

Task Coordination in Concurrent Engineering

James Bowen and Dennis Bahler

Dept. of Computer Science, Box 8206

North Carolina State University

Raleigh, NC 27695-8206

Phone: (919) 515-7014

Fax: (919) 515-7896

Email: {jabowen|drb}@adm.csc.ncsu.edu

Prepared for AAAI'92 Workshop on Enterprise Integration

Abstract

Concurrent Engineering (CE) is an important facet of Enterprise Integration. Galileo3 is a language for constructing interactive design advice systems for CE, based on a generalization of constraint processing. We briefly overview the language, emphasizing the features which we believe make it especially suitable for constructing applications to support task coordination in CE. Then we provide an example interaction with a typical application program written in the language; this example scenario shows how an advisor written in Galileo3 supports coordination among multiple engineers who are interested in different aspects of a product life cycle. Galileo3 runs on IBM PS/2 workstations under DOS. Earlier versions of Galileo have run on DEC Ultrix workstations and a future port of Galileo3 to this environment is planned.

1 Introduction

Concurrent Engineering (CE) is an important aspect of Enterprise Integration. CE is an approach to product development which takes into account not just the functionality intended for an artifact but also other aspects of its life-cycle such as its manufacturability and testability [11; 12; 13; 14; 15]. One way to manage the large systems of sometimes competing requirements that arise in CE is to use product development teams. The deployment of such teams, however, often presents many logistic and scheduling difficulties [16; 17]. In addition, as the design evolves, the number of concerns can grow quickly beyond what even a team can successfully manage.

To facilitate coordination in CE, what is needed is an Intelligent Networked Colocation Advisor (INCA)

which not only relieves the logistic and scheduling difficulties but also reduces the problem complexity perceived by team members. Our objective is to develop a generic software tool for building such systems.

Galileo3 is the latest in a series of languages that have resulted from this research. Galileo3 is a generic language for constructing constraint-based INCA systems for CE. Systems have been written in Galileo3 and its predecessors (Leo [8], Galileo [10] and Galileo2 [6]) for a variety of areas, ranging from electronics[2; 3; 4; 6; 8; 10] to buildings[7] to fiber-reinforced composite materials[1]. These systems act as clearing houses for ensuring the consistency of decisions that are made by engineers who are concerned with different phases of the product life-cycle.

Galileo3 runs on IBM PS/2 workstations under DOS. Earlier versions of Galileo have run on DEC Ultrix workstations and a future port of Galileo3 to this environment is planned.

In Section 2, we briefly review the constraint-based approach to knowledge representation for CE. In Section 3, we provide an overview of the Galileo3 language, emphasizing those features which make it suitable for supporting task coordination. In Section 4, we illustrate the language in action, by presenting extracts from a series of interactions with a application system. In Section 5, we present our conclusions.

2 Constraint Networks

Research [1; 2; 3; 4; 6; 8; 10; 7] has shown that frame- or object-based¹ constraint networks are a suitable basis on which to build a language for CE applications. A constraint restricts the values that may be assumed by

¹The term **frame** used in the AI literature has much in common with the term **object** used in the Object Oriented Programming literature. Frames are passive objects which do not pass messages. Like objects, frames are data structures which can be organized in a hierarchy, with frames at one level inheriting attributes from their ancestors in the hierarchy.

a group of one or more parameters. A constraint network is a set of constraints, which are interconnected by virtue of sharing parameters. A frame-based constraint network is one in which parameters are not required to be scalars; instead, they can be either scalars or frames – complex data objects which can be organized in an inheritance hierarchy.

In a frame-based constraint network, frames can be used to represent: the artifact being designed; the components from which the artifact is configured or the materials from which it is made; and the life-cycle environment in which the artifact will be manufactured, tested and deployed. Constraints can be used to express in an explicit way the mutual restrictions exerted on each other by artifact functionality, component/material properties, and life-cycle processes.

A major attraction of constraint networks for CE is that constraints support multi-directional inference: information can flow in any direction through a network. Thus, for example, the impact of a design decision on the options available to a test engineer can be determined by propagating the design decision and its consequences throughout the network. Equally, if testing decisions are made early on, the impact of these decisions can be reflected in restrictions placed on the designer’s freedom. By supporting multi-directional inference, one constraint network can support both of these forms of interaction with equal ease.

3 Galileo3

3.1 Program Structure in Galileo3

A program in Galileo3 specifies a frame-based constraint network and comprises a set of declarations and definitions. Galileo3 is upwardly compatible with its predecessors; thus, any program that is well-formed in Leo, Galileo or Galileo2 is also well-formed in Galileo3.

The declaration statements include declarations of the parameters that exist in the network and the constraints that exist between these parameters. The definition statements include definitions for: partitions which divide the network into different regions that may be seen by different classes of user; application-specific domains, functions, and relations that are used in constraint declarations; and icons for application-specific domains, if it is intended to present certain regions of the network to users graphically rather than textually, a textual presentation being the default.

Consider, for example, the Galileo3 statements given in Figure 1. These are extracts from a program, which is called KLAUS2, that was built to assist in the concurrent engineering of printed wiring boards (PWBs). KLAUS2 provides intelligent networked colocation for

```

1 'the facility where the board will be tested'(the-test-facility)
   : test-facility.
2 domain test-facility ::=
   ('the name of the equipment at'(name) : tester-name,
    'the maximum clock frequency testable at'(maxfreq) : frequency,
    'the maximum number of test points testable at'(maxtpoints)
     : counter).
3 domain tester-name ::= (dynatest1,dynatest2,erdsys,
   radiconi,radicon2).
4 domain frequency ::= (X : number(X) and 3 <= X <= 40).
5 domain counter ::= (X : integer(X) and X >= 0).
6 domain component ::= ('the type of'(type) : component-type,
   'the number of leads on'(nleads) : counter).
7 domain component-type ::= (processor,dsp,comm-controller,divider,
   crystal,resistor,capacitor,diode).
8 domain 'crystal oscillator'(osc-crystal) ::=
   (X : component(X) and X.type=crystal and
    exists(X.'the oscillation frequency of'(freq) : positive)).
9 domain 'frequency divider'(divider) ::=
   (X:component(X) and X.type=divider and ... ).
10 domain cpu ::= (X:component(X) and X.type=processor and ... ).
11 domain resistor ::= (X:component(X) and X.type=resistor and ... ).
12 all X : component(X) and X.type in (processor,comm-controller,dsp)
   implies exists(X.'the oscillator for'(crystal) : osc-crystal) and
   exists(X.'the pullup resistor for'(pullup) : resistor).
13 all X : osc-crystal(X) and the-test-facility.maxfreq < X.freq
   implies exists(X.'an ancillary divider for'(anc-divider) : divider).
14 field 'the circuit designers perspective'(configuration) ::=
   (X : component(X)).
15 field 'the perspective taken by test engineers'(testability) ::=
   (the-test-facility).
16 permission((X:component(X)),configuration).

```

Figure 1: Extracts from a Galileo3 program.

several members of the design team, including designers, manufacturing engineers, and test engineers. In considering these extracts from KLAUS2, we will see how the differing perspectives of each team member are supported. Later, in Section 4, we will see how KLAUS2 provides system-mediated negotiation when the differing perspectives of these users cause them to make conflicting decisions. For further information on KLAUS2, see [3].

In a Galileo3 program, statements can be written in any order, although good software engineering practice dictates that some agreed standard ordering be used, in order to facilitate program maintenance. In Figure 1, we have taken advantage of this free-ordering by listing the statements in an order which facilitates discussion in this paper; in KLAUS2 itself, however, the statements are ordered differently. Although the statements are numbered in Figure 1, this is purely for the purposes of exposition; the numbers are not part of the program text.

3.2 Representing the PPO Model

In Galileo3 programs, frames are used to represent the Product, Process and Organization (PPO) model. Consider, for example, statement 1, which declares the existence of a parameter called `the_test_facility` and specifies that it is of type, or domain, `test_facility`. The string delimited by apostrophes is a long synonym which specifies the name by which the parameter will be referenced in all output given to users of the program.

Statement 2 defines the application-specific domain `test_facility`. This specifies that a test facility is represented by a frame containing three slots. For each slot, we specify its name, its long synonym and its domain. In output given by the system to users, the long synonyms of the slots in a frame are concatenated with the long synonyms of the frame-valued parameters. This can be seen in Figure 2, where the slots of the `the_test_facility` parameter are presented in a scrollsheets; thus, for example, the slot `the_test_facility.maxfreq` is named as the maximum clock frequency testable at the facility where the board will be tested.

<input type="checkbox"/> Help	<input type="checkbox"/> File	<input type="checkbox"/> New	<input type="checkbox"/> Utilities	<input type="checkbox"/> Search
<input type="checkbox"/> Up	<input type="checkbox"/> Down	<input type="checkbox"/> Focus	<input type="checkbox"/> Toggle	
<input type="checkbox"/> the name of the equipment at the facility where the board will be tested				erdsys
<input type="checkbox"/> the maximum clock frequency testable at the facility where the board will be tested				9.8
<input type="checkbox"/> the maximum number of test points testable at the facility where the board will be tested				200
>>>				
KLAUS2 - a PWB Design Advisor (Testability)				

Figure 2

The constituent slots of a `test_facility` frame, as specified in statement 2, have scalar application-specific domains. These are defined in statements 3, 4 and 5. Notice that, while scalar domains may be defined extensionally by listing each possible value (statement 3), they may also be defined intensionally by specifying an appropriate logical formula (statements 4 and 5).

Frame-based inheritance is illustrated in statements 6, 7, 8, 9, 10 and 11. Statement 6 defines the frame domain `component`; for simplicity, we give only two slots, although in KLAUS2 this frame has many more slots. Notice, from statement 7, that there can be several different types of component. By using inheritance, frames for these different types of component can be defined in terms of the generic component frame. In statement 8, for example, it is specified that an `osc_crystal` frame, which is used to represent a crystal oscillator, is a `component` frame which has the value `crystal` in its `type` slot and which also has an additional slot `freq` that represents the frequency at which the crystal oscillates. In statement 9, a `divider` frame, which is used to represent a frequency divider, is specified to be a `component` frame in which the `type` slot has the value `divider` and which has several other slots – these additional slots are not specified here, however. A `cpu` frame is defined similarly in statement 10, and a `resistor` frame in statement 11.

3.3 Life-Cycle Interdependencies

In Galileo3, constraints are used to represent the mutually restrictive influences that the different parts of the PPO model exert on each other.

In Galileo3, any sentence (atomic, compound or quantified) in logic is a well-formed constraint. In Figure 1, statements 12 and 13 are universally quantified constraints. Statement 12 specifies that any parameter of domain `component` which is used to represent a CPU, a communications controller or a digital signal processor device, must have an extra slot to represent the oscillator which drives the device and a further slot to represent a pullup resistor.

Statement 13 specifies that if any crystal is oscillating faster than the maximum clock speed that is testable by the test facility then the crystal must have an ancillary divider circuit. Although Galileo3 supports both universal and existential quantification, the `exists` symbol in statement 13 is not an existential quantifier. In Galileo3, conditional existence of parameters and of parameter slots is based on free logic [5]; the `exists` token in statement 13 is a free logic existence specifier.

3.4 Multiple Perspectives

Galileo3 enables constraint networks to be divided into (possibly overlapping) regions called **fields of view**. A field of view is that region within a constraint network that is currently important to a user interacting with the network. A field of view can be either global or local. The global field of view consists of the entire constraint network. A local field of view contains only a subnetwork. Each field of view contains all the parameters that are of interest to the user, as well as all constraints which reference these parameters.

We define a field of view by specifying the set of parameters which it contains. In Figure 1, for example, we define two of the fields of view that are provided by the KLAUS2 application. Statement 14 defines a `configuration` field of view, which will be seen by a circuit designer, and specifies that it contains the set of all parameters of domain `component`. Statement 15 defines a `testability` field of view and specifies that its set of parameters contains just one, `the_test_facility`.

3.5 Multiple Interface Styles

Different fields of view can be presented to their users through different styles of interface. In the KLAUS2 application, for example, the `testability` field of view is presented through a scrollsheets interface (Figure 2), while the `configuration` is presented through a feature-based CAD interface in which each parameter

in the field of view is seen by the user as an icon on the CAD drawing (Figure 3). Although the specification of these different types of interface is a simple matter in Galileo3, detailed discussion is beyond the scope of this paper.

3.6 Representing Design Decisions

Galileo3 programs are interactive. A user can input additional constraints, to represent his design decisions. Thus, for example, if a test engineer decides to use an Erdsys tester, he can indicate this decision by inputting the following constraint: `the_test_facility.name = erdsys`. (In fact, the test engineer does not have to type this constraint — it can be input by using a mouse to select appropriate options in a series of pull-up menus; the test engineer need never know about such “unfriendly” names as `the_test_facility.name`.) A user can also input additional parameters. Thus, for example, a circuit designer interacting with the KLAUS2 application can introduce new parameters to represent various parts of his evolving circuit. To introduce a CPU, for example, he can either introduce a parameter of domain `component` and specify that the type `slot` of this parameter has the value `processor` or he can introduce a parameter of domain `cpu`, which amounts to doing the same thing.

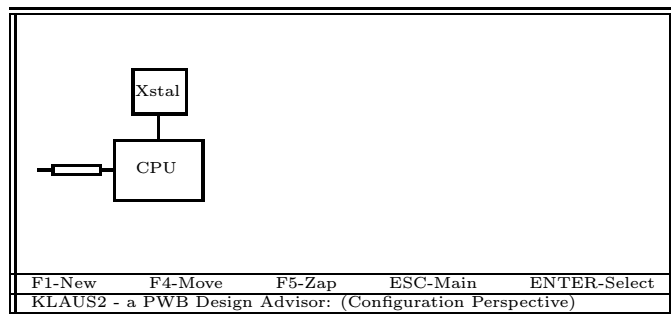


Figure 3

In Galileo3, we can specify which users of an application that supports multiple fields of view are allowed to introduce new parameters and what classes of parameters they are allowed to enter. Statement 16 of Figure 1, for example, specifies that users of the `configuration` field of view are allowed to introduce parameters of domain `component` or of any domain (such as `osc_crystal`, `divider`, `resistor` or `cpu`) that is a sub-class of the `component` domain.

3.7 Supporting User Queries

As well as specifying information by introducing new parameters and new constraints, a user can ask for information. He can, for example, ask for the range of al-

lowable values for any of the parameters in a network. He can also ask for justifications for these ranges — whenever the range of allowable values for a parameter is reduced by a constraint, the rationale for this reduction is noted by the run-time system as a dependency record which can be accessed later for justification purposes. A user can always withdraw any constraint or parameter that he has added. Thus, by introducing and withdrawing constraints and parameters, the user can investigate “what if” scenarios.

4 An Example Scenario

To illustrate how programs written in Galileo3 support task coordination in CE, we will show how the KLAUS2 application that we considered above supports interaction between several members of a PWB design team, including designers, manufacturing engineers, and test engineers. We will see how the differing perspectives of each team member are supported, and how KLAUS2 provides system-mediated coordination between team members when their perspectives lead to conflicting decisions.

Several points deserve emphasis about this scenario. First, this scenario presents only one of very many possible orders of interaction with KLAUS2. Second, we show only a few steps in this interaction. Finally, although the figures in this document are not actual screen dumps, KLAUS2 and Galileo3 are fully implemented, and these figures faithfully represent the various interfaces presented to the manufacturing, testing, and design users.

In the following scenario, we assume that the project leader has set up a database entry for a new project. We take up the story when the test engineer, who in this case is the first team member to make some decisions about this new project, starts to interact with KLAUS2 about the project. Figure 2 shows the interface presented by KLAUS2 to the test engineer after he has selected the test equipment to be used for the project. The largest window in this screen is a single-column spreadsheet, or “scrollsheet,” in which each cell occupies one or more lines.

Suppose the manufacturing engineer also interacts with KLAUS2 before circuit design begins. Figure 4 shows the results of his decisions about the drilling and assembly equipment to be used. Notice that the parameters representing the number of chosen hole sizes and the list of chosen hole sizes are not yet bound because their values depend on choices not yet made by the board designer.

[] Help	[] File	[] New	[] Utilities	[] Search
[] Up	[] Down	[] Focus	[] Toggle	
[] the name of the drilling machine	bransch2			
[] the magazine size of the drilling machine	2			
[] the list of standard drill sizes for the drilling machine	{0.04,0.045,0.05,0.055,0.06,0.065}			
[] the list of chosen hole sizes				
[] the number of chosen hole sizes				
[] the name of the assembly robot	yotsui19			
[] the insertion oversize needed by the assembly robot	0.016			
>>>				
KLAUS2 - a PWB Design Advisor (Manufacturing)				

Figure 4

Figure 3 shows the perspective of the circuit designer. Notice that this perspective provides a feature-based CAD interface rather than the scrollsheets provided to the test and manufacturing engineers. The drawing in this screen represents the circuit after the designer has introduced a CPU, and KLAUS2 has used its constraint information to induce automatically the need for an associated pullup resistor and oscillating crystal, and the designer has decided to accept these suggested components as part of his design.

Suppose that the designer, having accepted the crystal, specifies that it should oscillate at 25 Mhz. Now, however, the constraint in statement 13 of Figure 1 comes into play and, because 25 Mhz exceeds the maximum testable frequency of 9.8 Mhz specified earlier by the test engineer (Figure 2), the system introduces an ancillary divider circuit for this oscillator. The result can be seen in Figure 5, where KLAUS2 is suggesting that the new component on the screen should be added to the circuit.

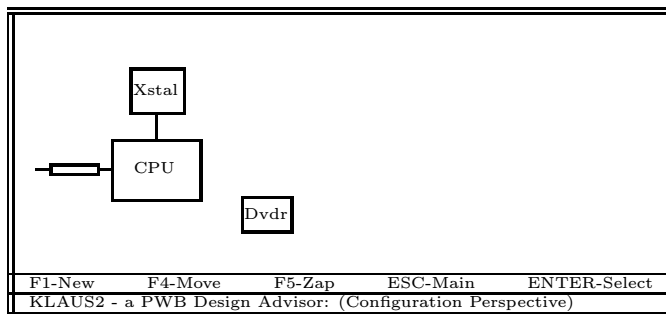


Figure 5

The designer is surprised by the introduction of this component, so he asks KLAUS2 to justify it. In KLAUS2's explanation (Figure 6), the constraint given above, which introduced the component, is paraphrased in natural language.

<p>Justification</p> <p>an ancillary divider for the oscillator for the cpu exists because of the following constraint</p> <p>(38) every crystal oscillator must satisfy the following:</p> <p>if the maximum clock frequency testable at the facility where the board will be tested < the oscillation frequency of the crystal oscillator then an ancillary divider for the crystal oscillator must exist and must be a frequency divider;</p> <p>and because of the following parameter value(s):</p> <p>the maximum clock frequency testable at the facility where the board will be tested = 9.8;</p> <p>the oscillation frequency of the oscillator for the cpu = 25.</p> <p>the maximum clock frequency testable at the facility where the board will be tested was established according to the perspective taken by test engineers.</p> <p>the oscillation frequency of the oscillator for the cpu = 25 because you said so.</p>

Figure 6

Despite this justification, the designer decides to reject this ancillary component. Now, however, the constraint which wanted to introduce the ancillary divider is violated, leading to the message shown in Figure 7.

<p>VIOLATION:</p> <p>an ancillary divider for the oscillator for the cpu should exist, but is prohibited.</p> <p>SUGGESTIONS:</p> <p>(1) Retract the constraint</p> <p>(104) an ancillary divider for the oscillator for the cpu should not exist.</p> <p>(2) Disable the constraint</p> <p>(38) every crystal oscillator must satisfy the following:</p> <p>if the maximum clock frequency testable at the facility where the board will be tested < the oscillation frequency of the crystal oscillator then an ancillary divider for the crystal oscillator must exist and must be a frequency divider.</p> <p>(3) Change the oscillation frequency of the oscillator for the cpu.</p> <p>(4) Request that, in the perspective taken by test engineers, a change be made to the maximum clock frequency testable at the facility where the board will be tested.</p>
--

Figure 7

Choosing among the suggestions offered in this message, the designer decides to disable constraint (38). However, this constraint also refers to a parameter in the test perspective, so the decision to disable the constraint must be accepted by the test engineer. Whenever a user disables a constraint other than one he previously asserted himself, he is required to enter a free-text explanation of his action, which is saved for possible use in a design audit. These free-text explanations are also used as a form of email in system-mediated interaction between users of different perspectives when they make conflicting decisions.

When the test engineer next logs into KLAUS2, he is told that a constraint of interest to him was disabled by another user. Checking on this, he calls up the design state from the database. The system tells him which constraint was disabled and produces the free-text explanation given by the circuit designer.

The test engineer decides that he is unwilling to allow this constraint to be disabled because of the difficulty in testing that would result. However, to com-

promise, he changes the test equipment to one which is able to handle a frequency of 25 MHz. After making this change, the test engineer reactivates the disabled constraint (38). Because of the higher frequency testable by the test equipment, the re-enabled constraint does not attempt to re-introduce an ancillary divider circuit, so no constraint violation occurs. The test engineer saves the new design state and starts to work on another project.

When the circuit designer next logs in, he is told that the test engineer has re-enabled the constraint but, since the unwanted divider circuit has not reappeared, the designer is content.

Suppose, however, that no such easy compromise was possible. For example, the designer might have selected an oscillation frequency that exceeded even the upper limit of the fastest available tester. In this case, the test engineer and designer will successively disable and reenable their shared constraint (38), offering free-text explanations to each other until one or the other gives way or appeals to the project leader. In this case, the test engineer could give give way by deciding to build a special divider test fixture; the circuit designer could give way by using a lower oscillation frequency.

5 Conclusions

Constraint networks provide an attractive technology for building systems to support coordination in CE. Constraints can represent a rich variety of interrelationships among hierarchically organized structured objects. Constraint networks allow information to flow in any direction through the network, and thus neither impose an *a priori* order of interaction nor prejudice the system towards one particular design methodology at the expense of the alternatives.

It is often thought that constraint networks are only good for parametric design. They are frequently believed to be incapable of handling structural, or architectural, design. This is a fallacy arising from a view of constraints that is too limited. As we have seen, in an interactive session with a constraint-based CE advisor, the user can specify any architecture he wants by introducing new parameters to represent his evolving architecture. In addition, constraint networks based on free logic [5] can autonomously add new parameters, which means that network-based CE systems can advise their users when it would be appropriate to add new components or processes to the evolving design of the product or its life-cycle.

We have seen how constraint networks are capable of supporting cooperation between different users of a system that provides intelligent networked colocation.

We recognize that the simple form of iterative mediation that was illustrated in Section 4 constitutes only the beginning of a facility for negotiation support. Indeed, more comprehensive support for negotiation constitutes a major task in our ongoing research.

References

- [1] Bahler D, Spainhour L, Rasdorf W, and Bowen J, 1992. "A Constraint-Based Approach to Fiber-Reinforced Composite Material Design, Analysis, and Verification," Technical Report, Department of Computer Science, North Carolina State University.
- [2] Bowen J and Bahler D, 1992. "An AI constraint network-based approach to bed-of-nails DFT for digital circuit design," *Computers and Electrical Engineering*, to appear.
- [3] Bowen J and Bahler D, 1992. "Frames, Quantification, Perspectives and Negotiation in Constraint Networks for Life-Cycle Engineering," *International Journal of AI in Engineering*, to appear.
- [4] Bowen J and Bahler D, 1992. "Supporting Multiple Perspectives: A Constraint-Based Approach to Concurrent Engineering," *Proc. 2nd International Conf. on Artificial Intelligence in Design*, Pittsburgh.
- [5] Bowen J and Bahler D, 1991. "Conditional Variable Existence in Generalized Constraint Networks," *Proc. 9th Natl. Conf. on Artificial Intelligence (AAAI-91)*, 251-256.
- [6] Bowen J and Bahler D, 1991. "Supporting cooperation between multiple perspectives in a constraint-based approach to concurrent engineering," *Journal of Design and Manufacturing*, **1**, 89-105.
- [7] Bowen J and Elsaïdy W, 1990. "Life Cycle Design of Elevator Hoistways," Technical Report, Department of Computer Science, North Carolina State University.
- [8] Bowen J, O'Grady P, and Smith L, 1990. "A Constraint Programming Language for Life-Cycle Engineering," *International Journal for Artificial Intelligence in Engineering*, **5**, 206-220.
- [9] Bowen J, Lai R, and Bahler D, 1992. "Lexical Imprecision and Fuzzy Constraint Networks," *Proc. 10th Natl. Conf. on Artificial Intelligence (AAAI-92)*, San Jose, CA.

- [10] Dholakia A, 1991. "RICK: A DFT Advisor for Digital Circuit Design," *AutoTestCon '91*, Anaheim, CA. **Winner, National Student Paper Competition.**
- [11] Evans B, 1988. "Simultaneous Engineering," *Mechanical Engineering* 110, 2.
- [12] Ishii K, Adler R, and Barkan P, 1988. "Knowledge-based simultaneous engineering using design compatibility analysis," in J Gero (Ed.), *Artificial Intelligence in Engineering: Design*, NY: Elsevier, 361-378.
- [13] Maddux K, and Jain S, 1986. "CAE for Manufacturing Engineering: The Role of Process Simulation in Concurrent Engineering," *Proc. Winter Ann. Meeting of Amer. Soc. of Mechanical Engineers*.
- [14] Matsumoto A, Jagannathan V, Buenzli C, and Saks V, 1989, "Concurrent Design for Testability," *Proc. of the Workshop on Concurrent Engineering Design*, IJCAI-89, Detroit.
- [15] National Science Foundation, 1987. *Research Priorities for Proposed NSF Strategic Manufacturing Initiative*, Report of an NSF Workshop, NSF, Washington DC.
- [16] Pourbabai B and Pecht M, 1991, "Management of the Product Design Process," *Proc. CALS & CE Conference*, Washington D.C.
- [17] Sprague R, 1991, "The Realities of Concurrent Engineering Deployment," *Proc. CALS & CE Conference*, Washington D.C.