A fundamental and theoretical framework for an intelligent CAD system

V Akman, P J W ten Hagen* and T Tomiyama†

Currently, there exists a line of research in mechanical CAD that is directed towards using AI and knowledge engineering ideas, but truly unifying approaches in this respect are lacking. The authors hope to fill this gap via a logic-based, theoretical approach. A formulation is given for how a designer's apprentice (or a design workbench) can be established. The relevance of naive physics and commonsense reasoning in machine design are demonstrated. As for the software development methodology several aspects of software engineering are considered. A design base language which is built upon logic programming and object-oriented programming paradigms is proposed.

intelligent CAD, design theory, logic, knowledge representation, naive physics, commonsense reasoning, object-oriented programming, logic programming

*The paradox is now fully established that
the utmost abstractions are the true weapons
with which to control our thought of concrete fact.
—A. N. WHITEHEAD*

CAD programs assist designers to specify artifacts, e.g. a house, a machine, or an electronic circuit. The assistance can range from mere registration of the design results to analysing the proper functioning of the designed object, maybe through simulation (or more properly, envisioning)¹. More advanced forms of assistance include problem-solving activities such as optimization, routing, and even suggesting a solution based on the given specifications. The latter activities become more dominant when 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 (e.g. analysis, synthesis, evaluation, and so on) by decomposition into subprocesses for parts of the partitioned design. Moreover, there may be forms of backtracking (undoing decisions), iteration, detailing, etc.

In some sense, the design process can be characterized by the way it will interact 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 the design transactions. Following this, a coherent solution for integrating the various design activities as well as user interface issues can be researched.

Much of a designer's activities consist of manipulations of the design object to add new information, to change or to inspect it. In advanced systems, the context in which these take place may vary over time. 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, changeable, etc. Much of the interaction between process and object representations is dependent on status information. In particular, status information must be allowed to be incomplete, inaccurate, or even inconsistent (in a given intermediate situation).

This paper considers mechanical CAD. It takes mechanical part design (machine design) as the domain of discussion for this 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 (e.g. chairs, bicycles) to the complex and sophisticated (e.g. cars, aircrafts). It multiplies the productivity many times and produces, using CIM techniques², robust products usually free from the errors of fallible human designers. (I've yet to meet any industrial manager who would agree with such an optimistic view,³ was the reaction of a referee of this paper.)

However, the practice of CAD in the industry is not without problems. It is commonly accepted that current

Department of Computer Engineering and Information Sciences, Faculty of Engineering and Science, Bilkent University, P. O. Box 8, 06572 Maltepe, Ankara, Turkey.

* Department of Interactive Systems, Centre for Mathematics and Computer Science (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.

† Department of Precision Machinery Engineering, Faculty of Engineering, The University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan.

Paper received: 25 May 1989. Revised: 30 January 1990

CAD systems are large software systems that are difficult to master and inflexible to adapt to growing needs. They can only deal with limited domains and occlude attempts at integration. They do not cleanly support a crucial ingredient of design, namely, interaction. What is worse is that they lack a distinguishing characteristic of human designers: they have little or no *intelligence*³. 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 huge quantities of design data and equally huge collections of design experience.

Efforts to provide a wider perspective on CAD are now underway at several companies, universities, and research institutes. Since design is a highly mental* activity, researchers have long felt the need to make CAD systems more intelligent. There is now an appreciable amount of ground-breaking work in this area which is commonly known as intelligent CAD. The reader is referred to some representative articles⁴⁻¹² and recent books¹³⁻¹⁷.

Our work, currently in progress at the Centre for Mathematics and Computer Science (CWI), Amsterdam, is directed towards contributing to the theory and practice of intelligent CAD systems. Our research project, titled 'Intelligent Integrated Interactive CAD' (IIICAD), regards AI techniques and knowledge engineering tools as fundamental to a design system which is hoped to be more substantial than expert systems^{6,8}. This paper gives a brief yet quite complete overview of the IIICAD philosophy. As such, it can also be regarded as a position paper on the capabilities future intelligent CAD systems should incorporate.

PROBLEMS WITH EXISTING CAD SYSTEMS

Mechanical CAD has evolved rapidly. Ivan Sutherland's revolutionary SKETCHPAD¹⁸ in the 1960s generated much enthusiasm for using interactive graphics in engineering. (SKETCHPAD allows geometric shapes to be represented and various conditions to be specified over these shapes in terms of constraints, to which they then obey.) This in turn motivated a stormy decade when turnkey 2D drafting systems gradually replaced the drawing boards in the professional environment. Finally, the dust settled with their general acceptance by the industry; 3D modelling systems became available and they were also widely accepted by the industry as indispensable tools in product development.

Nevertheless, borrowing an analogy from Bobrow *et al.*¹⁹, it is not unjust to claim that all these systems follow the 'low road' approach. They are mostly programmed in an *ad hoc* manner using languages like

Fortran and regard design from a singular viewpoint, e.g. as a mainly geometric activity. (The authors understand that this may not be a fair description of the way several recent CAD software systems have been produced.) Thus, despite their popularity, there are many problems with the existing CAD systems. As shall be seen in the following, even recent research cannot escape various inherited pitfalls. In this section, these problems are highlighted in detail.

Methodology

As with any other discipline, a critique of other approaches to CAD systems development presupposes a starting point, i.e. a vision of design. Briefly, we see design as an intellectual activity performed by human designers. We think that the essential thing in a designer is that s/he builds us his/her world. Thus, we believe that design systems should provide a framework where designers can exercise their faculties at large. With this view, we support, more or less, the idea of apprentice (assistant) – as opposed to autonomous – CAD systems. More consideration of this rather crude division will be given later. Suffice it to say that we carefully distinguish our view from other common views about the nature of design such as:

- Design is a routine process.
- Design is an inventive (innovative) process.
- Design is a problem-solving process.
- Design is a decision-making process.
- Design is an optimization process.

The first view above treats design as a rather straightforward activity where the designer selects from a previously known set of well-understood alternatives. A recent example is the AIR-CYL system⁹. Clearly, this view is ingenuous and does not reflect the intricate nature of design. The second view embraces the exciting ideas of artificial intelligence (AI) in order to create novel devices by using knowledge of naive physics, qualitative reasoning, planning, analogical reasoning, 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 probably only a toy system²⁰. While the remaining views in the above list regard design partly as an activity requiring intelligence and creativity, they underplay its holistic nature. They are mostly implemented as expert systems¹⁹ which solve specific problems of a specific design process. An example of this 'middle road' approach is the PRIDE system, which nevertheless is an interesting system with useful ideas behind it¹⁰. An annoying and often cited problem with expert systems is that they cannot deliver genuinely expert performance because they have no underlying mechanism to understand what is going on^{1,21,22}. 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 contributed to the emergence of terms like *deep* and *shallow* (although there are several drawbacks to such usage) in the AI literature¹⁹.

* It should be remarked that design is not basically a mental activity. Ask any painter, musician, architect, designer of mechanisms, or software engineer. Design is basically a physically creative activity. It is only in recent times (since the industrial revolution) that certain kinds (such as mechanical design) have become so abstracted from the associated physically creative (manufacturing) aspects that it can now appear to be a purely mental activity. This, however, is only an appearance, not a real property.

'High road' systems are, in the preceding sense, deep systems and IICAD is aimed at them. The knowledge of such systems is expected to represent the principles and theories underlying the subject 'design'. This may require the demystification of several aspects of design by way of formal, mathematical methods. (N.B. Such a formalization may not say much about what 'goes on' during the design process, and serves as a rather simple *post hoc* rationalization of it.) In the case of IICAD, the fundamentals of a general design theory which is based on axiomatic set theory can be found in Tomiyama and Yoshikawa²³. It should be noted that we are not claiming, by positioning the existence of such theoretical frameworks, that we know all the problem-solving components of general design and can offer a comprehensive and detailed model of it. Nor do we deny that there are many domain-specific sides to design. For instance, VLSI design is mostly 2D while mechanical design is inherently 3D. IICAD hopes to incorporate the similarities in design, leaving the application-dependent issues to further consideration as side-requirements. We believe that only through a clean design theory and formalization can one arrive at testable conjectures of design and build computer models of it. We shall later see that we value logic as the principal tool in this formalization^{7,8,24,25}. The reader is also referred to Coyne²⁶ for an interesting treatment of logic models of design.

Key problems

It is useful to identify the problems of the existing CAD systems before attempting to propose any solutions.

CAD systems support few design processes and models – the latter being only partial and lacking integration

The production of final drawings is where current CAD systems tend to excel. This clearly depends upon 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 much of the details involved, such as tolerances, finishes, materials, etc. On the other hand, they are virtually powerless with respect to initial sketching. There are systems that can accept rough drawings but there is no system which can handle crude information[†] during the design process.

Another weakness in terms of design processes can be seen by comparing software development and industrial machine development 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 the least, there are few established methods other than outright testing to guarantee the correctness of manufactured design objects. This issue is admittedly more involved than it

seems since the same problem still exists in the software world today, although to a lesser extent. Furthermore, there are prospects for help, as we shall see, in emerging areas such as naive physics and commonsense reasoning.

Integration of models is essential because mechanical design deals with complicated gadgets. A design object must be viewed from various angles using different models. A very simple example is a wristwatch which can be viewed as an intricate assembly of gears, as a simple device with two hands rotating about a pivot, or as an abstract machine pointing to numbers denoting the time. More complicated examples follow when we consider the kinematic, dynamic, finite element, and control-theoretic models of a robot manipulator. In general, the present trend is to integrate the CAD systems around models concerning products. We suggest that they should be process-oriented and the so-called conceptual design stage should be supported by tools that contribute to the integration of CAD systems. In a nutshell, we mean by integration:

- An integration of subsystems (i.e. auxiliary programs).
- An integration of design models and views based on an integrated model description scheme.
- An integration of design processes and the computerisation of even very 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 exacerbate the situation, the final output of conventional systems can be so impressive that many errors go unnoticed for they exceed the mental capacity of 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 which make suggestions against say, spelling mistakes but this is clearly 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, 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.' Things deteriorate quickly as the requests become more complicated. It simply becomes impossible to get anything achieved without undue emotional trauma. When one inputs raw data manually, errors and misunderstandings during human-machine communication are inevitable regardless of input devices. The ultimate solution is that CAD systems must accept substantially reduced yet comprehensive data instead of raw data. For instance, a CAD system should accept commands like 'I would like to generate an object with such and such properties' or 'I have supplied the minimum requirements, so proceed as you think fit.' (Here we are using natural language just to write our commands concisely and naturally; normally, we assume that the user enters these commands in some formalism other than natural language.)

[†] The same holds true for natural language input but this is a controversial issue. At present, it is not clear why natural language should be useful in design even if it were reasonably available.

Temporality, ambiguity, and inconsistency are not allowed

In design, instead of sticking to one particular idea we may want to experiment with several ideas. We may like to use the paragon of top-down stepwise refinement²⁷. This brings a time dimension to design. We may, during design, purge things we have previously built or introduce things that we have not considered before. We may require that the system temporarily “forgets” a particular facet of a design object since we are not concerned with it at the moment.

We also frequently want to separate the structure of a design object from the values of its attributes so that we can first decide about its shape during the conceptual design stage. For example, it is more important to recognise first the topology of a part if it is going to be inserted in another. Similarly, we 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. Simply stated, the fact that an entity has attributes is different that the notion that an attribute has a value²⁸ (pp 180–185).

Unstructured code is commonplace

CAD systems are huge. Experience in software engineering informs us that maintainable software must 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 say, the advances in technology)^{27,29–31}. It is difficult to observe these characteristics in today’s CAD systems which are not open-ended and sturdy. Since they are not developed on a strong theoretical basis, they 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, ‘spaghetti-like’ programs.

To return to the first point made in this section, there is a lack of integration. Mechanical engineering data exchange may cause deterioration of meaning. A 3D solid modelling system based on say, boundary representation (B-rep), cannot easily exchange data with another solid modelling system based on say, constructive solid geometry (CSG)³². (The problem has connections with a fundamental philosophical problem known as *intensional vs extensional* descriptions and will be touched upon later in this paper^{23,33}.) Finally, current design systems do not offer a framework for design. Instead, they give the designer a set of rather arbitrary functionalities through which s/he is supposed to find his/her way.

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-solving process is not present. Insight is also needed to interpret the outcome of some computation. As Richard Hamming is quoted to have said³⁴, ‘The purpose of computing is insight, not numbers.’ Traditionally, mechanical design systems contain a good deal of

numerical knowledge (e.g. bulky libraries of numerical code) but nothing else. Users are left alone in analysing the results of long, confusing computations. Recent research in coupled systems is directed towards integrating the explanation and the problem-solving abilities of expert systems with the precision of numerical computing³⁴.

The need for coupling is especially felt in domains which use high technology. For example, design of a space shuttle cannot be accomplished by merely symbolic or numerical techniques. Fluid dynamics, thermodynamics, finite element analysis, etc. need to be integrated with techniques of database management, truth maintenance (which will be explained in the following), and symbolic computation.

IIICAD PRINCIPLES

The experimental systems that will be built to test the theory outlined in this paper and to study the derived problems is called the IIICAD system. The language especially developed to specify the system is called IDDL¹². IIICAD contains a run-time system for IDDL.

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 specific models can be promoted to permanent changes in the metamodel and propagate from the metamodel to the other models. IIICAD must cover three fundamental aspects of design: intelligence, integration, and interaction. It should support the designer in the entire design process, using unified models with rich functionalities for various design activities. It must have models of the design object which carry maximum similarity to the designer’s own images about them. At the same time, it must have knowledge about the design processes. This is achieved by explicit descriptions of the design processes and a control mechanism to guide the designer, namely, an intelligent supervisor.

Architectural overview

IIICAD components are shown in the block diagram of Figure 1. 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 when necessary. While SPV corrects the obvious user errors, it does not have the initiative for the design process itself because IIICAD is envisaged to be a designer’s apprentice, not an automatic design environment. It is noted that scenarios are design process descriptions. External information is further exchanged with the user interface module and with the application modules, for which the specific models mentioned in the preceding paragraph may have been generated first.

Thus, we propose that IIICAD has a control mechanism to guide the designer, compares user actions with scenarios; and is envisaged to be a designer’s apprentice. Obviously, the first point suggests that the system has

Figure 1. IIICAD architecture (block diagram level). Key: IUI: intelligent user interface; SPV: supervisor, KB: Knowledge base; DB: database; API: applications programs interface; IDL: integrated data description language; IDDS: integrated data description schema; GKS: Graphical Kernel System, DICE: dialogue cells (NB GKS may be replaced by PHIGS)

control knowledge which is applied to the human designer; the second point suggests that the system checks the actions of the human designer against some control knowledge; and the third point suggests that the system is trained by the human designer (like a human apprentice is trained by a master craftsman). We believe that all these modes of behaviour are correct.

The Integrated Data Description Schema (IDDS) regiments the databases and knowledge bases and relieves the user from the burden of specifying where and how to store and retrieve data. The Knowledge Base (KB) is divided into two parts: an object store where all objects and their internal structure can be found¹¹, and a rule store where the relations between several objects are defined in terms of definite program clauses as in Prolog³⁵. The parts of the design artifact and their internal structure are kept in the object store whereas facts and inter-relationships among the parts are furnished by the rule store via definite program clauses. Functions connect the object and rule stores.

Clearly, the development of a knowledge base system is paramount to the project. Here, all information about the design objects, including their attributes and functions (or using the object-oriented terminology, instance variables and methods^{36,37}) as well as the relationships among the objects, will be stored. The KB 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 (procedural) objects and by the logic programming paradigm to describe (declaratively) the inter-relationships among the objects^{31,36}.

It is noted that the rule store is used most dynamically during the design process. To provide flexibility to this dynamic behaviour, IDDS should provide a mechanism to partition the rule base, i.e. to group the clauses for specific purposes. The advantages of this are two-fold: logical computation is made efficient since only a relatively small amount of knowledge is passed to the inference mechanism; and consistency is easier to maintain since less information need to be considered.

IDDS has a kernel language called the Integrated Data Description Language (IDDL) used by all system elements. IDDL is the means used to code the design

knowledge and the design objects in order to guarantee integrated descriptions systemwide. IDDL will be based on logic and accordingly knowledge engineering is the key factor in building our system^{12,35}.

In addition to the above principal elements, IIICAD has a high level interface called the Intelligent User Interface (IUI) which is also driven by scenarios written in IDDL. IUI accepts messages from the other subsystems and sends them to other lower level interface systems such as DICE (Dialogue Cells)³⁸ and the well-known GKS[†] (Graphical Kernel System)⁴⁰. Syntactical errors of the user are processed by IUI whereas semantic errors can be processed, as far as this is possible, by SPV. Our goal is to manipulate more user-oriented concepts. This is the basis for more direct communication (i.e. without intermediate abstract command repertoire). To support it, the user interface module must be provided with sufficient information about the semantics to recognise the symbols being used, both for input and output (interpretation for visualization). Such a system can be built as a two-layer interface system. The lower level is a user interface. The upper layer interprets the syntactically accepted constructs of the lower layer on the basis of declarative knowledge which has been made available by SPV (scenario monitor). Semantic recognition can drive numerous 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 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 modellers.
- Engineering analysis systems such as finite element packages.
- Product modellers.

User interface

There are several useful ways of looking at intelligent CAD user interface architecture. The following dichotomies are quite common:

- CAD versus automated design (AD)
- Designer's apprentice (assistant) versus autonomous design system
- Glass box versus black box

The boundary between CAD and AD is indeed hard to delineate. We cannot object to the view that the

[†] Regarding the use of GKS as a means of implementing interactive graphics in our system architecture, there may be differences of opinion. For instance, another system called PHIGS (The Programmer's Hierarchical Interactive Graphics System)³⁸⁻³⁹ provides an explicit hierarchical data structure and the capability for true dynamic data structures and dynamic user interaction.

ultimate aim of the computerization of design is to arrive at completely 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 human-machine cooperation must dominate. The same holds true for apprentice vs autonomous systems. An apprentice system has less hardwired knowledge than an autonomous system, but knows better how to interact and has a generic model of design. An autonomous system is very powerful for narrow domains. Besides, in such domains there may not be a need for a lot of interaction anyway. It is relatively easy to extend an apprentice by teaching it new skills. It is unwieldy to extend an autonomous system since its very constitution warrants myopia.

A more natural look at these dichotomies is via the metaphor glass box versus black box⁴¹. 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 (Figure 2(a)). On the other hand, a black box system resembles a batch processing environment; one submits the tasks to be executed and the system reports back with the results (Figure 2(b)).

The seasoned researchers of CAD may remember those times when ideas such as 'general CAD' became fashionable and then were silently abandoned⁴². Today, demands for integration suggest that we may want to reconsider 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 you have a cooperating set of experts who can communicate with each other then you can solve many problems (see Figure 3(a) where, lines denote the communication links between experts shown as boxes.) 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 (Figure 3(b)). The shell may especially take care of the system-wide communications and database management.

STRATEGIES OF DEVELOPMENT

We must be aware of two caveats in IICAD development. The first caveat states that everything should be based on a formal theory; only a firm basis allows us to avoid the infamous hacker trap – supplying numerous features which overwhelm the user. (Note, however, that the decision as to what features to provide is not strictly governed by having a formal basis; it is more determined by the style of a system's design and implementation and ultimately, the discipline of the makers.) The second caveat states that we should be careful about software development; while knowledge engineering is qualitatively different from software engineering, the knowledge bases will still have to be maintained¹⁹. In other words, completed AI programs are just starting points for more advanced programs, obviating the traditional idea of 'stable' databases³⁰.

Theory of CAD

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 design theory will have to satisfy the following:

- It is realistic in the sense that it has a close relation to design practice; it describes design processes as they are in practice or as users would like (or are accustomed) to do these.
- It is founded on a logical basis so that there are guarantees that a system, developed according to the design theory, will take sound steps.

These requirements for a design theory can be detailed as follows:

- Phenomenological part: here descriptions are given as they are in practice. Also, identification of a number of design types (such as routine design vs more creative types of design), design strategies, and design styles can be made.
- Foundation apart: here logical foundations are given for the conceptual descriptions in the phenomenological part. Thus, we allow for logical representations of incomplete descriptions of objects, patterns of reasoning involved in design^{3,43}, and a multiworld mechanism (which is described in the sequel).

There are three ontological aspects of design: processes, models, and activities. This implies that we need theories corresponding to each aspect. A theory of CAD is then an aggregate of the following three theories:

- Theory of design: a theory which describes the design processes and activities²³.

Figure 2. User interface architectures: (a) glass box versus (b) black box

Figure 3. (a) Intelligent tools versus (b) frameworks

- Theory of design objects: a theory which deals with the models of design objects⁴⁴. For our purposes, this can be a theory of machines; in VLSI design it would be a theory of VLSI, and so on. (On a related note, Kieras and Bovair⁴⁵ present an interesting discussion about the role of mental models – a personal theory of design objects, if one may say so – in a person's learning how to operate a device.)
- Theory of knowledge: a metalevel theory to describe our knowledge about design⁴⁶. (A less philosophical reference which treats the role of knowledge in knowledge-based systems is Newell⁴⁷.)

Design theory and logic

In General Design Theory²³, a design process is regarded as a mapping from the function space onto the attribute space (Figure 4). Both spaces are defined on an entity set. A design process is an evolution process about a metamodel. A metamodel is a set of attributive descriptions of a design entity. 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 design object are determined so as to be able to manufacture it.

To further illustrate the design process, three major components need to be recognised: entities, attributes of entities, and relationships among entities. A design process is thus a sequence of small steps (forward and backward) to obtain complete information about these components. Note that we also observed entities which do not change during design. For instance, when designing a bicycle lock, several pins and screws, which cannot be changed are used as building blocks. We call such objects invariants. Description of the lock at a particular design stage is, conversely, dynamically changing. We call these descriptions variants of a design process. Clearly, the concept of say, 'hole' should not change while designing a lock; however, values of the attributes such as the centre coordinates of the hole may change. We call such changeable concepts covariants.

It was mentioned before that to control the stepwise refinement of the design process we need to express unknown, uncertain, and temporal information about the design object. We can accomplish this using an amalgam of intuitionistic logic, modal logic, temporal logic, inheritance, and situational calculus⁸. We take it

Figure 4. Design according to General Design Theory

Figure 5. Design process model depicted as flowchart. key: M: metamodel; m: model; D: detailisation; d: modelling; e: evaluation; ?: feasibility check; OK: passing a check

for granted that descriptions of a design object are given by a set of propositions[§] (such as well-formed formulae in first order predicate logic). A simplified view of design is then as follows (Figure 5). A sequence of metamodels are generated from an initial specification and are then detailed. If they cannot pass a feasibility check, then a compromise is made or there is a need to backtrack. Note that the models derived from metamodels are in the meantime evaluated for consistency and for other concerns. We thus have:

$$\begin{aligned} \text{Specification: } s &= T^0 \rightarrow \dots \rightarrow T^n \rightarrow \dots \rightarrow T^N = g \\ \text{Metamodel: } M^0 &\rightarrow \dots \rightarrow M^n \rightarrow \dots \rightarrow M^N \\ \text{Propositions: } q_0 &\rightarrow \dots \rightarrow q_n \rightarrow \dots \rightarrow q_N \end{aligned}$$

In this scheme, s is the original design specifications and g is the design goal. Each design step has an associated set of propositions which are denoted by q_n above. Two central concerns here are first, how to choose q_n (i.e. how to proceed with design) and second, what if we discover $\sim q_n$ (i.e. how to deal with contradictions). Some other definitions are given

$$q_n = p_0 \wedge p_1 \wedge \dots \wedge p_k$$

where each p_i is an atomic fact concerning the metamodel

$$q_n \vdash^{C_n} m^n$$

where $\vdash$ is the syntactic turnstile as usual in proof theory⁵⁰, m^n is a model and C_n is control knowledge or modelling knowledge, and

$$q^n, m^n \vdash^{D_n} r$$

[§] To take for granted that descriptions of a design object are given by a set of propositions can be regarded as a mischievous attempt to hide an important assumption in the approach taken towards CAD – that a first order logic can be used so universally in modelling the design process and representing the knowledge used and generated. We now openly admit that we make this assumption and accept that this is a controversial position to hold, not only in design, but in AI as a whole^{24,25,43,48,49}.

where D_n is detailing knowledge and r is a proposition which should be added to q_n to arrive at the next description, q_{n+1} . Knowledge appearing above the syntactic turnstile is used in the derivations as metaknowledge. With this notation, the following partial classification of design becomes rather plain:

- Invention: given s , find q_n , C_n , D_n , and g .
- New product development: given s and C_n , find q_n , D_n , and g .
- Routine design: given s , C_n , and D_n , find q_n and g .
- Parametric design: given s , q_n , C_n , and D_n , find g .

As the preceding discussion views design as making changes to a database of facts, we are in fact quite close to the world of logic programming⁴⁷. (We admit that design can be viewed as 'modifying a database' only if the reader is familiar with the way in which logic programming languages are implemented^{35,51,52}; it does not, we should note, follow naturally from an understanding of proofs in a first order logic⁵⁰.) Logic with modes of truth, viz. modal logic with necessity and possibility, can therefore be used as a notational tool^{46,53}. Here we have not only affirmations such as that some proposition is true, but also stronger ones such as that p is necessary (denoted $\mathcal{N}_p$) and weaker ones such as that p is possible (denoted $\mathcal{P}_p$). If p is asserted necessary, then we can also assert p true. If p is asserted necessary, then we can also assert p true. If p is asserted true then we can also assert p possible.

Predicate calculus of higher order is useful to talk about inheritance – a subject of prime importance in object-oriented systems⁵⁴. The following is provable in the second predicate logic

$$\forall F[F(x) \supset G(x)]$$

(If any individual x has every property then x has any property.) In third order predicate logic, we can prove that

$$\forall F[V(F) \supset V(G)]$$

(Whatever is true of all functions of individuals is true of any function of individuals.) While they are, theoretically speaking, well-understood, the real challenge of higher order logics lies in their efficient implementation.

For temporal logic, we can use the following notation. Let αp denote that p holds after time t and βp denote that p holds before time t ; $[t_1, t_2]$ denotes a time interval. Several useful equalities suggest themselves

$$\alpha \sim p = \sim(\alpha p)$$

$$\alpha(p \vee q) = \alpha p \vee \alpha q$$

$$\alpha(p \wedge q) = \alpha p \wedge \alpha q$$

Also note that the following convention is useful

$$[t_1, t_2] \overset{\text{def}}{\alpha} p = t_1 \alpha p \wedge t_2 \beta p \wedge t_1 < t_2$$

Using temporal logic, we can describe inference control[†] for production rule systems in a more explicit way. For instance, in Prolog the order of rules matter^{31,35,51,52}. In general, this knowledge is embedded in the interpreter of the language. By disclosing this control we may introduce more flexible control. For example, the detailing knowledge D_n introduced above may be a set of rules of the following sort

$$t_1 \alpha q_1 \wedge t_2 \alpha q_2 \wedge t_1 < t_2 \supset t_2 \alpha q_3$$

$$t_1 \alpha q_1 \wedge t_2 \alpha q_2 \wedge t_1 > t_2 \supset t_2 \alpha q_4$$

These two formulas should be read as follows. 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 (i.e. t_1 is later than t_2) another property, q_4 , holds. Applications of temporal logic to design process representation should now be manifest.

Intuitionistic logic (a three-valued logic in which the law-of-excluded-middle is challenged) can naturally be blended with temporal logic. Let $t_p \alpha p \equiv \text{true}$. Now, by introducing a logical symbol 'unknown', we can formalise intuitionism in terms of temporality:

$$t_p \beta(p \vee \sim p) \equiv \text{unknown}$$

$$t_p \alpha(p \vee \sim p) \equiv \text{true}$$

Here, the idea is to regard the proposition p as a property that we are trying to prove about the current design object or design process.

We note that merging the complete functionalities of these assorted logics may very well result in high (even intractable) computational complexity. To partially overcome this, we must include only those functionalities which are relevant to our design requirements. (Clearly, only including those functionalities which are relevant does not necessarily avoid problems of intractable computational complexity.) For example, in the case of temporality we are satisfied with only α and β even though temporal logic is more complex than these simple operators might suggest.

Naive physics and machine design

In machine design, it is not yet precisely known in 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^{23,44,45,55,56}. Regarding this point of view, the representation of functions can be reduced to the representation of physical phenomena and qualitative reasoning about them. Minsky⁵⁷ (pp 5–6) explains this forcefully in the following two excerpts:

"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

[†]This idea is usually called metalevel control. It is well known to suffer from efficiency problems and we acknowledge that what is presented in this paragraph is difficult to implement efficiently in a general purpose way.

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 criticise the material part of the system when the correspondence breaks down.” (Minsky’s italics)

So as to give the reader a better appreciation of the kind of ‘physical’ reasoning we are seeking, we briefly examine at geometric reasoning. This is also an example of general knowledge that needs to be applied in specific situations. Normally, each specific case warrants only a small fraction of the general knowledge. It is necessary to isolate a sufficient yet still small fraction of the general knowledge in order to be efficient. For geometry, the emphasis is on generating the appropriate representation so as to answer queries as directly as possible. One rather 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 compared with the matching ‘forms’ in the geometry base. Then the closest one is used as a starter for inference. This is in the precise spirit of de Kleer’s *Random Access Local Consequent Methods*¹.

In qualitative physics, the reduction of information is arrived by creating an abstract layer (the naive or qualitative layer) which may, strictly speaking, be incorrect, but still sufficiently correct for the current problem. 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. (There are a number of developmental psychologists who would not agree with the second part of this sentence.) Accordingly, naive physics ideas are useful in machine design and we want to codify them in the IICAD system. To this end, we follow the Hayes manifesto⁵⁸ and hope to capture naive physics in logic. Additionally, we use as a mathematical tool qualitative reasoning based on simple symbolic manipulations. It is only fair to remark that the research programme advocated⁵⁸, has been pursued by some of the best minds in AI for the last 20 years, but with very little to show for the effort so far. This lack of results has recently become the subject of some debate in the AI literature; codification of commonsense reasoning (based upon a naive understanding of physics) is generally regarded as one of the hardest unsolved problems in AI research today.

Consider the following concepts underlying ‘change’ in physical systems: state, cause, equilibrium, oscillation, force, feedback, etc. Naive physics regards these concepts in a simple qualitative way. It maps continuous

variables to discrete variables taking only a small number of values (e.g. +, −, and 0). Accordingly, differential equations are mapped to qualitative differential equations (also known as confluences). A rather comprehensive mathematical theory of confluences is developed in de Kleer and Brown⁵⁵ and independently, in Kuipers⁴¹; the reader should, however, have no difficulty in understanding the following example. Consider Figure 6 which depicts a pressure regulator and is adapted from de Kleer and Brown⁵⁵. The confluence

$$\partial P + \partial A - \partial Q = 0$$

describes this device in qualitative terms. Here P is the pressure across the valve, A is the area available for flow, and Q is the flow through the valve; ∂ means change. Let us look at the way the regulator works. (In the following, $\Rightarrow$ means ‘implies’ and $\uparrow$ and $\downarrow$ stand for ‘increase’ and ‘decrease,’ respectively.)

An $\uparrow$ in pressure at a $\Rightarrow$ an $\uparrow$ in pressure at b $\Rightarrow$ an $\uparrow$ in flow through b $\Rightarrow$ an $\uparrow$ in pressure at d $\Rightarrow$ diaphragm at e pushes downward $\Rightarrow$ valve at b tightens $\Rightarrow$ an $\downarrow$ in area for flow $\Rightarrow$ a $\downarrow$ in flow through b.

Thus the regulator maintains constant pressure at c in spite of fluctuations at the supply side a. Note that if the valve is closed (i.e. $\partial A = \partial Q = 0$) then it is predicted that P is constant. The correct confluence for this state should instead read $\partial Q = 0$; we simply do not say anything about P . Such time periods during which a device shows significantly different behaviour are known as episodes⁵⁹.

We have to admit that when one speaks of mechanical or more specifically machine design, the idea of mechanism (interconnected links and joints) design should also be considered. Mechanism design represents the foundation of a large percentage of the entire field of machine design. Also, it is probably more complex than other types of mechanical design (i.e. static components). Examples of mechanisms of reasonable complexity (compared to the pressure regulator which is far too trivial) can be found in Faltings⁶⁰ and Nielsen⁶¹.

The use of qualitative reasoning facilitates the analysis of the working of physical devices. This may not be without a price. Qualitative techniques may cause ambiguities. Assume that a certain quantity M varies

Figure 6. Pressure regulator (adapted from de Kleer and Brown⁵⁵)

with N_1/N_2 , i.e. $M \propto N_1/N_2$. If $N_1 \uparrow$ while N_2 remains constant then M also $\uparrow$. However, we cannot reason without a finer knowledge of the individual changes when we are told that both N_1 and $N_2 \uparrow$ (or $\downarrow$). The techniques of order of magnitude reasoning are designed to handle precisely this kind of problem without requiring knowledge of the numerical values involved. The reader is referred to Raiman⁶² and Simmons⁶³ for two especially good papers on the subject.

In the pressure regulator example, we are employing a powerful principle in our reasoning, namely, the mechanistic world view. This asserts that every physical situation is regarded as a device made up of individual components, each contributing to the overall behaviour. Nevertheless, the laws of the substructures may not presume the functioning of the whole – the principle* of *no-function-in-structure*⁵⁵. 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 devices⁵⁵. A simplistic view of modelling devices comprises three kinds of constituents: materials (e.g. oil, air), components (e.g. shafts, wheels), and channels (e.g. water pipes, electric cables). Channels transfer material from one point to another. It is assumed that a channel is always full of material and obeys the law of conservation.

After modelling a device we can reason about it. In ‘envisionment’ we start with a structural description to determine all possible behavioural sequences^{1,56}. Thus, in envisioning, we momentarily forget about the real values of the problem variables and try to see all possible outcomes of some action. Envisionment has close ties with ‘causation’. This relationship between two states of affairs such that the first brings about the second can be modelled using temporal logic operators such as α and β in a situational calculus[†] setting^{68,69}. As an example of partial envisioning, imagine a box OBJ, with a smaller block ‘obj’ inside, sliding on a surface inclined towards water (see Figure 7). OBJ will slide down the inclined surface and reach its corner. Then it will fall and hit the water. If OBJ breaks then obj will be released to the water and will possibly sink. Modal logic comes handy to codify these facts. Thus, we may write statements like

$$\begin{aligned} &\text{on}(\text{OBJ}, \text{inclinedsurface}) \wedge \neg \text{fixed OBJ}) \\ &\quad \supset \mathcal{P}(\text{slide}(\text{OBJ})) \\ &\text{plunged}(\text{obj}, \text{water}) \wedge \text{akindof}(\text{obj}, \text{sugar}) \\ &\quad \supset \mathcal{N}(\text{dissolve}(\text{obj})) \end{aligned}$$

Naive physics concepts are required for design because a design object will have a physical existence and

Figure 7. Envisioning with a block sliding on an inclined surface

accordingly, obey natural laws. If we want to create designs corresponding to physically realisable design objects than we will have to refer to naive physics notions such as solids, space, motion, etc. Furthermore, if we want to reason about a design object in its destined environment (think of our pressure regulator to be installed in a nuclear reactor⁵⁹) we will need naive physics procedures such as envisioning, simulation, and diagnostics.

Nevertheless, the following caveat should be offered here. Naive physics might be sufficiently correct for use in describing (to some level of accuracy) some physical phenomena, but this is a different matter from using naive physics to design real world objects, where it is much harder to know *a priori* what level of correctness is required. Design usually assumes a high accuracy is obligatory and that the best representations of natural laws are requisite. That is why we use Newtonian mechanics for example, rather than something like Aristotelian mechanics, which might seem more intuitive to a naive physicist. It is one thing to use naive physics notions to predict what might happen in some physical system, it is quite another to use them to determine how to get something to happen in a physical system.

Studying thinking

As IICAD will be a tool for dealing with an intellectual process, we need theories that are directly related to thinking. We should certainly study epistemological theories from philosophy⁴⁶, cognitive science, and psychology since they contribute to our understanding of design. In database research, the counterpart of such knowledge is known as conceptual modelling (cf. the book in which Levesque²⁸ appears) and has been instrumental in establishing a sound basis for database management. (N.B. A general study of philosophy, cognitive science, and psychology may not necessarily contribute to our understanding of design. It may be that concepts, methods, and techniques from these disciplines might be useful in studying and understanding design, but these will be particular concepts, methods, and techniques. Certainly, the conceptual models used in databases show few signs of being so derived.)

In the history of AI, there are many interesting debates on foundational matters such as procedural versus declarative controversy and logic versus

* The principle of *no-function-in-structure* is known to be hard to apply in practice; even de Kleer and Brown have recently admitted as much⁶⁴ (pp 56–58) in response to Iwasaki and Simon’s criticism⁶⁵. For another good discussion of the issues involved, see also Gomez and Segami⁶⁶.

† McCarthy and Hayes have long since abandoned situational calculus as being inadequate for representing even trivial states of affairs. More powerful temporal logics have been developed; cf. Shoham⁶⁷ for a particularly interesting one. This last reference is also an excellent source on the computational aspects of causality.

special knowledge representation techniques^{48,69}. Neither of these debates is settled yet; it is quite possible that there are no unequivocal answers. The need for coupled systems for symbolic and numerical computation suggests an appropriate mix of procedural and declarative languages. In case of knowledge representation, we accept that logic is syntactical and hence sterile in regard to meaning. However, it should not be forgotten that it is the best formalization humanity has yet built. Furthermore, it is possible to implement say, frames entirely in logic^{25,58}. Yet, there are certain kinds of knowledge which it is extremely difficult, some would say impossible, to represent using first order logic.

Theory of knowledge deals with issues of the preceding sort and is especially necessary to place design knowledge in a particular framework to utilise it in the IICAD architecture. The following discussion is based on Tomiyama and ten Hagen³³, and draws a distinction between two opposing knowledge representation methods, i.e. extensional versus intensional descriptions. We do not take up here the philosophical [35] distinctions between these methods although those also are relevant to knowledge representation.

In order to discuss design, we need to describe entities, their properties, and the relationships among entities. 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_1 and e_2 stand in relationship r is described by $r(e_1, e_2)$. In an intensional description method these two facts can be represented by $e(p)$ and relation (e_1, r, e_2) , respectively. Extensional descriptions do not imply any preconceptions while intensional descriptions make assumptions such as that e 's property is limited to one particular p . When an intensional description has to be changed, this results in a modification of those predefined conditions. For instance, a shaft might be represented intensionally by

shaft(diameter, length, bearing₁, bearing₂)

If we now want to add new attributes, such as transferring power, this 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);
supportedby(s, b1); supportedby(s, b2);
bearing(b1); bearing(b2).
```

Thus, in an extensional description we need to write numerous obvious descriptions. However, modifying such a representation is just adding or deleting facts. On the other hand, an intensional description might be difficult to change but shows better performance. To see this consider an extensional description where we need explicit information about two adjoining edges. If we do not have this information explicitly, we need to infer it using rules. In this case, clearly an intentional description which lets us trace a 'next edge' pointer is much faster.

CAD applications request a flexible data description scheme which is easy to modify^{6,7,8}. Independent multiple

views of a design object demand independent small partitionings of the database. This is easily achieved by an extensional description method since we only have to pick up the relevant facts. In an intensional description this requires the creation of a totally new scheme.

There are variants which change during the design process. The extensional description can be used to recount these variants. We know that logic programming^{49,51} is useful to implement such description methods. On the other hand, we use invariants as building blocks in design. Such blocks do not change their structural properties although values of their attributes might change. Invariants, therefore, can be represented intensionally and their properties can be depicted as covariants.

RAPID PROTOTYPING AND CAD SOFTWARE

In building knowledge-based systems such as IICAD, we are trying to develop a system which will be open-ended; thus IICAD will evolve over time, with the definition of its purpose being refined as it becomes a fuller system. We therefore believe that, in the development of intelligent CAD systems the underlying strategy should be 'Plan to throw one away. You will anyhow'²⁹. This suggests that recent techniques of software engineering such as exploratory programming³⁰ and rapid prototyping⁷⁰ 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. We start with a skeletal implementation (rapid prototype) and add new modules until the system is reasonably complete for demonstration. In rapid prototyping we are interested in perceiving a glimpse of a future system in order to assess its strengths and weaknesses.

Smalltalk

With regard to exploratory programming and rapid prototyping we think that Smalltalk is an excellent implementation tool. First, we appreciate the emphasis of Smalltalk[‡] on interactive graphics and favour its sophisticated user interface³⁷.

Second, Smalltalk is not just an isolated programming language; it is a programming environment. For example, the ability to inspect in Smalltalk any data structure recursively to any depth gives the programmer a great ability for exploratory programming. Smalltalk's insistence on expressing everything as objects and messages using a simple syntax is conceptually clean, powerful, and consistent.

Our experience with Smalltalk also suggests that it is not yet perfect. First and foremost, learning Smalltalk and using it effectively are not easy tasks. Although there is a natural associated learning curve^{71,72}, a common joke tells that it is easier to learn Smalltalk in an established Smalltalk-using organisation where

[‡] Smalltalk-80 is a registered trademark of Xerox Corporation. In what follows, we shall simply write Smalltalk instead of Smalltalk-80.

colleagues who are Smalltalk experts are available. The so-called 'methods hierarchy' is simply too big; Smalltalk code that performs some function rarely resides in a well-defined location but is distributed all over the system. The concept of 'information hiding' then becomes somewhat wearisome as we almost always need to have an idea about how something works. Long chains of message-passing and 'super- and subclass responsibilities' make it difficult for the novice to understand how some action takes place. This is perilous because it tends to de-emphasise one of the major goals of Smalltalk – encouraging reuse of software. 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. Especially, the inheritance mechanism, even when it is enriched with multiple inheritance, is seen as a drawback for CAD systems. Recently, a new scheme called delegation is being popularized for flexible sharing of knowledge in object-oriented systems⁵⁴. 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).

Advantages of Loops

We close this section with a short overview of Loops, a language which is becoming increasingly popular due to its multiparadigm nature⁷³.

Loops supports a mechanism for annotated values. This helps programmer monitor arbitrary values without previously planning such access. There are two kinds of annotated values: property annotations and active values. The latter associate with any value a demon to be invoked when a data store or fetch is requested. The former associate with any value an optional property list in order to annotate the value with useful information.

From a CAD viewpoint, another useful construct of Loops is 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, etc. 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'. Perspectives are a form of composite object and provide a way to implement the multiple views of the same design object. Thus we may represent the concept 'Pressure Regulator' using perspectives like `mechanicalassembly`, `displayobject`, and `functionbox`. 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. In the third perspective, we are interested in the functioning of the regulator as a

control-theoretic device with feedback. Perspectives should be compared with the Smalltalk concept of model-view-controller triad – a standard implementation strategy for user interfaces⁷⁰.

Loops also 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.

IDDL: DESIGN BASE LANGUAGE

IDDL will be the base language upon which the subsystems of IICAD will be built. This section summarises the requirements for IDDL. For the sake of brevity we do not attempt to justify our requirements here; this has been done previously⁸. Besides, many of the requirements have rather obvious reasons behind them. It is also noted that the following description is incomplete; the current version of IDDL is presently under reconstruction¹².

IDDL should deal with two kinds of names: system names are unique and internal whereas user-defined names are modifiable and external; this follows from our decision to use extensional descriptions as much as possible. Similarly, IDDL should make a distinction between the facts that an entity has an attribute and an attribute has a value. It should be able to describe status and control information of the system with origins, destinations, and time stamps.

IDDL, as a language to construct knowledge bases, 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. It should be able to describe the knowledge to detail a metamodel, to check metamodel feasibility, and to control the detailing process. It should allow a myriad of views of a design object; these views are possibly independent, but still correlated.

IDDL should incorporate means to represent (and reason with) positive and negative information, known and unknown, and modalities such as necessary and possible. This requires that IDDL should consolidate logics such as modal, intuitive, and temporal. Inconsistencies need to be resolved with some sort of truth maintenance when transferring to a next metamodel. In other words we see design problem-solving also as selecting among plausible alternatives. One naive method for exploring the search space is to enumerate its points and test. This is the so-called 'British Museum' approach. It is more advantageous if we can find solutions that meet some criteria of acceptability. This implies that we should be able to compare partial solutions. A world is defined by a set of assumptions and defines a possible solution. Different assumptions on the same range are clearly contradictory. That is, the database that will record the different worlds will be inconsistent. We need a truth maintenance system (TMS) to work safely with an inconsistent database. A TMS records the inferences made by the problem-solver and then avoids repeating the same inferences. Once a contradiction is found, the world it belongs in is no longer considered. By using a technique

Figure 8. Enclosure mechanism for data encapsulation

called 'dependency-directed backtracking' a TMS is able to remove contradictions from the database and rebuild a consistent world. TMSs are studied in detail in references⁷⁴⁻⁷⁷. (De Kleer's assumption-based truth maintenance techniques⁷⁵⁻⁷⁷ specifically offer a way of avoiding the problems associated with dependency-directed backtracking.)

To control the behaviour of the system, 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 (Figure 8). From the viewpoint of interactive design, it is desirable for the human designer to be able to mark intermediate design stages, and later go back for examining or resuming from there. Consequently, it must be possible to represent the stages of design evolution on the level of IDDL.

Design produces intermediate results which are incomplete and even inconsistent during a time-spanning series of transactions. The design object's attributive representation must allow assumptions to be used for the evolution of the design object. IDDL, using nonmonotonicity, should be able to retract assumed (but later on unconsidered or disproved) propositions⁸ about design.

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 which knows everything about itself. Reuse of software may be assisted by the class inheritance mechanism. It is an additional benefit of object-oriented programming systems that they offer incremental compilation, dynamic binding, rich system building tools, and good user interfaces. Therefore object-oriented programming is a good choice for creating IICAD. On the other hand, logic programming is powerful for problem-solving since it can reflect the reasoning process most naturally and directly. IDDL uses the logic programming paradigm to express the design process, whereas the object-oriented programming paradigm is used to express design objects. In IDDL, invariants, variants, and covariants will be represented respectively by objects, their internal and external relationships, and behaviour of objects.

Multiworld mechanism

During the course of design, it is of vital importance that an intelligent CAD system allows the designer to represent the design object in various ways. We saw that these various descriptions are called models of the design object. Four examples of such models⁹ are:

- Functional specification of the design object in terms of the constraints that should be met.
- A geometrical model which creates a visual representation of the design object (something that current CAD systems do rather well).
- A finite element method (FEM) model enabling strength predictions to be made on complex mechanical constructions.
- A cost-analysis model to calculate the manufacturing costs.

Models are able to communicate with each other by means of a metamodel. A metamodel is a central description of the design object to which all models refer and depend on. All changes that occur in a certain model are propagated through the metamodel to all other models (which are updated appropriately).

The metamodel is represented in IDDL. Although the different models may use different languages internally (e.g. DICE for the user interface, C for the cost analyser, a CSG language for the solid modeller, etc.), the interface between a metamodel and the other models is realised in IDDL. This is achieved by a special language construct, the *multiworld mechanism*¹⁷. 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

⁸ This notion of nonmonotonicity is clearly reminiscent of TMSs but another, model-theoretic explanation of nonmonotonic logic is as follows⁶⁷. The meaning of a formula in classical logic is the set of interpretations that satisfy it; these are the models. We obtain a nonmonotonic logic by accepting only a subset of these models. These 'preferred' models are preferable in a certain respect. Now, in classical log. $A \models C$ if C is true in all models of A (here $\models$ is the semantic turnstile in model theory⁵⁰). As all models of $A \wedge B$ are also models of A , it follows that $A \wedge B \models C$ (hence the logic is monotonic). In the nonmonotonic case, we see that $A \models C$ if C is true in all preferred models of A , but $A \wedge B$ may have preferred models that are not preferred models of A . Many different preference criteria are possible, all resulting in different nonmonotonic logics.

⁹ The careful reader has probably already noticed that our liberal use of the word 'model' is a potential source of confusion in the light of the technical meaning this word has in model theory⁵⁰. We only hope that the context made it clear enough when we employed this word in the latter sense.

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, we make a distinction between dependent as against 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 while 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 which 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.

CONCLUDING REMARKS

We have presented a unifying framework to describe and use design knowledge. Our starting point, the theory of CAD, allows us to formulate IDDL specifications. It enables us to understand, clarify, model, and formalize design processes and design knowledge in an intelligent CAD environment. We believe that theoretical ideas of knowledge engineering and naive physics, formal tools such as logic, and practical software techniques such as prototyping are of essential help in constructing intelligent design systems.

The theory should first clarify, 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. Following this overall characterization, an attempt is made to identify the parts of the process and the objects best suited for computer representation. The global and the formal theories must be correlated in order to be realistic. For instance, they must both be able to describe the design object as an evolutionary model of the real artifact.

The development of IDDL was given priority to the development of other subsystems of IIICAD¹². This is due to the fact that IDDL will be the kernel language of our system. The two current goals of the project are then as follows:

- implement powerful tools for IIICAD development
- construct a more complete version of IDDL

Smalltalk is used as an implementation language on top of which IDDL and a Prolog-like inference engine will be built. Except for the hierarchical constructs, IDDL will 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. A Prolog-like inference engine is finally run to infer knowledge from the rule base.

Near future work includes:

- the development of subsystems of IIICAD such as SPV, API, and IUI
- the incorporation of existing CAD tools and knowledge based systems, including a qualitative reasoning expert and a product modeller, in the IIICAD framework
- the justification of the methodology of developing IIICAD and eventually the proof of its effectiveness by case studies.

ACKNOWLEDGEMENTS

This work is funded by Nationale Faciliteit Informatica (NFI), a Dutch government agency. An earlier version appeared as: **Akman, V, ten Hagen, P J W and Tomiyama, T** 'Design as a formal knowledge engineered activity' *Technical Report CS-R8744* Centre for Mathematics and Computer Science (CWI), Amsterdam (September 1987). The present paper is substantially different from the earlier version, with additional material incorporated at several places and many revisions. Some of the new material came from a CWI internal document titled 'Continueringaanvraag voor project NF-51/62-514: Intelligent CAD Systems.' A talk based on this paper was given by the first author during the *First Bilkent-Carnegie Mellon Joint Conference* Bilkent University, Ankara, Turkey (29–30 May, 1989).

Comments of two anonymous referees of *Computer-Aided Design* were very helpful in improving the presentation and the contents considerably. Especially, one of the referees had so many forceful observations and appropriate remarks that we could not help using them verbatim.

REFERENCES

- 1 **de Kleer, J** 'Qualitative and quantitative knowledge in classical mechanics' *Technical Report AI-TR-352* Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, MA, USA (December 1975)
- 2 **Yeomans, R W, Choudry, A and ten Hagen, P J W** *Design Rules for a CIM System* North-Holland (1985)
- 3 **Simon, H A** 'The science of design: creating the artificial' Chapter 3 in *The Sciences of the Artificial* MIT Press (1979) Seventh Printing
- 4 **Forbus, K** 'Intelligent computer-aided engineering' *AI Magazine* Vol 9 No 3 (1988) pp 23–36
- 5 **Lansdown, J** 'Graphics, design, and artificial intelligence' in **R A Earnshaw (ed)** *Theoretical Foundations of Computer Graphics and CAD* NATO ASI Series Vol F40 Springer-Verlag (1988) pp 1153–1174
- 6 **Tomiyama, T and ten Hagen, P J W** 'Organization of design knowledge in an intelligent CAD environment' *Technical Report CS-R8720* Centre for Mathematics and Computer Science, Amsterdam, Netherlands (April 1987)

- 7 **Akman, V, ten Hagen, P J W, Rogier, J L H and Veerkamp, P** 'Knowledge engineering in design' *Knowl.-Based Syst.* Vol 1 No 2 (March 1988) pp 67–77
- 8 **Veth, B** 'An integrated data description language for coding design knowledge' in **ten Hagen, P J W and Tomiyama, T (eds)** *Intelligent CAD Systems I: Theoretical and Methodological Aspects* Springer-Verlag (1987) pp 295–313
- 9 **Brown, D C and Chandrasekaran, B** 'Knowledge and control for a mechanical design expert system' *IEEE Comput.* Vol 1 No 7 (1986) pp 92–100
- 10 **Mittal, S, Dym, C L and Morjaria, M** 'PRIDE: an expert system for the design of paper handling systems' *IEEE Comput.* Vol 1 No 7 (1986) pp 102–114
- 11 **Veerkamp, P** 'Multiple worlds in an intelligent CAD system' in **Yoshikawa, H and Gossard, D (eds)** *Intelligent CAD, I: Proceedings of the IFIP TC5 WG5.2 Workshop* North-Holland, Amsterdam (1989)
- 12 **Veerkamp, P, Akman, V, Bernus, P and ten Hagen, P J W** 'IDDL: a language for intelligent integrated interactive CAD systems' in **Akman, V, ten Hagen, P J W and Veerkamp, P (eds)** *Intelligent CAD Systems II: Implementational Issues* Springer-Verlag (1989) pp 58–74
- 13 **Gero, J S (ed)** *Artificial Intelligence in Engineering* 3 Elsevier (1988)
- 14 **Gero, J S (ed)** *Artificial Intelligence in Design* Springer-Verlag (1989)
- 15 **ten Hagen, P J W and Tomiyama, T (eds)** *Intelligent CAD Systems I: Theoretical and Methodological Aspects* Springer-Verlag (1987)
- 16 **Akman, V, ten Hagen, P JW and Veerkamp, P J (eds)** *Intelligent CAD Systems II: Implementational Issues* Springer-Verlag (1989)
- 17 **Yoshikawa, H and Gossard, D (eds)** *Intelligent CAD, I: Proceedings of the IFIP TC5 WG5.2 Workshop* North-Holland (1989)
- 18 **Sutherland, I E** 'SKETCHPAD, a man-machine graphical communication system' in *Proc. AFIPS Spring J. Comput. Conf.* Spartan Books (1963) pp 329–346
- 19 **Bobrow, D, Mittal, S and Stefik, M** 'Expert systems: perils and promise' *Commun. ACM* Vol 2 No 9 (1986) pp 880–894
- 20 **Dyer, M G, Flowers, M and Hodges, J** 'EDISON: an engineering design invention system operating naively' *Artif. Intell. Eng.* Vol 1 No 1 (1986) pp 36–44
- 21 **Kuipers, B** 'Qualitative simulation' *Artif. Intell.* Vol 29 (1986) pp 289–338
- 22 **Doyle, J** 'Expert systems and the 'myth' of symbolic reasoning' *IEEE Trans. Softw. Eng.* Vol 1 No 11 (1985) pp 1386–1390
- 23 **Tomiyama, T and Yoshikawa, H** 'Extended general design theory' in **Yoshikawa, H and Warman, E A (eds)** *Design Theory for CAD* North-Holland (1987) pp 95–130
- 24 **Moore, R C** 'The role of logic in knowledge representation and commonsense reasoning' in **Brachman, R J and Levesque, H J (eds)** *Readings in Knowledge Representation* Morgan Kaufmann (1985) pp 336–341
- 25 **Hayes, P J** 'In defence of logic' in *Proc. 5th Int. Jt Conf. Artif. Intell. (IJCAI-77)* Cambridge, MA, USA (August 1977) pp 559–565
- 26 **Coyne, R** *Logic Models of Design* Pitman (1988)
- 27 **Wegner, P** 'Capital-intensive software technology' *IEEE Softw.* Vol 1 No 3 (July 1984) pp 7–45
- 28 **Levesque, H J** 'The logic of incomplete knowledge bases' in **Brodie, M L, Mylopoulos, J and Schmidt, J W (eds)** *On Conceptual Modelling* Springer-Verlag (1984) pp 165–186
- 29 **Brooks, F P** *The Mythical Man-Month* Addison-Wesley (1975)
- 30 **Ramamoorthy, C, Shekhar, S and Garg, V** 'Software development support for AI programs' *IEEE Comput.* Vol 2 No 1 (1987) pp 30–40
- 31 **Bobrow, D** 'If Prolog is the answer, what is the question? or What it takes to support AI programming paradigms' *IEEE Trans. Softw. Eng.* Vol 1 No 11 (1985) pp 1401–1408
- 32 **Woodward, J R** *Computing Shape* Butterworth, UK (1986)
- 33 **Tomiyama, T and ten Hagen, P J W** 'Representing knowledge in two distinct descriptions: extensional vs intensional' *Technical Report CS-R8728* Centre for Mathematics and Computer Science, Amsterdam (1987) [Submitted for publication]
- 34 **Kitzmillier, C T and Kowalik, J S** 'Symbolic and numerical computing in knowledge based systems' in **Kowalik, J S (ed)** *Coupling Symbolic and Numerical Computing in Expert Systems* Elsevier (1986)
- 35 **Delahaye, J-P** *Formal Methods in Artificial Intelligence* North-Oxford Academic Publishers (1988)
- 36 **Stefik, M and Bobrow, D** 'Object-oriented programming: themes and variations' *AI Magazine* Vol 6 No 4 (1986) pp 40–62
- 37 **Goldberg, A** *Smalltalk-80: the Interactive Programming Environment* Addison-Wesley (1983)
- 38 **van Liere, R and ten Hagen, P J W** 'Introduction to dialogue cells' *Technical Report CS-R8703* Centre for Mathematics and Computer Science, Amsterdam, Netherlands (January 1987)
- 39 **ISO** 'The Programmer's Hierarchical Interactive Graphics System (PHIGS)' Second draft proposal 9592/1, *ISO document ISO/TC97/SC21 N1945* (June 1987)
- 40 **ISO** 'Graphical Kernel System (GKS) – Functional description' *ISO IS 7942* (July 1985) also cf. 'GKS-3D functional description' Draft international standard 8805, *ISO document ISO/TC97/SC21 N1414* (June 1987)
- 41 **David, B T** 'Multi-expert systems for CAD' in **P J W ten Hagen and T Tomiyama (eds)** *Intelligent CAD*

- 42 **Jacquart, R, Regnier, P and Valette, F R** 'GERMINAL: towards a general and integrated system for computer aided design' *Proc. 11th Des. Automation Workshop* Denver, CO, USA (June 1974) pp 352–358
- 43 **Simon, H** 'The structure of ill-structured problems' in *Models of Discovery (and Other Topics in the Methods of Science)* Boston Studies in the Philosophy of Science Vol LIV (Synthese Library Vol 114), D. Reidel, Dordrecht, The Netherlands (1977) pp 304–325
- 44 **Akman, V and ten Hagen, P J W** 'The power of physical representations' in **V Akman, P J W ten Hagen and P Veerkamp (eds)** *Intelligent CAD Systems II: Implementational Issues* Springer-Verlag (1989) pp 170–194
- 45 **Kieras, D E and Bovair, S** 'The role of a mental model in learning to operate a device' *Cognitive Sci.* Vol 8 (1984) pp 255–273
- 46 **Carnap, R** *Meaning and Necessity: A Study in Semantics and Modal Logic* University of Chicago Press (1947)
- 47 **Newell, A** 'The knowledge level' *AI Magazine* Vol 2 No 2 (1981) pp 1–20
- 48 **Bobrow, D G and Collins, A (eds)** *Representation and Understanding* Academic Press (1975)
- 49 **Kowalski, R** *Logic for Problem Solving* North-Holland (1979)
- 50 **Hodges, W** *Logic* Penguin, London (1977)
- 51 **Clocksin, W F and Mellish, C S** *Programming in Prolog* Springer-Verlag (1981)
- 52 **Sterling, L and Shapiro, E** *The Art of Prolog* MIT Press (1986)
- 53 **Hughes, G E and Cresswell, M J** *Introduction to Modal Logic* Methuen (1968)
- 54 **Lieberman, H** 'Delegation and inheritance: two mechanisms for sharing knowledge in object-oriented systems' in **Cointe, P (ed)** *Proc. 3eme Journees d'Etudes Langages Orientes Objects AFCET*, Paris (1986)
- 55 **de Kleer, J and Brown, J S** 'A qualitative physics based on confluences' *Artif. Intell.* Vol 24 (1984) pp 7–83
- 56 **Bobrow, D G (ed)** *Qualitative Reasoning about Physical Systems* MIT Press (a Bradford book) (1986)
- 57 **Minsky, M L** *Computation: Finite and Infinite Machines* Prentice-Hall (1967)
- 58 **Hayes, P J** 'The second naive physics manifesto' in **Hobbs, J R and Moore, R C (eds)** *Formal Theories of the Commonsense World* Ablex (1985) pp 1–36
- 59 **Forbus, K** 'Qualitative process theory' *Artif. Intell.* Vol 24 (1984) pp 85–168
- 60 **Faltings, B** 'A theory of qualitative kinematics in mechanisms' *Technical Report UIUCDCS-R-86-1274* University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL, USA (1986)
- 61 **Nielsen, P** 'The qualitative statistics of rigid bodies' *Technical Report UIUCDCS-R-87-1354* University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL, USA (1987)
- 62 **Raiman, O** 'Order of magnitude reasoning' in *Proceedings of the American Association for Artificial Intelligence (AAAI-86)* Philadelphia, Pa. (August 1986) pp 100–104
- 63 **Simmons, R** 'Commonsense arithmetic reasoning' in *Proc. American Assoc. Artif. Intell. (AAAI-86)* Philadelphia, PA, USA (August 1986) pp 118–124
- 64 **de Kleer, J and Brown, J S** 'Theories of causal ordering' *Artif. Intell.* Vol 29 (1986) pp 33–61 [Reply to following paper]
- 65 **Iwasaki, Y and Simon, H A** 'Causality in device behaviour' *Artif. Intell.* Vol 29 (1986) pp 3–32
- 66 **Gomez, F and Segami, C** 'Exploring the no-function-in-structure principle' *Journal of Exper. Theoretical Artif. Intell.* Vol 1 No (1989) pp 79–89
- 67 **Shoham, Y** 'Reasoning about change: Time and causation from the standpoint of artificial intelligence' *Technical Report YALEU/CSD/RR # 507 (PhD dissertation)* Yale University Department of Computer Science, New Haven, USA (December 1986)
- 68 **McCarthy, J** 'Programs with common sense' in **Minsky, M (ed)** *Semantic Information Processing* MIT Press (1968) pp 403–418
- 69 **Brachman, R J and Levesque, H J (eds)** *Readings in Knowledge Representation* Morgan Kaufmann (1985)
- 70 **Cunningham, W** 'The construction of Smalltalk-80 applications' *Working Paper Tektronix Computer Research Laboratory*, Beaverton, Or, USA (November 1985)
- 71 **Nielsen, J and Richards, J T** 'Comments on the learnability and usability of Smalltalk for casual users' *Technical Report RC-12080 (# 53339)* IBM T J Watson Research Centre, Yorktown Heights, New York (April 1986)
- 72 **Penney, D J** 'Implementing a Smalltalk-80 file system and the Smalltalk-80 system as a programming tool' in **Krasner, G (ed)** *Smalltalk-80: Bits of History, Words of Advice* Addison-Wesley (June 1984)
- 73 **Stefik, M, Bobrow, D and Kahn, K** 'Integrating access-oriented programming with a multiparadigm environment' *IEEE Softw.* Vol 3 No 1 (1986) pp 10–18
- 74 **Doyle, J** 'A truth maintenance system' *Artif. Intell.* Vol 12 (1979) pp 231–272
- 75 **de Kleer, J** 'An assumption-based TMS' *Artif. Intell.* Vol 28 No 2 (1986) 127–162
- 76 **de Kleer, J** 'Extending the ATMS' *Artif. Intell.* Vol 28 No 2 (1986) 163–196
- 77 **de Kleer, J** 'Problem solving with the ATMS' *Artif. Intell.* Vol 28 No 2 (1986) 197–224