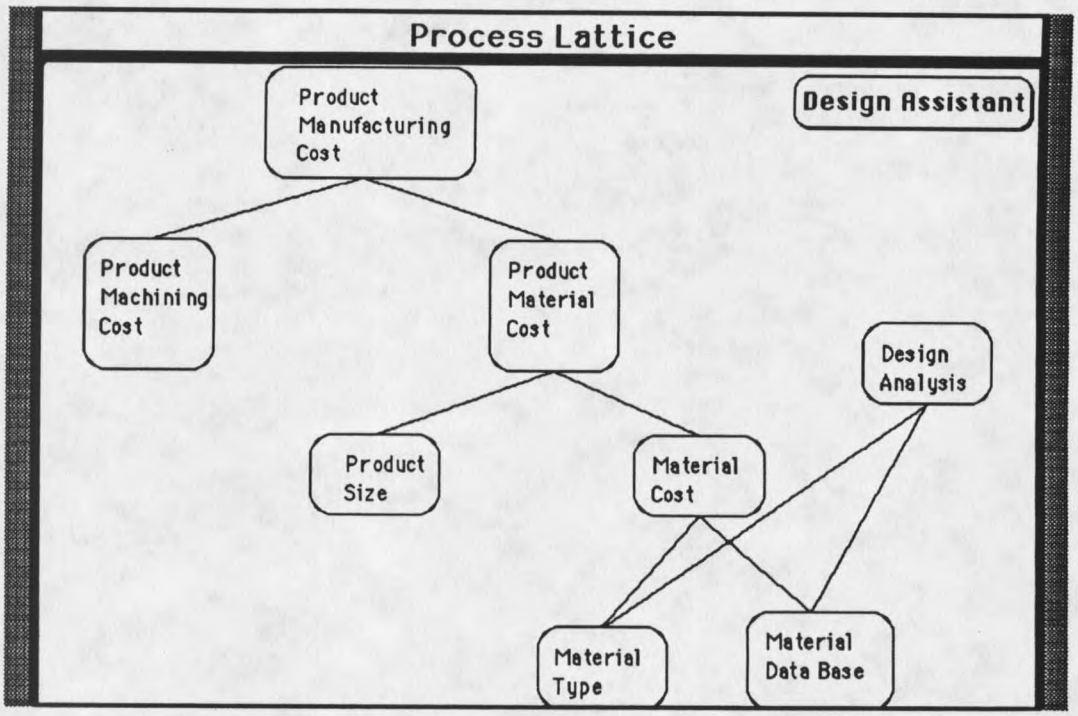

Figure 18. Process Lattice.

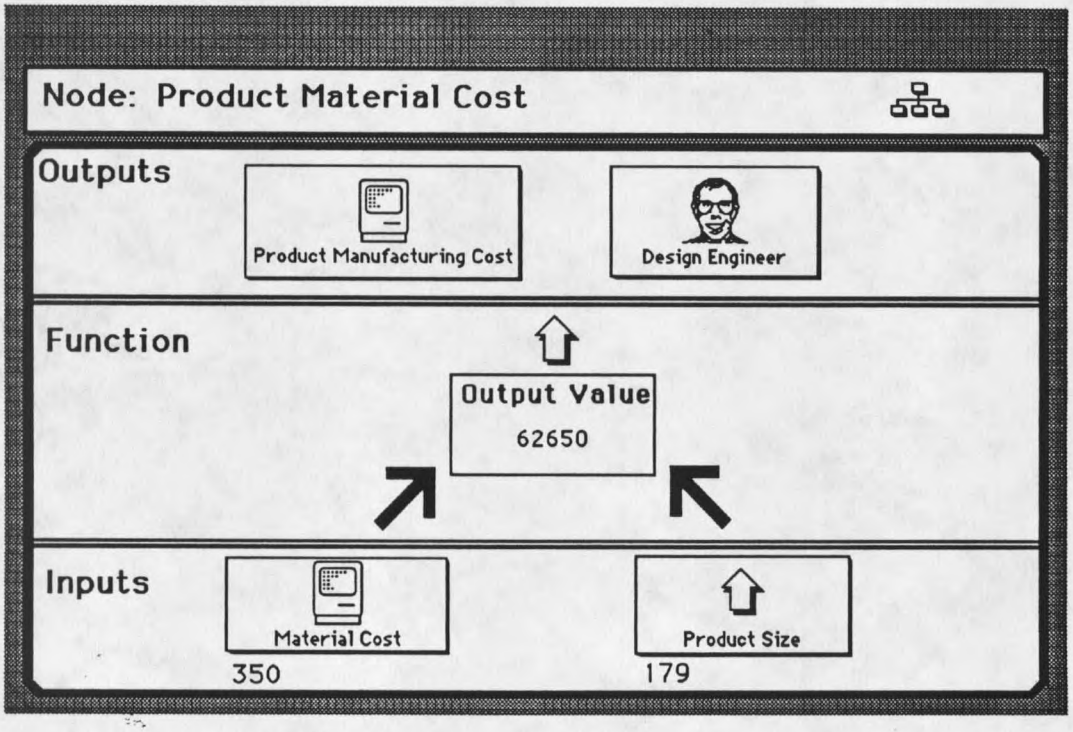

Figure 19. Node Description.

In this case the node transforms the two inputs, viz. Material Cost and Product Size, into one output value. This value is then sent to the node Product Manufacturing Cost as well as to the Design Engineer via an output value in the Constraint Monitor.

Material Cost Data Base

The Material Data Base, as illustrated in Figure 20, is used to store material attributes. In terms of the Design Assistant prototype, the only attribute used is material cost. These values, as well as material type, can be dynamically changed and the results automatically propagate through the process lattice. The new constraint analysis values are then displayed to the design engineer.

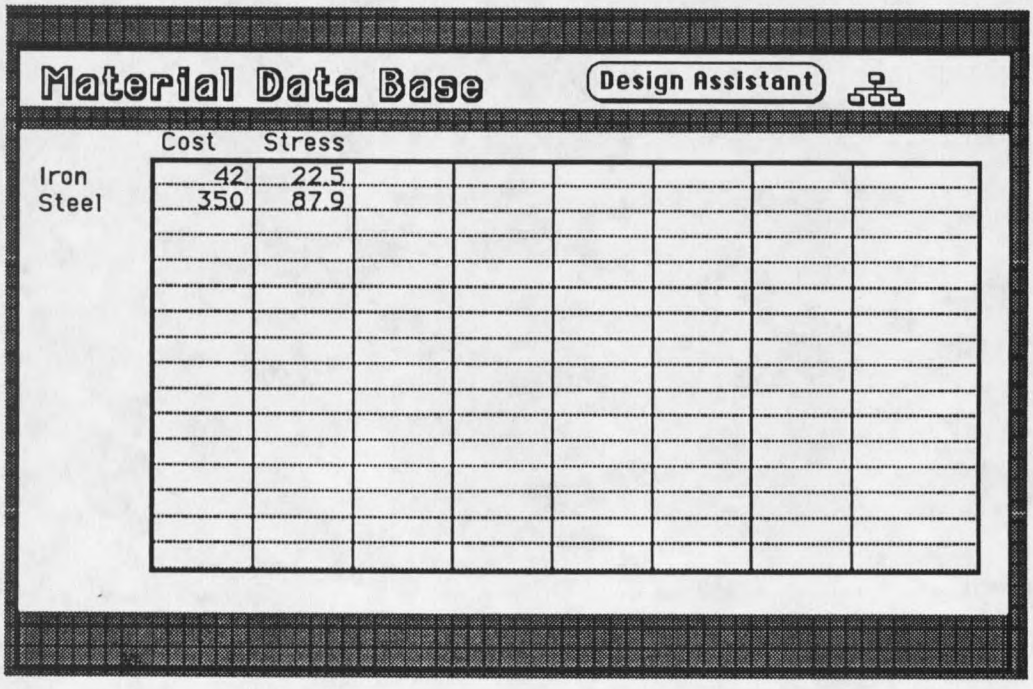

Figure 20. Material Cost Data Base.

Computer Aided Design (CAD) System

A CAD system would normally be integrated into the Design Assistant. Whether the CAD system itself is designated the primary interface to the Constraint Monitor, or whether some configuration like the Design Assistant is used, needs to be addressed from a human factors' viewpoint. Regardless, functionally the system must facilitate the integration of manufacturing constraint information into the design process.

Product Definition Data Base

The product definition data base consists of information regarding the mechanical product. This data is a combination of CAD generated data as well as specific CIM product management data. Figure 21 illustrates a conceptual data description for product definition data. The particulars of this data description are dependent upon the particular CAD and CIM system used. But nevertheless, this figure provides some idea of the type of information needed.

In this Figure, an *atom* refers to a single value, a *list* to a list of values, and a *schema* to a collection of attributes and their associated values.

In the prototype Design Assistant, a much simpler product data definition was used for demonstrational purposes. But though simplified, this product data was used as in a real application, namely changes in the data flowed upward through the lattice causing specified constraint information to be sent to the design engineer.

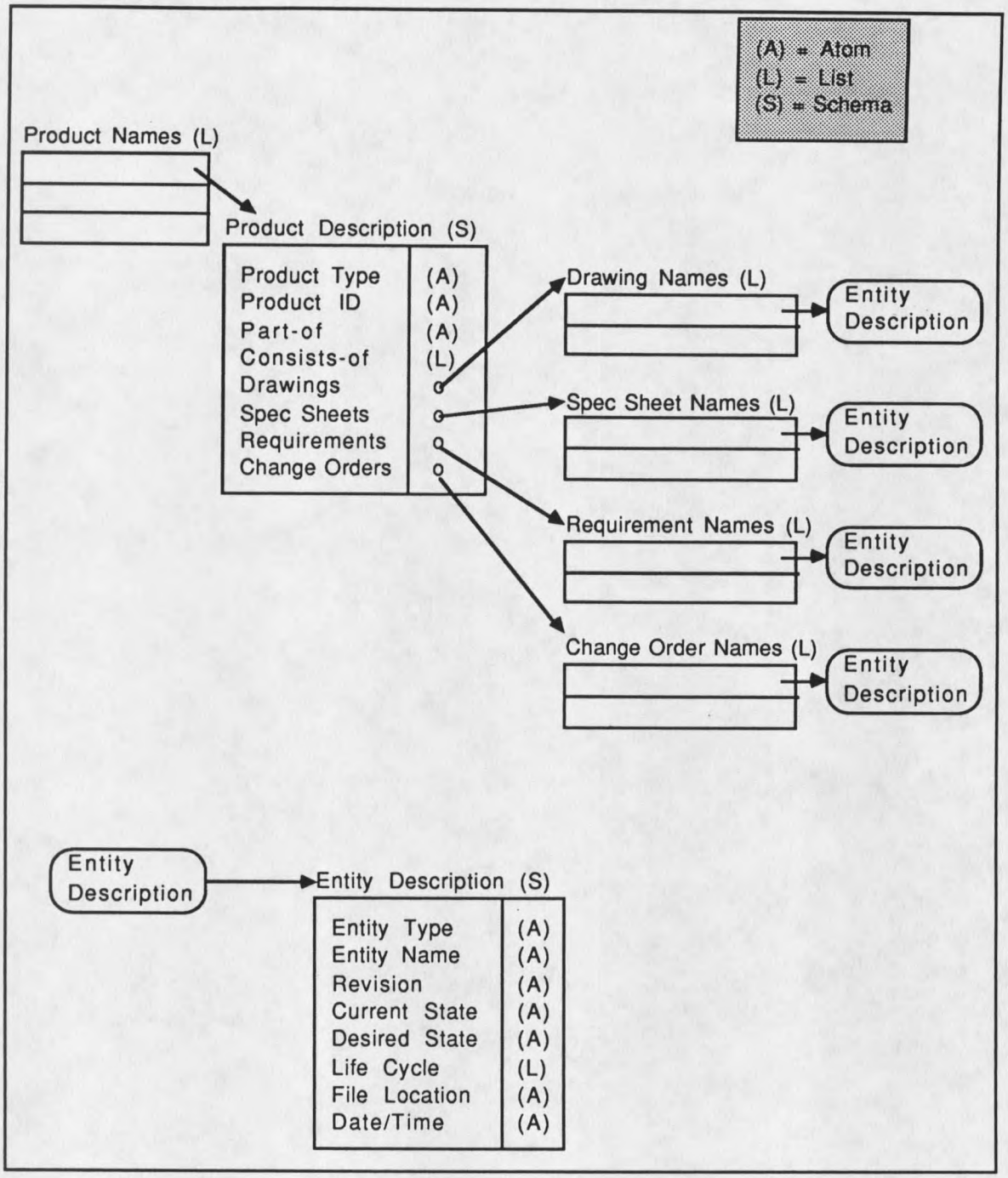

Figure 21. Data Description for Product Definition Data.

Design Recommendation Data Base

Figure 22 is an entry in the design recommendation data base. The underlying motivation of this data base is to provide the design engineer information regarding all recommendations made concerning the product's design. These recommendations could be generated by a traditional algorithm, an expert system, or a human expert. These recommendations not only facilitate the development of the product, but they also provide a product development history. Such information could prove useful for future needs, such as product development traceability.

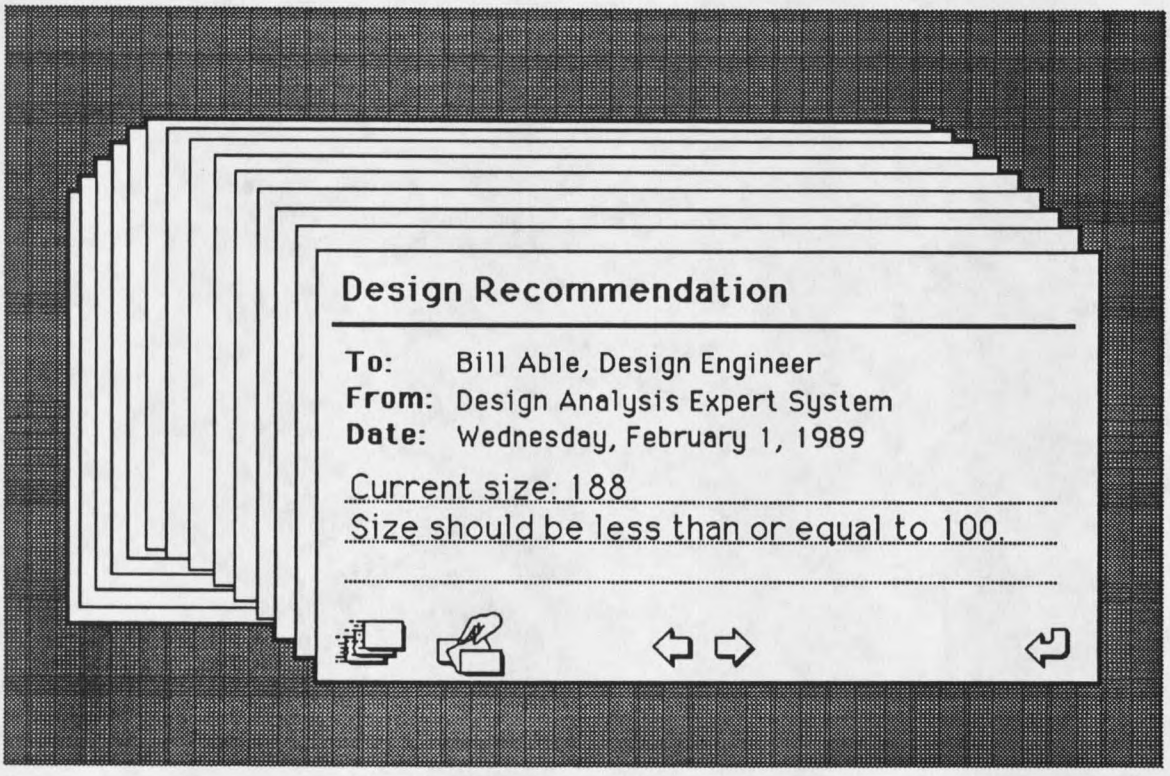

Figure 22. Design Recommendation Data Base.

The buttons in this data base provide means to view all design recommendations, sort recommendations by name or date, and to manually traverse the recommendations. This is merely a demonstrational data base. It is not intended to be a final design, but rather simply to demonstrate proof of concept.

Implementation Summary

In summary, this chapter discussed the application of a hierarchical organizational paradigm to CIM. In particular, the implementation language HyperCard was discussed as well as the implementation of the process lattice simulator and the Design Assistant, which incorporated various simplified components of a product design system to demonstrate the concept of the Constraint Monitor.

The following chapter discusses related research areas. In particular, it addresses issues that need to be considered in the expansion of this prototype Constraint Monitor into an actual production system.

RELATED RESEARCH AREAS

Since the Constraint Monitor is a distributed system, related areas of research involve the reliability, synchronization, and predictability of distributed systems.

System Reliability

The reliability of the distributed system concerns itself with the existence of single points of failure. As Hewitt [1986] notes, open systems must be designed such that the failure of any particular component can be accommodated by the other components. That is, the system as a whole must be able to operate continuously. Hewitt outlines four interrelated research areas: (1) system architectures that provide reliable results without centralized control, (2) programming languages that can express inherent concurrency and asynchrony of various components, (3) systems capable of operating over potentially inconsistent data, and (4) mathematical semantics to ground and unify the three preceding areas. These four areas need to be addressed in terms of developing a distributed CIM system that contains no single-point of failure.

System Synchronization

System synchronization needs to be considered in the development of a real-world CIM system based on the process lattice paradigm. The lattice used in this thesis achieves synchronization via message passing, since a process can become enabled only after it receives a sent message. This use of a message passing, though useful for synchronizing a distributed system, has potential problems. For instance, the number of messages sent within a fixed time span cannot increase indefinitely because of various limitations, such as limitations on the processing speed for information received as well as limitations on the number and size of message buffers. Subsequently, some form of flow control is needed. This could include the requirement that the receiver acknowledge each message sent before a new message can be sent. The question of the type of flow control that would best suit a Constraint Monitor based on the process lattice paradigm needs to be addressed.

System Predictability

In an open system inconsistent information can arise since the information available from many sources can disagree due to delays and asynchrony in its arrival while incomplete information can arise due to time constraints in obtaining information. The ramifications of systems with inherently inconsistent and incomplete knowledge, such as exhibited by open systems, is being studied in an area of research called chaos theory, a branch of non-linear mathematics.

Huberman and Hogg [1988], based on theoretical research in chaos theory, posit that random unpredictable fluctuations in performance may afflict large computer systems whose features include distributed control, asynchrony, resource contention, extensive communication among agents, and inconsistent and incomplete knowledge. In essence, a large distributed system with no central control could be nondeterministic. In other words, when presented with identical input, it might not always produce the same result. This seems to contradict common sense since in principle it seems possible to predict every action of a computer system. The problem arises not with the computers themselves but with their interactions with an unpredictable environment, such as the physical world or remote processing nodes on a network that lack a central controller. In essence, the problem is that since processing nodes -- by design -- have access to only their input and output ports, individual nodes must often make decisions based on incomplete knowledge of the entire system. This does not pose a problem if a node's information is consistent with that of the system as a whole. However, this is not necessarily the case. The data or instructions sent to another node in the network may be obsolete by the time it reaches its destination. This delay in information access may cause the behavior of the system as a whole to become unpredictable.

Kephart et al. [1989] simulated the dynamics of resource allocation in a model of computational ecosystems which incorporates many of these features of distributed systems. Their simulation verified quantitatively Huberman and Hogg's [1988] theoretical predictions of chaotic behavior. The simulation demonstrated that under certain conditions the slightest perturbation of the system caused unpredictable variations in the system's behavior. Furthermore,

attempts to overcome these instabilities by enabling processing nodes to monitor the network and change their behavior accordingly, resulted not in a more stable system but rather in one even more unstable.

Since the lattice paradigm used in this thesis to incorporate manufacturing constraints into the design process shares salient characteristics with the systems studied in chaos theory, research in this area warrants close attention. In particular, an issue that needs to be addressed is whether a Constraint Monitor developed for a large manufacturing system would exhibit unpredictable behavior. And if so, would this behavior be detrimental?

CONCLUSION

This thesis applied a process lattice paradigm, developed by Factor and Gelernter [1988], to the problem of incorporating manufacturing constraints into the design process. This paradigm was appropriate for this problem since regardless of whether a constraint analysis task is performed by a human expert, an expert system, or a conventional analysis program, each task can be considered a process that performs a design optimization function based on some given constraint. Each process receives input, such as product requirements and product definition data, and based on its knowledge of some manufacturing constraint, provides output in the form of constraint information. Furthermore, many of these processes could execute concurrently. For instance, a constraint based on material cost may be independent of one based on current material inventory. Hence, the problem of incorporating manufacturing constraints into the design process can be considered a problem of organizing asynchronous processes.

In this thesis economic constraints were used to demonstrate the feasibility of the organizational paradigm. Economic constraints were selected since substantial manufacturing cost savings are possible if cost constraints are integrated in the design process. This is possible since currently manufacturing costs are often considered a by-product rather than a prime mover of the design. Though all design processes do not occur with total disregard for economic constraints, typically these economic constraints are incorporated in

an ad hoc manner. The design approach outlined in this thesis imposes a formal organizational scheme on this ad hoc system in an attempt to provide the design engineer with timely estimates of the economic impact of various design alternatives. It should be noted, however, that this design is not limited to economic constraints, but is extendable to other manufacturing constraints. Furthermore, it should be noted that the design proposed in this thesis is not a knowledge-based system that incorporates the expertise of designers, process planners, and cost analysis engineers. Instead, it is an organizational scheme that integrates current expertise into a well-defined system.

REFERENCES

Appleton, D. S. 1985. "Building a CIM Program," in A Program Guide for CIM Implementation, C. M. Savage, ed., pp. 5-10. Dearborn, Michigan: The Computer and Automated Systems Association of SME.

Carriero, N. and D. Gelernter. 1989. Linda in Context. Communications of the ACM 32, 4 (Apr) 444-458.

Chang, T. and R. A. Wysk. 1985. An Introduction to Automated Process Planning Systems. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Compton, W. D. and N. A. Gjostein. 1986. Materials for Ground Transportation. Scientific American 255, 4 (Oct.) 93-100.

Factor, M. and D. Gelernter. 1988. The Parallel Process Lattice as an Organizing Scheme for Realtime Knowledge Daemons. Yale University Technical Report.

Fleischer, G. A. and B. Khoshnevis. 1986. "Incorporating Economic Impact Assessment into Computer-aided Design," in 1986 International Industrial Engineering Conference Proceedings, pp. 163-174. Norcross, Georgia: Institute of Industrial Engineers.

Hecht-Nielsen, R. 1988. Neurocomputing: picking the human brain. IEEE Spectrum (March) 13-18.

Hewitt, C. 1986. Concurrency in Intelligent Systems. AI Expert Premier 45-50.

Huberman, B. A. and T. Hogg. 1988. "The Behavior of Computational Ecologies," in The Ecology of Computation, B. A. Huberman, ed., pp. 77-115. Amsterdam, The Netherlands: North-Holland.

Kephart, J. O., T. Hogg, and B. A. Huberman. 1989. "Dynamics of Computational Ecosystems," to appear in Distributed Artificial Inteligence, vol. 2. Los Altos, CA: Morgan-Kaufmann.

Peterson, J. 1977. Petri Nets. ACM Comp. Surveys 9, 3 (Sept) 223-252.

Quinn, M. J. 1987. Designing Efficient Algorithms for Parallel Computers. New York, N.Y.: McGraw-Hill, Inc.

Razouk, R. and G. Estrin. 1982. "Modeling and Evaluation of Communication Protocols," in Computer Networks and Simulation II, S. Schoemaker, ed., pp. 167-190. New York, N.Y.: North-Holland Publishing Company.

Savage, C. M. 1985. "CIM and Management Policy: A word to the president," in A Program Guide for CIM Implementation, C. M. Savage, ed., pp. 11-18. Dearborn, Michigan: The Computer and Automated Systems Association of SME.

Stephens, M. and K. Whitehead. 1986. "The analyst -- a workstation for analysis and design," in Software Engineering Environments, I. Sommerville, ed., pp. 104-117. London, United Kingdom: Peter Peregrinus Ltd.

Sunshine, C. A. 1982. "Formal Modeling of Communication Protocols," in Computer Networks and Simulation II, S. Schoemaker, ed., pp. 53-78. New York, N.Y.: North-Holland Publishing Company.

Suri, R. 1988. "A New Perspective on Manufacturing Systems Analysis," in Design and Analysis of Integrated Manufacturing Systems, W. Dale Compton, ed., pp. 118-133. Washington, D.C.: National Academy Press.

Tipnis, V. A. 1988. "Process and Economic Models for Manufacturing Pperations," in Design and Analysis of Integrated Manufacturing Systems, W. Dale Compton, ed., pp. 92-117. Washington, D.C.: National Academy Press.

Treleaven, P. C. 1982. "Parallel Models of Computation," in Parallel Processing Systems, D. J. Evans, ed., pp. 275-284. Cambridge, England: Cambridge University Press.

Whiddett, D. 1987. Concurrent Programming for software engineers. New York, N.Y.: Halsted Press.

Whitney, D. E., J. L. Nevins, T. L. De Fazio, R. E. Gustavson, R. W. Metzinger, J. M. Rourke, and D. S. Seltzer. 1988. "The Strategic Approach to Product Sesign," in Design and Analysis of Integrated Manufacturing Systems, W. Dale Compton, ed., pp. 200-224. Washington, D.C.: National Academy Press.

APPENDIX

This appendix describes the functions of CIM data and process management systems. Each functions inputs, outputs, process, and driving requirements are specified. The requirements are abstracted from work done while a member of the CIM design team at Lockheed Missiles and Space Co.

Verify User Action

Inputs

1. User Requested Actions
2. User Control Knowledge

Outputs

1. User Requested Actions & Status

Process

Use the User Control Knowledge to determine whether the user is authorized to perform the User Requested Actions.

Requirements

1. Authorize and control CIM processing and data.

2. Provide the capabilities for a user to have access to all data and processes with one log-on.

3. Utilize user authorization and configuration management data to authorize and control CIM processing and data.

4. Ensure that access to data bases or portions of data bases is limited to that data to which the user has authorization.

5. Inhibit the ability of any user to access queries/reports to which that user has no access authority.

Determine CIM System Action

Inputs

1. Product Definition Data
2. CIM System Control Knowledge

Outputs

1. CIM System Requested Action

Process

Use the CIM System Control Rules to determine whether any CIM action should be requested based on information in the Product Definition Data Base.

Requirements

1. Automatically execute processes defined within the CIM system based on a set of conditions, including the automatic promotion of the product through its life cycle.

2. Provide a means of automatically notifying individuals based on an event within the CIM system.

Action Controller

Inputs

1. Network Requested Actions
2. User Requested Actions & Status
3. CIM System Requested Actions

Outputs

1. Sequenced Actions to be Executed

Process

Sequence all requested actions.

Requirements

1. Resolve contentions between multiple user requested actions, requested CIM system actions, and requested network actions.

2. Receive and distribute system event notification messages to the appropriate recipients.

Execute Action

Inputs

1. List of Sequenced Actions
2. User Control Knowledge
3. Product Definition Data
4. CIM System Control Knowledge

Outputs

1. Messages, Possible Actions, Reports
2. Messages, Requested Actions
3. Modified User Control Knowledge
4. Modified Product Definition Data
5. Modified CIM System Control Knowledge

Process

Execute each requested action. These actions include:

1. login the user
2. execute file system actions
3. generate reports
4. update the knowledge bases

5. determine possible user actions
6. define and modify control rules for a specific product

Requirements

1. Provide the capability to define life cycles and the rules for a specific product. This includes the ability to modify the life cycles and rules.

2. Control and maintain the product definition data.

3. Ensure current availability of product definition data and associated attribute information for each data element within the product definition data base.

4. Provide status of where a product is within its defined life cycle. This includes the ability to status assemblies within a product.

5. Generate transactions initiating and terminating CIM processes.

6. Generate transactions initiating and terminating CIM processes that execute in the BATCH mode.

7. Generate transactions to send or request receipt of data from external systems.

8. Replicate data from one component of CIM to another and perform the coding and structure translation required.

9. Generate errors to the system manager where version use between functions is incompatible.

10. Collect and maintain configuration and status of CIM hardware and software.

11. Display configuration and CIM hardware and software status to the system manager.

12. Maintain status of all parts in the system by part number.

13. Maintain standing query/report formats for each originating user in the CIM system

14. Provide the capability to allow an originating user to add/delete users from access to queries/reports he has created.

15. Provide a menu-driven file utility to copy, move, delete, rename, view, undelete, and to do a directory search as well as other functions appropriate to the CIM environment.

16. Maintain a Part Status Record for each part number. This record will contain the date and time of each step of the engineering-manufacturing process.

17. Update the Part Status Record with the date and time when notified of completion of the engineering design.