Chapter 5
Desirable Functionalities of Intelligent CAD Systems

Varol Akman
Department of Computer Engineering and Information Science
Bilkent University
Ankara, Turkey

Paul J. W. ten Hagen
Department of Interactive Systems
Center for Mathematics and Computer Science
Amsterdam, The Netherlands

Tetsuo Tomiyama
Department of Precision Machinery Engineering
University of Tokyo
Tokyo, Japan

5.1 INTRODUCTION

Computer-aided design programs assist a designer in specifying an artifact. The assistance can range from merely registering the design results to analyzing the proper functioning of the designed object, perhaps through simulation or, more properly, through envisioning (de Kleer, 1975). More advanced forms of assistance include problem-solving activities such as optimization and routing, to even suggesting a solution based on the given specifications. The latter activities become more dominant when the assistance can be extended to the earlier and more difficult phases of design.

The design process can be defined as transforming a set of specifications into a set of attributed objects, which together perform as required by the specifications. The process can be structured in terms of stages (analysis, synthesis, and evaluation). Moreover, there may be forms of backtracking, iteration, detailing, and so on. In fact, the design process can be characterized by the way it interacts with the design object. An important goal is to find appropriate means for describing the design process and to define the semantics in terms of design transactions (Mostow, 1985).

Much of a designer's activities consist of manipulations of the design object, modifications, and inspections. The context in which these take place may vary.

Even the purpose of such activities may initially be left unspecified. This could, in some cases, influence the way in which the corresponding transactions are visualized. A design object must be properly represented to identify the status of the object information: proposed, decided, and so on. Interaction between process representations and object representations depends on the status information. In particular, the status information must be allowed to be incomplete, inaccurate, and even inconsistent in a given intermediate situation (Purcell, 1988).

Mechanical part design is the domain under discussion here because it probably has the strongest industrial appeal. As a well-established, integral part of Computer-Integrated Manufacturing (CIM), mechanical CAD is the backbone of today's highly industrialized world. It helps engineers develop products ranging from the simple and ordinary to the complex and sophisticated. It multiplies the productivity many times and renders, using CIM techniques (Yeomans *et al.*, 1985), robust products.

However, it is commonly accepted that current CAD systems are difficult to master and inflexible to growing needs. They can only deal with limited domains and occlude attempts at integration. They do not properly support a crucial ingredient of design, i.e., interaction. What is worse is that they lack the distinguishing characteristic of human designers: they have little or no intelligence (Simon, 1979). As technologies advance, it is conjectured that only systems that embed advanced reasoning capabilities will be able to deal with the complexity arising from the management of large quantities of design data.

Efforts to provide a wider perspective of CAD are now underway at several companies, universities, and research institutes. Because design is a highly mental activity, researchers have long felt the need for making CAD systems more intelligent (Akman *et al.*, 1989). It should, on the other hand, be remarked that design is not basically a mental activity. Design is basically a physically creative activity. It is only since the industrial revolution that certain kinds, such as mechanical design, have become so abstracted from the associated physically creative (manufacturing) aspects that design can now appear to be a purely mental activity.

The work this chapter is based on advances the theory and practice of intelligent CAD systems. It is based on IIICAD (Intelligent, Integrated, and Interactive CAD) which employs artificial intelligence (AI) techniques and knowledge engineering tools to obtain a system that is potentially more substantial than expert systems (Lansdown, 1988). This chapter gives an overview of the IIICAD philosophy.

5.2 PROBLEMS WITH EXISTING CAD SYSTEMS

Sutherland's revolutionary SKETCHPAD (Sutherland, 1963) generated much enthusiasm for using interactive graphics in engineering. SKETCHPAD allowed geometric shapes to be represented and various conditions to be specified over these shapes in terms of constraints. This, in turn, provoked a stormy decade when turnkey, two-dimensional drafting systems gradually replaced drawing boards in the professional environment. Finally, the dust settled with their general acceptance by industry. Three-dimensional modeling systems became available and they were also widely accepted as indispensable tools in product development.

Nevertheless, borrowing an analogy from Bobrow *et al.* (1986), it can be claimed that all these systems follow the "low road" approach. They are programmed in an ad hoc manner and regard design from a singular viewpoint, i.e., as a mainly geometric activity. Thus, despite their popularity, there are problems with the existing CAD systems. Even the recent research cannot escape various inherited pitfalls. In this section, these problems are highlighted.

Methodology

As with any other discipline, a critique of CAD systems presupposes a starting point, i.e., a vision of design. Briefly, design can be seen as an intellectual activity performed by human designers where the essential thing about a designer is that she builds her vision of the world. Thus, design systems should provide a framework in which designers can exercise their faculties at large. With this view, the idea of apprentice (as opposed to autonomous) CAD systems can be more or less supported. This chapter challenges some of the commonly held views about the nature of design such as:

(1) Design is a routine process
(2) Design is an innovative process
(3) Design is a problem-solving process
(4) Design is a decision-making process
(5) Design is an optimization process

The first view treats design as a rather straightforward activity where the designer selects from a known set of well-understood alternatives. A recent example is the AIR-CYL system (Brown and Chandrasekaran, 1986). Clearly, this view is ingenuous and does not reflect the intricate nature of design. The second view goes to the other extreme and embraces the exciting ideas of AI to create novel devices via brainstorming and discovery heuristics. It is quite early to predict whether this can be achieved in domains more involved than the usual micro worlds of AI; thus, EDISON is only a toy system (Dyer *et al.*, 1986). Although the remaining views in the preceding list regard design as an activity requiring intelligence and creativity, they underplay its holistic nature. They are mostly implemented as expert systems which solve the specific problems of a specific design process. An example of this "middle road" approach is the PRIDE system (Mittal *et al.*, 1986). An annoying and often cited problem with expert systems is that they cannot deliver genuinely expert performance because they have no underlying mechanism with which to understand what is going on (Doyle, 1985). This problem manifests itself when a particular expert system is unable to solve a simple problem in spite of its proven expertise with difficult problems. This discrepancy has contributed to the emergence of terms like *deep* and *shallow.*

"High road" systems are deep systems and IIICAD is aimed at them. The knowledge of such systems is to represent the principles and theories underlying the discipline of design. This requires demystifying several aspects of design by way of formal, mathematical methods. Clearly, such a formalization may not say much about what goes on during the design process, serving as a post hoc rationalization of it. In the case of IIICAD, the fundamentals of a general design theory that is based on axiomatic set theory can be found in Tomiyama and Yoshikawa (1987). A comprehensive and detailed model of design is not easily available. There are many domain-specific sides to design. For instance, VLSI design is mostly two-dimensional, whereas mechanical design is inherently three-dimensional. IIICAD hopes to incorporate the similarities in design processes, leaving the application-dependent issues for further consideration. Only through a clean design theory and formalization can one arrive at testable conjectures of design; logic is the principal instrument in this endeavor (Hayes, 1977; Moore, 1985; Akman and Tomiyama, 1989).

Key Problems

CAD Systems Support Few Design Processes/Models and Lack Integration

Producing final drawings is where current CAD systems excel. This clearly depends on what is meant by final drawings, but in the usual sense of them being what is handed to a production engineer, CAD systems are not that good even at producing final drawings; they cannot yet handle many of the details involved, such as tolerances, finishes, and materials. Also, they are virtually powerless with respect to initial sketching. There are systems that can accept rough drawings, but there is no system that can handle such information during the design process.

Another weakness in terms of design processes can be seen by comparing software development and machine design cycles. There is no corresponding tool in mechanical CAD for what is known as *formal specifications* in software. The same goes for *documentation*, which is de rigueur in software engineering. Last but not least, there are few established methods other than testing to guarantee the correctness of manufactured design objects.

Integration of models is essential because mechanical design deals with complicated gadgets. A design object must be viewed from various angles using different models, e.g., the kinematic, dynamic, finite element, and control theoretic models of a robot manipulator (Forbus, 1988). In general, the present trend is to integrate CAD systems around models concerning products. In fact, models should be process-oriented and the so-called conceptual design stage should be supported by tools. Thus integration will be a synthesis of (1) design subsystems, (2) design models and views based on an integrated data description scheme, and (3) design processes (especially those of the early design stages).

CAD Systems Do Not Support Error Checking

Current CAD systems are not fully able to recognize inconsistencies in their input data. To worsen the situation, final outputs of conventional systems are so impressive that many errors go unnoticed for they exceed the capacity of the designers. A remedy is to provide continuous error checks and to make sure that only the correct commands are accepted. Unfortunately, semantic error checks are difficult. A few systems provide good user interfaces to make suggestions against simple mistakes, but this is only a small part of what is eventually needed.

Data Entry Is Problematic

This has to do with the lack of task-domain terminology in the system. Because a conventional CAD system has no commonsense knowledge of machine design and cannot follow the designer's intentions (McCarthy, 1968), one is likely to enter a good deal of information to state simple requests like "Here I need a hole to insert the shaft I just created." When one inputs raw data manually, errors and misunderstandings during man–machine communication are inevitable (Akman, 1988). The ultimate solution is that CAD systems must accept substantially reduced yet comprehensive data instead of raw data. For instance, they should accept commands like "Generate an object with such and such properties" or "You have been supplied the minimum requirements, so proceed as you think fit."

Temporality, Ambiguity, and Inconsistency Are Not Allowed

In design, instead of sticking to one particular idea, one may want to experiment with several ideas; e.g., when using top–down stepwise refinement (Wegner, 1984). This brings a time dimension to design. A designer may purge things previously built, introduce things that have not been considered before, or require that the system temporarily forget a particular facet of a design object that for the moment is not being worked on.

Designers also frequently want to separate the structure of a design object from the values of its attributes so that they can first decide about its shape during the conceptual design stage. For example, it is more important to recognize first the topology of a mechanical part if it is going to be inserted in another part. Similarly, a designer may sometimes wish to acknowledge the existence of a point rather than specify its exact location. A similar problem has been studied in database theory where it is known as *null values* (Levesque, 1984).

Unstructured Code Is Commonplace

Software engineering requires that maintainable software be modular both "in the small" (to allow modification of minor components in specific applications) and "in the large" (to allow changes in major components based on, e.g., advances in technology) (Brooks, 1975). It is difficult to observe these characteristics in today's CAD systems, which are not open-ended, because they are not developed on a strong theoretical basis and consist of large amounts of code with unclear specifications, functions, and interactions. More often than not, efficiency is made a central concern and this gives rise to cryptic programs.

Data Exchange May Cause Deterioration of Meaning

When there is a three-dimensional solid modeling system based on, say, Boundary Representation (B-rep), data cannot easily be exchanged, data with another solid modeling system based on, say, Constructive Solid Geometry (CSG). This problem has connections with a fundamental philosophical problem known as *intensional* versus *extensional* descriptions and will be touched on later in this chapter (Tomiyama and ten Hagen, 1987).

Symbolic and Numerical Computing Are Not Coupled

Mechanical engineering systems normally use complex numerical and optimization procedures during design. However, in many cases, insight into the problem is not present. Insight is also needed to interpret the outcome of some computations. As Richard Hamming once said (Kitzmiller and Kowalik, 1986), "The purpose of computing is insight, not numbers." Traditionally, mechanical design systems contain numerical knowledge in the form of bulky libraries of numerical code but nothing else. Users are left alone in analyzing the results of long, confusing computations. Recent research in coupled systems is directed toward integrating the explanatory and problem-solving abilities of expert systems with the precision of numerical computing.

5.3 IIICAD PRINCIPLES

The architecture of IIICAD is directly inspired by, if not derived from, the theory. A central model of the design object, a metamodel, is maintained. From this metamodel more specific models will automatically be derived. Changes to the latter can be promoted to permanent changes in the metamodel and propagate from the metamodel to the other models.

Architecture

The Supervisor (SPV) is at the core of IIICAD and controls all information flow. It adds intelligence to the system by comparing user actions with scenarios (which describe standard design procedures) and by performing error handling. Although the SPV corrects user errors, it does not have the initiative for the design process because IIICAD is envisaged to be a designer's apprentice, not an automatic design environment. *Scenarios* are design process descriptions. External information is further exchanged with the user interface module and with the application modules.

Thus, IIICAD should (1) have a control mechanism to guide the designer,

(2) compare user actions with scenarios, and (3) be a designer's apprentice. Point (1) suggests that the system have control knowledge that is applied to the human designer, point (2) suggests that the system check the actions of the human designer against some control knowledge, and point (3) suggests that the system be trained by the human designer (like a human apprentice who is trained by a master craftsman).

The Knowledge Base (KB) is divided into two parts: an object store where all objects and their internal structures can be found, and a rule store where the relations between several objects are defined in terms of definite program clauses, as in Prolog. The parts of the design artifact and their internal structures are kept in the object store, whereas facts and interrelationships among the parts are furnished by the rule store. Functions "connect" the object store and the rule store.

All information about the design objects, including their attributes and functions (or using the object-oriented terminology, instance variables and methods [Goldberg and Robson, 1983]), as well as the relationships among the objects, will be stored in the KB. It can contain both procedural and declarative descriptions of the artifact that is being designed. Some form is given to this by the object-oriented programming paradigm to describe the objects and by the logic programming paradigm to describe the interrelations among the objects (Bobrow, 1985; Stefik and Bobrow, 1986).

It should be noted that the rule store is most dynamically used during the design process. To provide flexibility to this dynamical behavior, the SPV should provide a mechanism to partition the rule base and group the clauses for specific purposes. The advantage of this is twofold: (1) logical computation becomes efficient because only a relatively small amount of knowledge is passed to the inference mechanism, and (2) consistency is easier to maintain because less information will be considered.

IIICAD has a kernel language called the Integrated Data Description Language (IDDL) used by all system elements. The IDDL is the means used to code the design knowledge and the design objects, and guarantees integrated descriptions systemwide.

In addition to the preceding above principal elements, IIICAD has a high-level interface called the Intelligent User Interface (IUI), which is also driven by scenarios written in IDDL. The IUI accepts messages from other subsystems and sends them to lower level interface systems. Syntactical user errors are processed by the IUI, whereas semantic errors can be processed, as much as this is possible, by the SPV. The goal is to manipulate more user-oriented concepts; this is the basis for more direct communication (i.e., without intermediate command repertoire). To support it, the IUI must be provided with sufficient information about the semantics to recognize the symbols being used, both for input and output. Such a system can be built as a two-layer interface system. The lower layer is a user interface. The upper layer interprets the syntactically accepted constructs of the lower layer on the basis of declarative knowledge that has been made available by the SPV. Semantic recognition can drive typical user interface processes such as anticipation, error correction, default handling, and focusing.

The Application Interface (API) secures the mapping between the central model descriptions about a design object and the individual models used by the application programs. The following list is a first-order approximation to the desirable application programs: conceptual design systems to handle vague information, consultation and problem-solving systems for engineering applications, basic/detailed design systems coupled with geometric modelers, engineering analysis systems such as finite element packages, and product modelers.

User Interface

There are several useful ways of looking at intelligent CAD user interface architecture. The following dichotomies are quite common (David, 1987): CAD versus automated design (AD), designer's apprentice versus autonomous design system, and glass box versus black box.

The boundary between CAD and AD is indeed hard to delineate. One view is that the ultimate aim of design computerizationt design is to arrive at automatic design systems which can compete with and even surpass the best human designers. However, the interactive nature of design will probably dictate that for a long time to come, CAD as man–machine cooperation must dominate. The same holds true for apprentice versus autonomous systems. An apprentice system has less hard-wired knowledge than an autonomous system, but it knows better how to interact and has a generic model of design. An autonomous system is powerful for narrow domains. It is relatively easy to extend an apprentice system by teaching it new "skills." It is unwieldy to extend an autonomous system because its very constitution warrants myopia.

A more natural look at these dichotomies is via the metaphor glass box versus black box. If a CAD system has a glass box structure, then the user can, at any time, look through it to see partial results and processes. On the other hand, a black box system resembles a batch processing environment—one submits tasks to be executed and the system reports back with results.

The seasoned researchers of CAD may remember those times when ideas such as "general CAD" became fashionable (Jacquart *et al.*, 1974). Today, demands for integration suggest reconsideration of that sweeping panorama of design. The view which regards design as a large collection of intelligent tools is different from the view which regards a design system as a framework. The intelligent tool approach assumes that if one has a cooperating set of experts who can communicate with each other, then one can solve many problems. The framework approach regards the "shell" of the design systems as their biggest advantage; the domain-specific issues can be dealt with separately, using the facilities provided by the shell.

5.4 STRATEGIES

There are two caveats in the development of IIICAD. The first is that everything should be based on a formal theory to avoid the infamous hacker trap, i.e., assorted features that overwhelm the user. The second caveat is to be careful about software development: while knowledge engineering is different from software engineering, the KB will still have to be maintained (Ramamoorthy *et al.*, 1987).

A well-founded design theory may serve as a basis for specification and implementation of intelligent CAD systems. To be useful for this purpose, a theory must be realistic in the sense that it has a close relation to design practice; it must describe design processes as they are in practice or as users would like or are accustomed to. A design theory must also have a logical basis so that there are guarantees that a system, developed according to the theory, will take sound steps. Thus, logical representations of incomplete descriptions of objects, patterns of reasoning involved in design (Simon, 1977), and multiple worlds (which are described later) are to be allowed.

There are three ontological aspects of design: processes, models, and activities. A theory of CAD is then an aggregate of the following:

(1) A theory that describes the design processes and activities (Tomiyama and Yoshikawa, 1987).
(2) A theory that deals with the models of design objects (Akman and ten Hagen, 1989). For this discussion, this must be a theory of machines (Minsky, 1967); in VLSI design, it would be a theory of VLSI, and so on.
(3) A metalevel theory that describes existing knowledge about design.

The Role of Logic

In General Design Theory (Tomiyama and Yoshikawa, 1987), a design process is regarded as a mapping from the function space onto the attribute space. Both spaces are defined on an entity set. A design process evolves about a metamodel. During design, new attribute descriptions will be added (or existing ones will be modified) and the metamodel will converge to the design solution. In other words, design specifications will initially be presented in functional terms and the design will be completed when all relevant attributes of the artifact are determined so as to be able to manufacture it. To further illustrate the design process, three major components need to be recognized: (1) entities, (2) attributes of entities, and (3) relationships among entities. A design process is thus a sequence of small steps (forward and backward) to obtain complete information about these components.

It was mentioned before that to control the stepwise refinement of the design process, a designer needs to express unknown, uncertain, and temporal information about the design object. This can be accomplished using an amalgam of intuitionistic logic, modal logic, and temporal logic (Veth, 1987). It is taken for granted that descriptions of a design object are given by a set of propositions (such as well-formed formulas in first-order logic). A simplified view of design is then as follows. A sequence of metamodels is generated from an initial specification. If they cannot pass a feasibility check, then a compromise is made or backtracking is applied. The models derived from the metamodels are, in the meantime, evaluated for consistency. Thus,

Specification: s = T[0] - - > . . . - - > T[n] - - > . . . - - > T[N] = g
Metamodel: M[0] - - > . . . - - > M[n] - - > . . . - - > M[N]
Propositions: q[0] - - > . . . - - > q[n] - - > . . . - - > q[N]

Here, s is the original design specification and g is the design goal. Each design step has an associated set of propositions that are denoted by q[n]. Two central concerns are (1) how to choose q[n] (i.e., how to proceed with design) and (2) what if it is discovered that ~q[n] (i.e., how to deal with contradictions). The following definitions are therefore given:

q[n] = p[0] & . . . & p[i] & . . . & p[k]

where each p[i] is a fact concerning the metamodel,

q[n], C[n] | – m[n]

where |– is the syntactic turnstile, m[n] is a model, and C[n] is control knowledge, and

q[n], m[n], D[n] | – r

where D[n] is detailing knowledge and r is a proposition that should be added to q[n] to arrive at the next description, q[n + 1]. Knowledge appearing right before the syntactic turnstile is used in the derivations as metaknowledge. With this notation, the following classification of design suggests itself:

Invention: Given s, determine q[n], C[n], D[n], and g.
New product development: Given s and C[n], determine q[n], D[n], and g.
Routine design: Given s, C[n], and D[n], determine q[n] and g.
Parametric design: Given s, q[n], C[n], and D[n], determine g.

Logic with modes of truth, i.e., modal logic with necessity and possibility, can be used as a notational tool. Here, there are not only affirmations that some proposition is true, but also stronger ones, such as that p is necessary (denoted Lp) and weaker ones, such as that p is possible (denoted Mp). If p is asserted to be necessary, then it can also be asserted that p is true. If p is asserted as true, then we can also assert p as possible. The following equalities can be used:

Lp = ~M~p
Mp = ~L~p

Three well-known modal logic systems are T, S4, and S5. T is characterized by the axiom schemata of the predicate calculus plus the following:

Lp = > Lp
L(p = > q) = > (Lp = > Lq)

There is a rule of inference (referred to as necessitation) in T which allows derivation of Lp from p. S4 is obtained by the addition of

$$Lp => LLp$$

to T, and S5 by by the addition of

$$Mp => LMp$$

For temporal logic, let t >> p (t << p) denote that p holds after (before) time t. Let t[1]...t[2] denote the time interval whose end points are t[1] and t[2]. Then,

$$t >> \sim p = \sim (t >> p)$$

$$t >> (p|q) = (t >> p|(t >> q)$$

$$t >> (p \& q) = (t >> p) \& (t >> q)$$

$$t[1] \ldots t[2] >> p = (t[1] >> p) \& (t[2] << p) \& (t[1] < t[2])$$

Using temporal logic, inference control (usually called metalevel control) can be described. For instance, in Prolog, the order of rules matter (Bobrow, 1985). In general, this knowledge is embedded in the interpreter of the language. By disclosing this control, more supple control may be introduced. For example, D[n] may be a set of rules of the following sort:

$$(t[1] >> q[1]) \& (t[2] >> q[2]) \& (t[1] < t[2]) => (t[2] >> q[3])$$

$$(t[1] >> q[1]) \& (t[2] >> q[2]) \& (t[1] > t[2]) => (t[2] >> q[4])$$

That is, if q[1] holds after t[1], q[2] holds after t[2], and t[1] is earlier than t[2], then a new property, q[3], holds after t[2]. Otherwise, another property, q[4], holds.

Intuitionistic logic can be blended with temporal logic. Let t[p] >> p be defined as tt (true). Now, introducing a logical symbol uu (unknown), intuitionism can be formalized in terms of temporality:

$$t[p] << (p| \sim p) = uu$$

$$t[p] >> (p| \sim p) = tt$$

Naive Physics

In machine design, it is not yet precisely known by which symbolism one should describe the functions of machines. There is, however, a view that functions can be represented in terms of the physical phenomena that the machine exhibits (de Kleer and Brown, 1984). Therefore, the representation of functions can be reduced to the representation of physical phenomena and the qualitative reasoning about them.

> When a particular machine is described to us, we do not first ask questions about its material construction. Given an engineering drawing, a circuit diagram, a patent description, something must first convince us that we understand how it works *in principle*. That is, we must see how it is "supposed" to work. We inquire only later whether this member will stand the stress, or whether that oscillator is stable under load, etc. But the *idea* of a machine centers around some *abstract* model or process.... The abstract idea of a machine, e.g., an adding machine, is a *specification* for how a physical object *ought* to work. If the machine that I build wears out, I censure *it* and perhaps fix it. Just as in physics, the parts and states of the physical object are supposed to correspond to those of the abstract concept. But in contrast to the situation in physics, we criticize the *material* part of the system when the correspondence breaks down. [Emphasis removed in various places.] (Minsky, 1967, pp. 5–6)

In order to give the reader a better appreciation of the kind of physical reasoning being sought, a quick look at geometric reasoning is provided. This is also an example of general knowledge, which needs to be applied in specific situations. For geometry, the emphasis is on generating the appropriate representation in order to answer queries as directly as possible. One obvious solution is to encode the essential knowledge procedurally. This kernel is surrounded by declaratively described methods about how it is to be used. A given specific problem, described declaratively, is matched with the "geometry base" and the nearest answer is used as a starter. This is in the spirit of de Kleer's (1975) Random Access Local Consequent Methods (RALCMs).

In qualitative physics, the reduction of information is arrived at by creating an abstract layer (the naive or qualitative layer) which may, strictly speaking, be incorrect but sufficiently correct for the problem at hand. Naive physics observes that people have a different kind of knowledge about the physical world. This knowledge can best be described as common sense, and is attained after years of interaction with the world (McCarthy, 1968; Akman *et al.*, 1989). Naive physics ideas are useful in machine design and should be codified in the IIICAD system. To this end, the proposal in Hayes (1985) is followed with the hope of capturing naive physics in logic. Additionally, qualitative physics is used as a mathematical tool based on symbolic manipulations.

Consider the following concepts underlying "change" in physical systems: state, cause, equilibrium, oscillation, force, and feedback. Qualitative physics regards these concepts in a simple qualitative way. It maps continuous variables to discrete variables, taking only a small number of values (+, −, 0). Accordingly, differential equations are mapped to qualitative differential equations (also known as confluences) (Kuipers, 1986).

Qualitative techniques may cause ambiguities. Assume that a certain quantity M varies with N/L, i.e., M is proportional to N/L. If N increases (decreases) while L remains constant, then M also increases (decreases). However, a finer knowledge of the individual changes is required if we are told that both N and L increase (or decrease). The techniques of order of magnitude reasoning are designed to handle precisely this kind of problem without requiring knowledge of the numerical values involved (Raiman, 1986).

In qualitative reasoning, a "mechanistic" world view is adopted. This asserts that every physical situation is regarded as a device made up of individual components, each contributing to the overall behavior. Nevertheless, the laws of the substructures may not presume the functioning of the whole: the principle of no-function-in-structure (de Kleer and Brown, 1984). Additionally, assumptions that are specific to a particular device should be distinguished from the class-wide assumptions that are common to the entire class of devices. A simplistic view of modeling devices comprises three kinds of constituents: materials, components, and channels. Channels transfer material from one component to another.

After modeling a device, designers can reason about it. Envisioning starts start with a structural description to determine all possible behavioral sequences, momentarily forgetting about the real values of the problem variables while trying to see all possible outcomes (Kuipers, 1986). Naive physics concepts are required for design because a design object will have a physical existence and, accordingly, obey natural laws. If designers want to create designs that correspond to physically realizable design objects, then they will have to refer to naive physics notions (Akman *et al.*, 1990). Furthermore, if they want to reason about a design object in its destined environment, envisioning, simulation, and diagnostics tools need to be available (Forbus, 1984).

Intension Versus Extension

In order to discuss design, the entities, their properties, and the relationships among them need to be described. In an extensional description method, the fact that an entity *e* has property *p* is described by p(e) and the fact that entities e and f stand in relationship r is described by r(e,f). In an intensional description method, these two facts can be represented by e(p) and rel(e,r,f), respectively. Extensional descriptions do not imply any preconceptions, whereas intensional descriptions make assumptions. When an intensional description has to be changed, it results in a modification of those predefined conditions. For instance, a shaft might be represented intensionally by shaft(diameter,length,bearing_1,bearing_2). If we now want to add a new attribute, such as power, the addition results in a redefinition of the predicate shaft. On the other hand, an extensional description might consist of the following facts: shaft(s), equal(diameter(s),D), equal(length(s),L), supported_by(s,b_1), supported_by(s,b_2), bearing(b_1), and bearing(b_2).

Thus, in an extensional description we need to write numerous obvious descriptions. However, modifying such a representation is easy: just add or delete facts. On the other hand, an intensional description is difficult to modify but shows better performance. CAD applications demand a flexible data description scheme that is easy to modify (Tomiyama and ten Hagen, 1987). Independent multiple views of a design object demand independent small partitionings of the database. This is easily achieved by an extensional description method because only the relevant facts have to be picked up.

5.5 RAPID PROTOTYPING

IIICAD will evolve over time, with the definition of its purpose being refined as it becomes a fuller system. This suggests that recent techniques of software engineering such as exploratory programming (Ramamoorthy *et al.*, 1987) and rapid prototyping are not to be overlooked. These techniques are more permissive than the rigid method of formal specifications in that they advocate iterative enhancement. This leads to an evolutionary software life cycle. One starts with a skeletal implementation and adds new modules until the system is reasonably complete for demonstration. In rapid prototyping, the interest is in perceiving a glimpse of a future system in order to assess its strengths and weaknesses (Brooks, 1975).

Smalltalk and Loops

With regard to exploratory programming and rapid prototyping, Smalltalk is a fine implementation tool which is appreciated for two reasons. First, is the emphasis of Smalltalk on interactive graphics and its sophisticated user interface (Goldberg and Robson, 1983). Second, Smalltalk is not just an isolated programming language; it is a programming environment. For instance, the ability in Smalltalk to inspect any data structure recursively gives the programmer a great ability for exploratory programming. Smalltalk's insistence to express everything as objects and messages using a simple syntax is conceptually powerful and consistent.

Experience with Smalltalk also suggests that it is not yet perfect (Nielsen and Richards, 1986). First and foremost, using it effectively is not an easy task. Long chains of message passing make it difficult for the novice to understand how some action takes place. The scale of the system and its generalist design may frustrate even the professionals with a wide background in other programming languages. The class hierarchy is rigid. The inheritance mechanism, even when it is enriched with multiple inheritance, is seen as a drawback for CAD systems (Veth, 1987). A scheme called *delegation* is currently being popularized for pliable sharing of knowledge in object-oriented systems. In its most common form, inheritance is a way to arrange classes in a tree so that they inherit methods from other classes. Delegation is more general: an object can delegate a message to an arbitrary object rather than being limited to the paths of a class hierarchy (or lattice, in case of multiple inheritance).

Loops is also becoming increasingly popular due to its multiparadigm nature (Stefik *et al.*, 1986). It supports a mechanism for annotated values. This helps programmers monitor arbitrary values without previously planning such access. There are two kinds of annotated values: property annotation and active value. The latter associates with any value a demon. The former associates with any value an optional property list.

From a CAD viewpoint, another useful construct of Loops is as a composite object. Composite objects contain other objects as parts. A car may be described structurally as consisting of a body, a mechanical system, and an electrical system. The body has four doors, six windows, and so on. Parts can contain other parts, e.g., a door has an ashtray and a handle. Objects may belong to more than one structure, e.g., the power window controller can be viewed as a component of the mechanical system or the electrical system.

Loops introduces *perspectives*, which are a form of composite object and which provide a way to implement the multiple views of the same design object. Thus a pressure regulator may be represented using perspectives like mech_assembly, disp_object, and func_box. The first two perspectives both treat coordinates, but with different interpretations. In the first perspective, the coordinates refer to physical dimensions of the regulator to be manufactured whereas in the second perspective, they are just measures on a display screen. The third perspective concerns the functioning of the regulator as a control theoretic device with feedback. Loops offers a remedy to the complicated method hierarchy of Smalltalk. In Loops, a method is defined over a set of classes. This means that methods do not belong to a particular class.

5.6 IDDL: A DESIGN BASE LANGUAGE

Every human-controlled production process for some artifact contains numerous tasks. Only a few of these are supported by computerized design tools. IDDL contains new constructs which support writing code for a wide range of design tasks.

Design is a complex activity. Consequently, the motives for improving design support can be quite different, e.g., making designing more economical, improving the quality of designs, and making designing more independent of expertise. Such general objectives have to be analyzed in order to distill a number of problem areas for which adequate scientific and technical treatment would provide better support. The problems are classified in such a way that language constructs needed for formulation of design support methods can be gradually introduced. With each new facility added, both the language and the application domain will grow.

Representational Issues

A typical CAD system must have three major components: (1) a design object component, (2) a design process component, and (3) a user interface component. Improvements in the representation of design objects is a necessity for integration of design tasks. On the other hand, a satisfactory design process representation is necessary for enlarging the scope of design tasks. A good user interface is crucial to be able to directly specify domain-specific terminology. This requires semantic processing of user transactions. The user interface must be directly coupled to the object and process representations.

A design process representation is a list of actions, which, when executed in a meaningful order, may specify a design object. Each action is preceded by a condition that expresses when the action is possible: in this case, execution will contribute something to the design object. Actions can be structured in design scenarios. A scenario exists for each design task. The status of the design process can be explicitly maintained in a rule base. The status of the artifact is maintained in the same rule base. Status information and user input together form the operands of the conditions that precede the action rules. The conditions, therefore, represent the control structure of a scenario. The control structure of the entire design process is obtained by adding to this scenario structures which consist of a subscenario hierarchy and the option to activate several scenarios concurrently.

The design object is represented by atomic objects and object relations, which can be used to create composite objects. Hence, all object structuring is obtained explicitly by asserting relations. These relations together make up the object status previously mentioned. Equally, the design process status can be represented as object relations. This makes it possible to take a close look experimentally at how a design object and its generating process are interrelated.

The part of an artifact and the design process status that is relevant for a given scenario is called a *world*. Much of the potential of IIICAD will depend on the flexibility of the world mechanism being built. A world can create a particular, restricted view on the artifact. In a world, one can execute the actions of the corresponding scenario more efficiently.

All automated parts of the design process now follow the same scheme: If a condition is defined over a process status, and an object status holds, then action is taken as specified in the consequence of the condition. This action may lead to a new status for both the object and the process, for which another rule may apply, and so on. This process comes to a halt when new input from the user is needed to continue.

The KB consists of the collection of all objects that are created and the collection of all relationships that hold among these objects. In addition, every object has its current status, e.g., the values of its attributes. Most objects have attributes that are represented by functions which map objects onto attribute values. A set of such functions forms the "inside" structure of an object.

One can identify a particular configuration of objects and their relationships, e.g., a world. Worlds as a configuration of objects and their relationships partition the KB in a straightforward manner. At any point in time, a functional separation of the KB is achieved by calling the set of available objects an "object base" and the relationships among them a "rule base."

Requirements for IDDL

The IDDL will be the base language on which the subsystems of IIICAD will be built. This section summarizes the requirements for IDDL (Veth, 1987).

The IDDL, as a language used to construct the knowledge base, should support incremental programming and easy maintenance. It should be able to describe not only design objects but also design processes. It must embed the stepwise nature of the design process and should be able to describe the knowledge to detail a metamodel, to check metamodel feasibility, and to control the detailing process. It also should allow a myriad of views of a design object; these views are possibly independent but still correlated.

The IDDL should incorporate positive and negative information, known and unknown, and modalities such as "necessary" and "possible." This requires that

IDDL consolidate logics such as modal, intuitive, and temporal. Inconsistencies need to be resolved with some sort of truth maintenance when transferring to the next metamodel. In other words, design problem solving can also be seen as selecting among plausible alternatives. One naive method for exploring the search space is to enumerate its points and test. It is more advantageous to find solutions that meet some criteria of acceptability. This implies the ability to compare partial solutions. A world is demarcated by a set of assumptions and defines a possible solution. Different assumptions on the same range are clearly contradictory. That is, the KB that records the different worlds will be inconsistent. A Truth Maintenance System (TMS) is needed to work safely with an inconsistent database. It records the inferences made by the problem-solver and then avoids repeating the same inferences. Once a contradiction is found, the world it belongs to is no longer considered. Via dependency-directed backtracking, a TMS is able to remove contradictions from the database and rebuild a consistent world (de Kleer, 1986).

To control the behavior of the system, the IDDL must have metaknowledge that selects which rule to apply at a certain moment. It should also have a "focusing control" mechanism which can create a small world where it is clearly defined what kind of information is accessible. From the viewpoint of interactive design, it is desirable for the human designer to be able to mark intermediate design stages, and to later go back to examine or resume from there. Consequently, the stages of design evolution must be representable on the level of IDDL.

Design produces intermediate results that are incomplete and even inconsistent during a time spanning a series of transactions. The design object's representation must allow assumptions to be used for the evolution of the design object. The IDDL, using nonmonotonicity, should be able to retract assumed (but eventually unconsidered or disproved) propositions.

In spite of the slight weaknesses cited earlier, object-oriented programming languages deliver extensibility, flexible modifications of code, and reusability. Important issues in object-oriented programming are data encapsulation and information hiding. Thus an object can be regarded as an independent program that knows everything about itself. It is an additional benefit of object-oriented programming systems that they offer incremental compilation, dynamic binding, system building tools, and good user interfaces. Therefore, object-oriented programming is a good choice for creating IIICAD. On the other hand, logic programming is powerful because it can reflect the reasoning process most naturally. IDDL uses the logic programming paradigm to express the design process, whereas the object-oriented programming paradigm is utilized to express design objects.

A Multiworld Mechanism

During the course of design, it is vital that an intelligent CAD system allows the designer to represent the design object in various ways. These various descriptions are called *models of the design object.* Four examples of such models are

(1) A functional specification of the design object in terms of the constraints that should be met

(2) A geometrical model which creates a visual representation of the design object
(3) An FEM (finite element method) model enabling strength predictions to be made on complex mechanical constructions
(4) A cost analysis model to calculate the manufacturing costs

Models are able to communicate with each other by means of the metamodel. This is a central description of the design object to which all models refer to and depend on. All changes that occur in a certain model are propagated through the metamodel to all other models, which are then updated appropriately. Although the different models may use different languages internally (e.g., C for the cost analyzer, a CSG language for the solid modeler), the interface between a metamodel and the other models is realized in IDDL. This is achieved by the multiworld mechanism (Akman *et al.*, 1988). A model is created by a call to a scenario which in turn opens a world. A world is a part of the design object description together with some information that belongs to the particular model that is created. The multiworld mechanism lets the designer open several worlds simultaneously; in other words, several models may be active at the same time. For instance, the designer may input new constraints and examine the results of these in a CSG model and an FEM model concurrently.

Moreover, the multiworld mechanism allows the designer to create multiple descriptions of the design object in parallel. Here, a distinction is made between dependent versus independent worlds. *Dependent* worlds result in a unique design object description; they reflect the same metamodel. *Independent* worlds, however, can result in different design object descriptions; they do not reflect the same metamodel. In a nutshell, dependent worlds are used to create multiple views of the design object, whereas independent worlds give rise to alternative design solutions.

The multiworld mechanism is embedded in the scenario part of IDDL. Scenarios specify how many worlds exist simultaneously and how they relate to each other. Special care is taken for constructs that close or update a world, thereby transferring some of its properties to the metamodel. This control mechanism also checks the validity of the worlds with respect to the metamodel and propagates changes in the metamodel to all applicable worlds.

5.7 CONCLUSION

IIICAD is a unifying framework for describing and using design knowledge. Theoretical ideas of AI such as naive physics, formal tools such as logic, and practical software techniques such as prototyping are essential to IIICAD. IIICAD clarifies, on an abstract level, what the design process is in relation to the design object. No distinction is drawn between the parts of the process carried out by the designer and by the assisting program.

Smalltalk is used as an implementation language on top of which IDDL and a Prolog-like inference engine can be built. Except for the hierarchical constructs, IDDL can take advantage of the already existing object-oriented constructs of

Smalltalk in order to represent the objects in the object base. The hierarchical structure of the objects can then be fully implemented as clauses in the rule base.

REFERENCES

Akman, V. 1988. Geometry and graphics applied to robotics. In *Theoretical Foundations of Computer Graphics and CAD*, ed. R. A. Earnshaw, pp. 619–638. Berlin: Springer-Verlag.

Akman, V., and ten Hagen, P. J. W. 1989. The power of physical representations. In *Intelligent CAD Systems 2: Implementational Issues*, eds. V. Akman, P. J. W. ten Hagen, and P. Veerkamp, pp. 170–194. Berlin: Springer-Verlag.

Akman, V., and Tomiyama, T. 1989. The role of logic and commonsense reasoning in intelligent CAD. In *Proc. of 3rd IFIP WG 5.2 Workshop on Intelligent CAD*, Osaka, Japan, pp. 15–20.

Akman, V., ten Hagen, P. J. W., Rogier, J. L. H., and Veerkamp, P. 1988. Knowledge engineering in design. *Knowledge-Based Systems* 1:67–77.

Akman, V., Franklin, W. R., and Veth, B. 1989. Design systems with common sense. In *Proc. of 3rd Eurographics Workshop on Intelligent CAD Systems: Practical Experience and Evaluation*, Texel, The Netherlands, pp. 317–322.

Akman, V., Ede, D., Franklin, W. R., and ten Hagen, P. J. W. 1990. Mental models of force and motion. In *Proc. of IEEE International Workshop on Intelligent Motion Control*, ed. O. Kaynak, pp. 153–158. IEEE Press.

Bobrow, D. 1986. If Prolog is the answer, what is the question? or What it takes to support AI programming paradigms. *IEEE Transactions on Software Engineering* 11:1401–1408.

Bobrow, D., Mittal, S., and Stefik, M. 1986. Expert systems: Perils and promise. *Communications of the ACM* 29:880–894.

Brooks, F. P.1975. *The Mythical Man-Month.* Reading, Massachusetts: Addison-Wesley.

Brown, D.C., and Chandrasekaran, B. 1986. Knowledge and control for a mechanical design expert system. *IEEE Computer* 19:92–100.

David, B. T. 1987, Multi-expert systems for CAD. In *Intelligent CAD Systems 1: Theoretical and Methodological Aspects*, eds. P. J. W. ten Hagen and T. Tomiyama, pp. 57–67. Berlin: Springer-Verlag.

Doyle, J. 1985. Expert systems and the myth of symbolic reasoning. *IEEE Transactions on Software Engineering* 11:1386–1390.

Dyer, M.G., Flowers, M., and Hodges, J. 1986. EDISON: An engineering design invention system operating naively. *Artificial Intelligence in Engineering* 1:36–44.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* 24:85–168.

Forbus, K. 1988. Intelligent computer-aided engineering. *AI Magazine* 9:23–36.

Goldberg, A., and Robson, D. 1983. *Smalltalk-80: The Language and Its Implementation.* Reading, Massachusetts: Addison-Wesley.

Hayes, P. J. 1977. In defense of logic. In *Proc.of 5th International Joint Conference on Artificial Intelligence* (IJCAI-77), Cambridge, Massachusetts, pp. 559–565.

Hayes, P. J. 1985. The second naive physics manifesto. In *Formal Theories of the Commonsense World*, eds. J. R. Hobbs and R. C. Moore, pp. 1–36. Norwood, New Jersey: Ablex.

Jacquart, R., Regnier, P., and Valette, F. R. 1974. GERMINAL: Towards a general and integrated system for computer aided design. In *Proc. of 11th Design Automation Workshop*, Denver, Colorado, pp. 352–358.

Kitzmiller, C. T., and Kowalik, J. S. 1986. Symbolic and numerical computing in knowledge based systems. In *Coupling Symbolic and Numerical Computing in Expert Systems*, ed. J. S. Kowalik. Amsterdam: Elsevier.

de Kleer, J. 1975. Qualitative and quantitative knowledge in classical mechanics. Technical

Report AI-TR-352, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28:127–162.

de Kleer, J., and Brown, J. S. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7–83.

Kuipers, B. 1986. Qualitative simulation. *Artificial Intelligence* 29:289–338.

Lansdown, J. 1988. Graphics, design, and artificial intelligence. In *Theoretical Foundations of Computer Graphics and CAD*, ed. R. A. Earnshaw, pp. 1153–1174. Berlin: Springer-Verlag.

Levesque, H. J. 1984. The logic of incomplete knowledge bases. In *Conceptual Modeling*, eds. M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, pp. 165–186. New York: Springer-Verlag.

McCarthy, J. 1968. Programs with common sense. In *Semantic Information Processing*, ed. M. Minsky, pp. 403–418. Cambridge, Massachusetts: MIT Press.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines.* Englewood Cliffs, New Jersey: Prentice-Hall.

Mittal, S., Dym, C. L., and Morjaria, M. 1986. PRIDE: An expert system for the design of paper handling systems. *IEEE Computer* 19:102–114.

Moore, R. C. 1985. The role of logic in knowledge representation and commonsense reasoning. In *Readings in Knowledge Representation*, eds. R. J. Brachman and H. J. Levesque, pp. 336–341. Los Altos, California: Morgan Kaufman.

Mostow, J. 1985. Toward better models of the design process. *AI Magazine* 6:44–57.

Nielsen, J., and Richards, J. T. 1986. Comments on the learnability and usability of Smalltalk for casual users. Technical Report RC-12080, IBM T. J. Watson Research Center, Yorktown Heights, New York.

Purcell, P. A. 1988. Computer environments for design and designers. *Design Studies* 9:144–149.

Raiman, O. 1986. Order of magnitude reasoning. In *Proc. of American Association for Artificial Intelligence (AAAI-86)*, Philadelphia, 100–104.

Ramamoorthy, C., Shekhar, S., and Garg, V. 1987. Software development support for AI programs. *IEEE Computer* 20:30–40.

Simon, H. A. 1977. The structure of ill-structured problems. In H. A. Simon, *Models of Discovery (and Other Topics in the Methods of Science)*, pp. 304–325. The Netherlands: D. Reidel, Dordrecht.

Simon, H. A. 1979. The science of design: Creating the artificial. In H. A. Simon, *The Sciences of the Artificial*, pp. 55–83. Cambridge, Massachusetts: MIT Press.

Stefik, M., and Bobrow, D. 1986. Object-oriented programming: Themes and variations. *AI Magazine* 6:40–62.

Stefik, M., Bobrow, D., and Kahn, K. 1986. Integrating access-oriented programming with a multiparadigm environment. *IEEE Software* 3:10–18.

Sutherland, I.E. 1963. SKETCHPAD: A man-machine graphical communication system. In *Proc. of AFIPS Spring Joint Computer Conference*, pp. 329–346. Baltimore: Spartan Books.

Tomiyama, T., and ten Hagen, P. J. W. 1987. Representing knowledge in two distinct descriptions: Extensional vs. intensional. Technical Report CS-R8728, Center for Mathematics and Computer Science, Amsterdam.

Tomiyama, T., and Yoshikawa, H. 1987. Extended general design theory. In *Design Theory for CAD*, eds. H. Yoshikawa and E. A. Warman, pp. 95–130. Amsterdam: North-Holland.

Veth, B. 1987. An integrated data description language for coding design knowledge. In *Intelligent CAD Systems 1: Theoretical and Methodological Aspects.* eds. P. J. W. ten Hagen and T. Tomiyama, pp. 295–313. Berlin: Springer-Verlag.

Wegner, P. 1984. Capital-intensive software technology. *IEEE Software* 1:7–45.

Yeomans, R. W., Choudry, A., and ten Hagen, P. J. W. 1985. *Design Rules for a CIM System.* Amsterdam: North-Holland.