Chapter 11

Designing with words and pictures in a logic modelling environment*

Aart Bijl

11.1 Introduction

At EdCAAD we are interested in design as something people do. Designed artefacts, the products of designing, are interesting only in so far as they tell us something about design. An extreme expression of this position is to say that the world of design is the thoughts in the heads of designers, plus the skills of designers in externalizing their thoughts; design artifacts, once perceived and accepted in the worlds of other people, are no longer part of the world of design.

We can describe design, briefly, as a process of synthesis. Design has to achieve a fusion between parts to create new parts, so that the products are recognized as having a right and proper place in the world of people. Parts should be understood as referring to anything - physical objects, abstract ideas, aspirations. These parts occur in some design environment from which parts are extracted, designed upon and results replaced; in the example of buildings, the environment is people and results have to be judged by reference to that environment. It is characteristic of design that both the process and the product are not subject to explicit and complete criteria.

This view of design differs sharply from the more orthodox understanding of scientific and technological endeavours which rely predominantly on a process of analysis. In the latter case, the approach is to decompose a problem into parts until individual parts are recognized as being amenable to known operations and results are reassembled into a solution. This process has a peripheral role in design when evaluating selected aspects of tentative design proposals, but the absence of well-defined and widely recognized criteria for design excludes it from the main stream of analytical developments.
We can identify the key characteristics of design *(Figure* 11.1) as:

(1) Design objects - subject to diversity of expression; different perceptions of things; lack of agreed abstract definitions;
(2) Design processes not problem solving in the orthodox form of problem statements that reveal solution paths; conflicting criteria for validating results; many solutions; and
(3) Design knowledge - no formal, complete and shared knowledge base; relies on integration of overt and intuitive knowledge; necessarily manifest in idiosyncratic design practices.

*This chapter is similar to a paper on Computer-Aided Design and Artificial Intelligence presented at the Reading ESCAD'85 Workshop on this topic, July 1985.

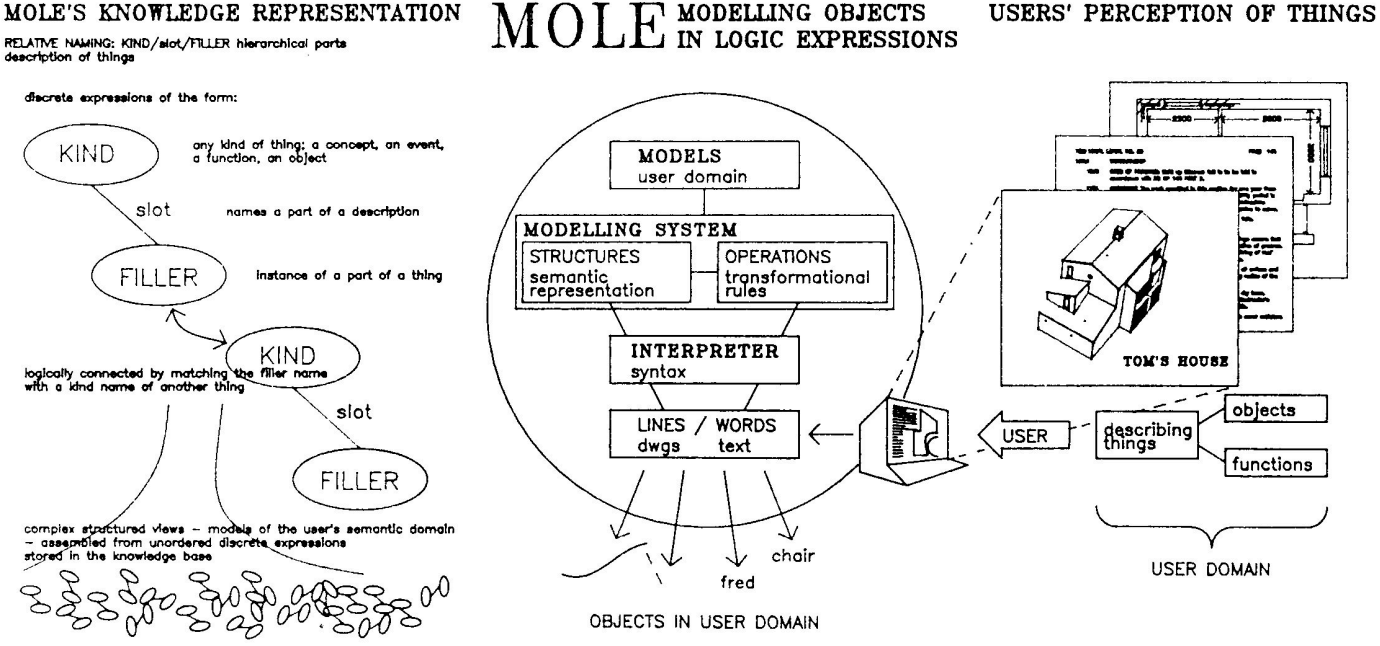

Figure 11.1. MOLE logic modelling system. (a) User domain exists in the heads of designers; (b) expressions of user perception conveyed to system by words and lines; (c) words and lines mean things to a user, mapping onto things perceived in the user domain; (d) system interprets expressions into MOLE representation of KINDs, slots, FILLERs and relationships between KINDs; (e) representation employs matching and inference to build up physically disjoint but logically connected parts and variant hierarchies describing things; (f) system constructs models that reflect what it has been told by a user and reveals what it 'knows' back through the system by means of text and drawings

Knowledge of designers, what each knows, can be achieved only in the external manifestations of their thoughts, in the drawings and words of designers. Intelligent behaviour is then perceived essentially as the ability to externalize thought, to give expression to both overt and intuitive knowledge (*Figure* 11.1). Good design is the product of other people's perception of a designer's externalized expression of thought which, in turn, is built on the designer's perception and interpretation of values and aspirations of those other people.

This understanding of design places responsibility within people and focuses research attention on formal expressions that pass between people.

11.2 Design

In the field of computer-aided design, theory has to provide a description of the world of designers that can inform our efforts in devising abstract models which, in turn, might find expression in the form of computer systems. The description contained in the theory does not have to be correct, it does not need to refer solely to indisputable facts, but it must be informative in describing conditions that are relevant to a system failing or succeeding.

It is possible to think of design as a particular kind of activity which may or may not employ well-defined methods, or as objects that are the products of design activity. It is tempting to categorize design according to classes of objects, identifying objects that have known bodies of overt knowledge associated with them. This knowledge might refer to conditions that have to be satisfied during the course of designing an object, and functions that the object has to perform which may partially define the object, as in the example of mechanical engineering design. Such knowledge can form the basis of well-defined methods which then are incorporated as parts of a design process. The temptation is to focus our attention on what we already know how to do, to devise methods that consist of overt stepwise operations and to equate such methods with the whole of a design process.

The weakness of this approach to design is that it is too dependent on anticipations of design objects, their particular properties and functions. When people's perceptions of objects change, the methods have to be reconstructed and so do the computer implementations of the methods.

The definition of design outlined in the previous section (and in *Figure* 11.1) recognizes that design products, objects, are inherently not predictable by overt procedures alone, nor by problem-solving methods operating autonomously in a computer. Design is more aptly described as an activity of event exploration, in which partial responses lead to redefinition of a goal.

This view of design is not intended to support the notion that design is somehow mystical. Instead, and quite simply, it recognizes a distinction between abilities of people that can be represented as overt knowledge and different abilities that remain within individuals as intuition. Overt knowledge can be recognized as knowledge that is represented in some formal environment outside of individuals, such as language depicted in words on

paper, so that the external representation includes the explanation of the knowledge. Intuition refers to knowledge acquired directly by individual experience, learned inwardly, that cannot be explained overtly. In any activity in which people are answerable to other dissimilar people we find overt knowledge informing intuition, and intuitive knowledge then appears to be decisive in determining people's actions.

11.3 Computer-aided design

The general goal of CAD should be to increase the expressive power of designers, enabling them to manipulate external representations of their own thoughts, relate those thoughts to knowledge of other people and convey conclusions to other people.

In seeking this goal we accept the notion of an integrated design system in which different interests in a proposed design artifact can be represented within a single model. This approach is supported by extensive UK experience of working systems for the Oxford Regional Health Authority (OXSYS), developed by ARC (Hoskins, 1977), and the Scottish Special Housing Association, the SSHA system developed by EdCAAD (BijI, 1978, 1979).

11.3.1 Examples of past practical systems

OXSYS employed a concept of parts as physical objects bounded and fully described within notional paraxial six-sided rectangular boxes. Boxes could be stacked alongside and on top of each other to result in assemblies of objects corresponding to compositions of parts, to represent whole buildings. The implications of this approach were that parts had to be thought of as a predefined library of discrete components, with no ability to deal with subsequent instance relationships between components that might propagate changes to the description of a component, and junctions between components could be described only with reference to paraxial planes.

SSHA took a slightly different approach. The equivalent of boxes were described in the context of each design for every instance of component. A box would receive a general specification of material content, excluding the details of boundary conditions, and the system would look at relationships with adjacent boxes to identify necessary junction details, to generate each unique component description. The implications of this approach were that parts had to be thought of as a predefined library of junction details, with components generated from junctions and sensitive to relationships with other components, but components could still be defined only with reference to paraxial planes.

These two approaches took their cues from different views of buildings, the first as constructions assembled out of manufactured components and the second as objects whose parts are shaped or poured in situ. It is noteworthy that in each case the system's ability to work only with paraxial geometry did not mean that the geometry of objects or parts had to be paraxial, as in the example of pitched roofs. Similarly, other restrictions to the system could be overcome by adding fixes in the form of further

program code. The trouble with these fixes is that they are local to each particular application.

Drawing and modelling things

We can make another observation about these two examples, focusing on the separation between the drawing part and the non-graphical part of these systems. The drawing part was modelled on geometric entities and the shapes provided by the drawing part form the basis for the building model. Other non-graphical properties that describe a building were represented as attributes and values attached to shapes. These attributes and values would provide the information required for design analyses and evaluations. The resultant separation between drawing shapes and describing other non-graphical parts of a building has the consequence that shape descriptions cannot be controlled by changes to descriptions of other properties; a designer cannot change other attributes and then expect the system to reveal consequential changes to shape attributes. The designer's changing perception of a design object has to find expression by means of the designer manipulating geometric entities, and manipulations have to conform to the knowledge of geometry encapsulated in the system.

This separate emphasis on drawing expressions of shape gain credence from the apparently dominant role of drawings in conventional design practice. However, this view ignores the connection between drawing and thinking about objects depicted in drawings, which occurs spontaneously in the heads of designers while they are drawing. If we acknowledge this connection and if we also accept that thoughts about other attributes can affect shape attributes just as much as shape can condition those other attributes, then we will be prompted to revise the abstract models on which current systems are based.

Expert systems

More recently, we have the phenomenon of expert systems (Michie, 1979; Lansdown, 1982). What can we expect from expert systems? These rest on the view that people, human experts, commonly employ knowledge that they cannot express overtly in the form of complete stepwise deterministic procedures when working towards some specified goal. This knowledge can be elicited from experts and can be represented in some non-deterministic, rule-based machine environment, typically built on inference rules of the form: if - (condition) - then - (replace with condition) -. Knowledge represented in a rule-based system can then be applied to new instances of problems by presenting the system with symptoms and goals. The idea is that expert systems can function in place of human experts.

We have already discussed design as being dependent on combinations of overt and intuitive knowledge and the decisive role of intuition. When expert systems are imported into the field of design the knowledge which they are intended to handle looks very similar to intuitive knowledge. We should therefore approach expert systems with caution. A weak anticipation of design objects, the lack of prior knowledge of the properties that will describe such objects, cannot be translated into the firm goal specifications required by expert systems. As a consequence, while the external products of the intuition of designers (as human experts) may be conveyed

to an expert system it does not follow that the system will be able to employ those products as knowledge when dealing with a new instance of design. The role of expert systems in design is inherently limited to discrete analytic subtasks of design, and cannot contribute to design synthesis. Thus expert systems cannot function in place of human designers when applied to design.

We should not believe that it is possible to elicit knowledge from experts which they do not know they possess, represent that knowledge within expert systems without ourselves having to understand the knowledge and then expect these systems to use such knowledge to produce meaningful results for people. The position presented here (rather informally) is not intended as a general refutation of expert systems but to moderate ambitions associated with practical applications of these systems.

In this brief discussion we have illustrated some of the deeper issues underlying computer-aided design systems. Notice that at this conceptual level we do not need any technical knowledge about computers. The issues present themselves as abstract ideas, referring to our perception of things and our ability to formalize our perception as overt expressions. This ability is manifest as theory which supports formalisms on which computer systems are built. What should now be evident is that theory is of primary importance in determining the practical utility of systems.

11.4 Questions that need answers

We need to develop theories that tell us how designers perceive things and how they perform operations on their perceptions, manifest in their overt expression of their thoughts. This knowledge should inform our efforts in devising new formalisms and new systems that can assist the work of designers. New theory needs to be general, abstracted from instances of designers' behaviour and not dependent on particular prescriptions for future behaviour. The need to accommodate anything a designer might think of will have the effect of linking efforts on design theory to equivalent efforts in other fields, an obvious example being natural language understanding.

To illustrate the kind of questions we might attempt to answer consider the world of things a designer might think about and things a designer might do to produce a design object. We might model this world in an orthodox fashion, in terms of discrete entities, relationships and operations. These constituents of the model imply meanings: entities might correspond to discrete physical objects, described in terms of their attributes; relationships describe the composition of objects into larger composite objects; operations enable transformation of objects and composite objects. These meanings imply as certain understanding of the world, a theory. What we want to know is whether the theory implicit in the model corresponds with a designer's perception of the world or what different theory is required that will generate new constituents for a new model.

Do designers think of things as discrete objects that are complete in themselves and compositions of objects that are made up of separate objects that are placed together? Is the description of one object changed

as a consequence of its relationship to another? When two objects are joined does each lose its identity as a separate object? Does the notion of entities as objects with physical boundaries have any general relevance to designers' perceptions of things during the course of designing? We can extend these questions and expose yet more questions at more fundamental levels of understanding, and it will be difficult to provide a theory that offers answers. Yet any action, any system that is implemented, perhaps unknowingly implies answers. In the context of practical applications these implicit answers are found in retrospect to be wrong when a system fails.

11.4.1 Alternatives to problem solving

In Section 11.1 design was described as not problem solving. We can explore this position as a useful way to expose questions. We need to question the concepts that we have imported from problem-solving fields.

A typical problem-solving approach can be identified by the following necessary constituents:

(1) A known state of being, within a single well-defined domain;
(2) Knowledge of procedures, available within the domain, by which a given state may be changed; and
(3) A goal expressed in terms that
    (a) Specify some new state, including the conditions that have to be met by a solution, and
    (b) Specify boundaries to the selection of procedures for changing the existing state.

This definition excludes design only if we add that a problem-solving approach has to rely solely on overt knowledge. When we do so we also exclude many other activities not usually associated with design. This observation suggests that advances in design theory are likely to prove significant well beyond particular fields of design application.

Further, we can identify the following conditions as being necessary for any problem-solving process. Any instance of problem has to have a start and a finish, and is wholely contained within those boundaries. Typically, problem specifications and solutions can be undone by moving these boundaries. The wholeness of a problem them gets decomposed into discrete parts, as subproblems, until parts present the conditions that are required by the available change procedures. To ease the task of matching part instances with procedures, emphasis is placed on prior typing of parts. Sequences of change procedures provide solution paths. Solutions are found by aggregating results, and problems have to have single (or very few) solutions. A match between a solution and a goal can then be recognized as a correct result. This view of problem solving rests on notions that are important to the internal functions of problem-solving systems:

(1) Wholeness of design goals;
(2) Differentiation between whole and parts;

(3) Discreteness of parts;
(4) Prior typing of parts; and
(5) Correctness of results.

These notions have become widely and firmly established in many fields. The effects on design are evident in computer-aided design systems. What we see is the selective decomposition of ill-defined design practices into well-defined subtasks that are amenable to a problem-solving approach. So we get programs that perform analytic functions to evaluate thermal performance and energy requirements of proposed designs for buildings. In most cases, these systems contribute nothing to our understanding of design synthesis, where synthesis has to reconcile disparate interests in a design object.

The question we have to consider is whether the notions that support problem solving are valid for design and, if not, can we formulate valid alternative notions?

Wholeness of things

In general, the start and finish of a design appears to be circumstantial. The context in which design occurs is people, and they decide when a design is to start and is finished. There are few overt criteria for recognizing that a design is complete, and in the case of buildings there are none.

The boundaries to any instance of design are ill defined. A design process includes response to ever-more unforeseen considerations. In the case of buildings these considerations come from an unbounded domain of any arbitrary subset of all people. We do not know how to draw a boundary around the design interest in a building so that we know we have the whole building.

As a typical instance the design of a door handle sits in the context of a door, in a room, in a dwelling, in a building, in a locality, in a town, etc. Equally, the door handle sits in the context of fire safety, affecting the ease of escape through doors by people with burnt hands, and it is relevant to social concern for welfare and health provision, etc. Furthermore, the handle has to be manufactured with available machining technology, marketed and installed. We can say more about the door handle but we don't know where to draw a boundary around it to define a complete and discrete design interest.

Whole and parts

Decomposition of design into parts has the effect that the parts are differentiated according to different domains, with no known relationships between domains that allow results to be aggregated into design solutions. In the case of buildings we do not know how to add the results of a thermal performance evaluation to a result of a daylight-distribution appraisal.

Design has to reconcile different arbitrary and contradictory interests in a design object, and this reconciliation cannot be achieved by overt processes alone. Parts are defined by the perceptions of different people, and their synthesis with respect to any perception of the whole is dependent on idiosyncratic contributions from those individuals.

Discreteness of parts

If we recognize the existence of parts we do not know how to define them as discrete parts. In the case of buildings a part tends to be defined by the context of other parts and changes to the context change the part. It is not practical to work with discrete parts where changes to one part are likely to propagate unforeseen changes to many others.

Prior typing of parts

Part instances occur as unforeseen events that do not permit confidence in prior typing. Attempts at typing either have to be undone as new instances make unforeseen demands on type specifications or they compel conformity of instances, which adds extraneous constraints on a design product.

The difficulty presented here becomes still more instructive if we also question the usual distinction between parts as objects, which may be represented by data, and parts as functions which can be performed by or on objects. It is not clear whether this is a useful differentiation for purposes of a theoretical understanding of design; it is possible to consider functions as parts of descriptions of objects.

Correctness of results

Lastly, we come to the notion of correctness of results, design products. Here we have to recognize that design goals generally do not include explicit specifications that allow us to match products so that we can know that we have correct results. In the case of buildings it is interesting to note that there is no abstract formal definition of a building that will enable us to tell buildings apart from other things. Nor do we have rigorous classifications of buildings that differentiate between, say, houses and offices among all other instances of buildings. In the absence of such knowledge, goals cannot be explicit, and we have to accept uncertainty in our solutions.

11.4.2 Search for alternative notions

What other notions can we think of that will be more appropriate to design and, incidentally, to many other human activities that exhibit properties similar to those of design? The answers could have far-reaching implications, but we are at present ill equipped to provide them. Our thinking is too much conditioned by established concepts associated with successful problem-orientated systems. We need to dig deep into theory, to bring to the surface concepts that exist in other theoretical fields, such as mathematics, linguistics and philosophy, and reshape them into a theory that describes design.

Meanwhile, we can identify promising questions. Can we think of parts as unbounded instances of things, with no notion of while and independent of types, just parts of parts of parts? Can we conceive of a notion of things that uniformly embraces objects and functions as well as activities and events? Can we represent relationships between parts that will enable us to bridge across arbitrarily different domains to work with divergent interests in design objects? Can we create an overt systematic environment that will be able to represent any unrehearsed thing in the head of any designer?

Referring to experience of established systems, the orthodox answer to all these questions would be no, but then we would not be able to design.

11.5 Help from artificial intelligence

To sum up our previous discussion, we cannot and should not reshape existing design practices into some coherent, explicit and complete system. A more promising strategy might be to devise a systematic environment that can reflect the knowledge of designers, accommodate knowledge from individual designers and allow designers to formulate and exploit their own design practices. Such a strategy focuses attention on the ability of computers to interpret anything that they might be told by designers, in any form of expression used by designers in words and pictures.

This strategy prompts us to look at techniques emerging from the field of artificial intelligence. To develop this theme we should recognize that Al itself is not a single, coherent and well-defined field. It covers a broad spectrum of interests loosely linked by a shared ambition to represent more of human intelligence in machines. This ambition spans a range from strong Al, aimed at machines that can act autonomously and in places of people, to weak Al, which expects machines to be responsive and supportive of people, to aid the actions of people. The latter is closely related to the ambition of CAD and, following the view of design presented in this chapter, we should expect most help from that branch of Al that is focused on linguistics.

11.5.1 Formalisms

Central to all Al, and not only Al, is the development of formalisms to represent human intelligence. We can think of a formalism as some conceptual idea that is developed and articulated into a well-formed abstract system. Typically, a system consists of some variant of generalized abstract representational entities, relationships and operations that, in combination, can be used to represent human perceptions of particular things - objects, events, functions - occurring in some world domain. The well-formedness of a formalism means that it can be defined wholely overtly, and that any operation which it supports will always result in a changed representation which conforms to the definition of the formalism. It is this well-formedness which makes formalisms central to any goal of representing knowledge overtly, detached from individual persons and in a manner that can be implemented as physical machine systems.

As an example, we can regard arithmetic as a formalism, with numbers as entities, plus operators to change arithmetic states. The results of applying any operators to any numbers always remain within the domain of arithmetic. Here we should note a further significant property of formalisms; any formalism is bounded and is not itself capable of dealing with things that lie outside its boundary. Typically we use arithmetic by tagging numbers with labels that refer to other things, such as apples and pence, and we can multiply the numbers to find the price of a dozen apples. We cannot, however multiply apples by pence or apples by oranges. The price

of apples is found by simultaneously operating more than one system, by mixing formalisms.

The boundedness of any formalism should be regarded as a temporal state, as a description of our understanding of things. Over long periods of time we can evolve formalisms to include new understandings. However, we are still left with some deep questions. All formalisms appear to conform to some metaformalism which determines general properties of each instance and which excludes certain kinds of human understanding. If we accept that the motivation for extending formalisms is a broad anticipation that they should support ordinary activities of people, then it remains unclear how we can mix formal systems with the informal systems presented by people. This last point is pertinent to designers and other people.

We can see formalism as a representational environment that is available within a computer system. The power of any formalism has to be assessed in terms of the range of things it can represent, its generality. Commonly in CAD, applications systems are based on formalisms that consist of entities which can represent properties of objects in some world domain, relationships between entities for modelling descriptions of composite things, and operations for transforming arrangements of entities. The problem in CAD is the close correspondence between its formalisms and its applications and the consequent invalidation of a formalism when a designer's expectations of an application change. Too often, effort on development of applications gives scant recognition to underlying formalisms, and we get formalisms embedded in applications that lack generality.

11.5.2 AI and linguistics

In Al the search is for abstract formalisms that are far removed from particular applications, which will support techniques that can be employed on many different applications. An example is provided by work on formal grammars and parsers to support syntactic analysis and semantic interpretation of natural language. It is instructive to note that separate efforts tend to be focused on selected fragments of language, identifying deep problems that challenge the validity of existing formalisms and prompt searches for new formalisms. A very long time is expected to elapse before separate efforts will converge on a formal system that will be capable of understanding people's informal language.

To expand on this point, briefly, consider the normal occurrences of strings of words formed into sentences. We can categorize words according to their roles as parts of speech, we can set up relationships between categories of words, we can let inflections of words influence relationships, we can let meanings associated with word symbols influence sentence structure and we may know how to apply analytical, interpretive and transformational functions to strings of words. We may know this through our human perception of language, but we are still a long way from achieving a formalism that can encompass all this knowledge. To illustrate effort applied to a fragment of language over the past decades (Kamp, 1980), we have:

Every farmer who owns a donkey beats it.

It is argued that this kind of sentence is not amenable to a purely syntactic analysis of an individual sentence and also requires knowledge of semantics associated with the words, resulting from previous sentences and represented in some knowledge base. In this instance, the problem is to identify what 'beats' what and with what effect on other instances. Can 'it' refer to something outside this sentence, and if 'it' is 'a donkey' how many donkeys get beaten? We might then weave a story around this sentence and say: 'The scene is the Spanish Civil War and the army is pouring over the hill, so every farmer who . . . ' What happens to the semantics associated with the word 'beat' if we now mean that the farmers 'beat it out of here'? At this point good linguistics will shrug their shoulders and look forward to the next few decades.

11.5.3 Natural language and CAD

The problems presented by natural language have similarities with those experienced in CAD. In both cases we are dealing with expressions in the form of words (and pictures), the need to devise some syntactic structure to aid interpretation and generation of expressions, the need for some form of representation to hold the content of messages conveyed by expressions, and we have to allow the changing contents of a knowledge base to influence interpretation. In both cases, our systems have to survive changing and unexpected demands made by people.

From the field of natural language we get the distinction between syntax and semantics plus arguments about their separation or interdependence. The distinction does not always remain clear, but a simple way of describing the difference is to say that syntax refers to an abstract structure that is contained within a single formalism, defining operations that can be executed within that formalism. Semantics refers to interpretation or denotation of syntactic tokens as represented in one formalism for purposes of passing them to another formalism, the interconnections between different formalisms. Thus, for example, we can view a parser as a system for analysing language expressions, whose operations are determined by the syntactic structure of some formal grammar. Separately, we have a formalism for representing knowledge, a different syntactic structure for linking entities, relationships and operators. If we then expect to use the knowledge base to verify the results of the parser, or to use the parser to modify the contents of the knowledge base, we need to interpret the output of one into a form that can be received by the other; the two formalisms have to understand each other. Note that in this context we are using a notion of formal understanding which should not be confused with people's informal sense of meaning.

To illustrate a link with CAD we can refer to another development of semantics in natural language (Shank, 1975). It argued that manipulation of words as symbols alone is not sufficient for language interpretation, and that a system also needs to be supplied with meanings in the form of semantic primitives against which words can be matched; in this case, semantics is closely coupled with actions within a knowledge base. The idea here is that it should be possible to devise a set of primitives that could be used to represent all meanings that people might perceive in some world

domain, rather like a model of human knowledge. The idea is similar to a notion in CAD, that it should be possible to devise a model of the design process which could correspond to all designers' individual perceptions of design. Both cases pose similar, deep questions.

A notable difference between natural language and CAD is the circumstances in which research investigations are conducted; the recognition of problems, the time scales which are applied to them and the validation of results. Natural-language projects apply a narrow focus to deep problems over long periods of time, working at the fundamental level of formalisms, and results are validated among linguists. Investment is justified by a broad and loose anticipation that some results will eventually have widespread practical application.

CAD projects typically take a broad sweep at relatively superficial problems, with targets set in shorter time periods, and results are validated in terms of their practical utility. Investment is justified by promises of immediate applications, resting on the belief that there are no deep problems or, if there are, then these will somehow be solved in other fields and drawn into CAD as a consequence of experience of failure. This pattern of circumstances explains the relatively poor standing of CAD research when judged by researchers in other fields, who see CAD as preoccupied by unquestioning efforts to apply borrowed techniques. This also explains that problematic record of CAD in supplying applications that can be widely adopted by the design professions.

11.5.4 Prolog logic programming

So far, our discussion of Al has focused on abstract formalisms that are targeted at perceptions of world domains, related to people. The general intention is that these formalisms should also be implemented in computers. Translation of abstract formalisms into physical implementations has itself prompted advances in Al; irritation with established programming techniques that impede researchers' efforts at making computers do things has prompted fundamental advances in software technology. The Prolog logic programming language provides a good example (Clocksin and Mellish, 1981). Prolog can be viewed as an abstract system consisting of a computationally treatable subset of resolution logic (Kowalski, 1979). More commonly, Prolog is viewed as an implementation of the system in the form of a general-purpose programming language. This language offers the virtue that all instances of expressions in the language have to be correct only with respect to the logic system - you can tell the computer what you want without anybody having to know anything about the consequential machine procedures. Thus we have an example of declarative access to computer resources.

The Prolog language employs techniques of symbol matching and inference which are visible to the user as an ability to recognize and match instances of the same words or names, and the ability to use inference rules to arrive at new information. Inference rules in Prolog take the form of... (true) ... if ... (condition) ... and ... (condition) .... etc. Matching is used to find if the specified conditions are fulfilled in a knowledge base, to establish whether the left part of the expression is true. Roughly, the left

part is viewed as a goal, the right parts refer to facts if they are present in a knowledge base and the combined expression describes a rule by which a goal might be established as a fact.

In principle, a computer application programmed in Prolog needs to consist only of expressions that conform to the logic system of Prolog itself. Interpretation from Prolog to lower levels of software is done automatically and entirely by computer. In principle, this development promises:

(1) Declarative rather than procedural access to computer resources, i.e. you say what you want rather than tell the machine how to find it;
(2) Removal of the prior condition of completeness of tasks as interpreted from a user domain into a computer system, by shifting the emphasis from procedural models to rule-based systems - you tell the system as much as you want it to know now, and more later;
(3) Removal of the distinction between data and instructions from considerations affecting computer applications - the corresponding distinction between descriptions and functions can then be regarded as a matter for end users and their perception of tasks in a user domain; and
(4) Eventually, removal of the distinction between programming and using computers, leading to removal of specialist programmers as interpreters beween users and computers - provided users can express themselves in the logic environment, they should be able to tell computers what they want them to know and do.

Qualifications need to be added to those promises. For inexperienced users the structure of Prolog expressions, its syntax, remains difficult to use - in future, one might expect a more natural language interface. The computer resources required for a full and responsive implementation of Prolog are far greater than can be afforded by modest potential users, and cut-down versions fail to exhibit the fundamental advantages of Prolog (1) and (3) above - this limitation should be removed by current progress on computer hardware design and production technology.

11.6 Formulating a modelling environment

Linking our brief exploration of Al with our earlier discussion of CAD we can now make some general observations. We should be wary of importing techniques simply because they exist in other fields and, upon superficial examination, they appear to offer benefits for CAD. When we do import techniques we should be prepared to question the concepts on which they have been built, to reshape them from CAD. We should expect CAD to generate fundamental investigations that will result in new techniques which can be recognized as valid contributions to other fields. A simple criterion is that research papers from CAD should be accepted by journals read in Al and related fields.

At Edinburgh, EdCAAD is working closely with the School of Epistemics on a European collaborative Espirit project. Our traditional concern in EdCAAD, supported by the UK Science and Engineering Research Council, is focused on modelling descriptions of design objects (buildings)

in a manner than can support arbitrary modification of descriptions by designers during the course of designing. With the same motivation that prompted Alto develop new software technology, EdCAAD developed an early interest in Prolog, which has resulted in its C-Prolog implementation. Now, on the Espirit project, we are studying a linguistic approach to graphics, to link drawings and text as complementary forms of expression for dialogue with a machine-resident knowledge base.

11.6.1 Intentions fora modelling environment

Current work on logic modelling illustrates the kind of thinking that EdCAAD applies to CAD research. The emerging system is called MoLe (Modelling objects with Logic expressions), and it is being built on Prolog (*Figure* 11.1). We are devising an abstract system, a formalism, for representing descriptions of things - objects, events, functions - which has to be viewed as distinct from its possible applications in any user's world domain. The intentions for the system refer to our earlier discussion of CAD:

(1) Wholeness of things: the formalism should not see a whole as different to a part, no separate status for a root part, and any part may become a component part of any other part;
(2) Discreteness of parts: there should be no notion of boundaries to parts or to assemblies of parts such that boundaries exclude other parts, and any changes to a description of a part should progagate consequential changes to any other part that it describes;
(3) Prior typing of parts: the formalism should not be dependent on any prior type definitions that have to correspond to subsequent types perceived in a world domain, and any part may inherit partial descriptions of any other part; and
(4) Correctness of results: the formalism should not imply any notion of an independent or absolute arbiter of correctness, it should reflect only what it has been told by users, plus what is can infer from what it has been told.

In any application, users might want to think in terms of part/whole distinctions, discreteness of parts, typed instances and criteria of correctness. However, then it is the users' definitions of these notions that need to be reflected in the formalism and it should not impose any penalty on users when they change their mind.

11.6.2 Descriptions

The MoLe formalism is based on a notion of relative naming. Words are used to label things, and things are described by similarly labelled other things, linked by labelled relationships. Thus we get a formal structure which consists of a kind, for any kind of thing, a slot which names a part relationship to the kind, and a filler which names an instance of some other thing that forms part of the kind. Composite logical structures are then built up by matching filler and kind names, so that we get hierarchical parts descriptions of things.

Because these hierarchies remain abstract and are not bounded by any consideration of physical implementation, we can proceed to define the constituents of this formal structure so that we achieve our desired logical modelling environment. First, all kind, slot and filler names can be any words declared by users when using the system. Kind names have to be unique for any one thing that has a unique description and, in the case of different instances of the same thing, the system can help by automatically adding suffixes to maintain the uniqueness of kind names. Filler names can be kind names, or they can be terminal values or instructions. Slot names have to be unique to any group of slots directly associated with a kind name. These are the basic conditions that are necessary to ensure that single parts-hierarchies can be arbitrarily created and modified, and that their representation within the formalism will always be unambiguous.

We then want descriptions to inherit parts from each other, to set up links across separate parts-hierarchies. We do this by making a part of one description be an instance of another part which already has its own description, and then changing the inherited description of the new instance. Thus we get instances with single linear chains of antecedents. More ambitiously, we can make a part inherit partial descriptions from several other parts, resulting in variants that involve multiple inheritance. The formal problems which are generated are non-trivial, especially when the intention is to allow users to make arbitrary modifications to complex descriptions.

11.6.3 Expressions

Expressions are the means by which things are described in the formalism of the modelling environment, and they are the means by which the environment reveals what it knows. User expressions are of two types; change expressions which change descriptions, and query expressions that reveal a view of a description (without changing it). It is our intention that the structure of expressions should ensure that all instances are:

(1) Logically resolvable as true or false;
(2) Interpretable as queries that receive answers; and
(3) Subject to change operators, so that changed expressions always conform to the abstract definition of a description.

11.6.4. Drawings

In addition to words, drawings are also regarded as a kind of expression for purposes of communicating with the modelling environment. Here, drawings have to be regarded as objects that serve as partial descriptions of things that are represented in MoLe. Thus we need a general understanding of drawings, which we can describe to the system so that it can use this understanding to represent any instances of drawings. We are developing general abstract definitions of constituents and relationships that occur in drawings, in terms of drawing compositions, complex and simple shapes, segments, construction points and, as lowest-level primitives, construction lines. Change operations then involve composition, decomposition and

transformation (Szalapaj and Biji, 1984). The structure has to support any designer getting hold of any parts of a drawing, as parts of objects that the drawing depicts, and support a designer arbitrarily changing drawing parts to change the system's representation of the designer's perception of things.

This brief sketch of current work has glossed over many issues, including some that yet remain to be resolved. The question of a suitable user interface has not been touched, the syntax of expressions remains awkward and has been developed only for purposes of testing logical coherence. The deeper question of meaning, as something that might be considered separately from expressions and might be represented overtly within the formalism, remains a matter of contention. Here, on the spectrum from syntax to semantics, we lean more towards syntactic analysis, though we remain uncertain about the distinction. Implicit in this work is the view that expressions are the externalized manifestation of human knowledge and intelligence, that expressions can be subjected to interpretation in an overt representational environment, that representations may have the purpose of executing transformations and that the purpose of transformations is to give different views of knowledge as conveyed by expressions. However, meaning, in the sense that it governs human actions, occurs only in the heads of people.

11.7 Conclusions

Design has been described as not problem solving, leading to conclusions that have implications on many other human activities. Formal representational environments implemented as computer systems must not be dependent solely on an analytic model of human tasks, and must be capable of reflecting events in the course of design synthesis. We need to question currently established notions, i.e.:

(1) Wholeness of things;
(2) Differentiation between whole and parts;
(3) Discreteness of parts;
(4) Prior typing of parts; and
(5) Correctness of results.

A brief review of Al identified the role of abstract formalisms as essential to the goal of representing human intelligence in machines and the importance of generality. Natural language can be viewed as a branch of Al that has particular relevance for CAD, both fields being concerned with the structure and interpretation of expressions, leading to systems that have to survive changing and unexpected demands from people. Differences in the circumstances of these two fields of research were highlighted, drawing attention to the relatively superficial nature of efforts in CAD.

Coming from another branch of Al, the Prolog logic programming language represents a notable advance in software technology, aimed at making it easier to translate abstract formalisms into physical machine systems.

Our interest in Al is prompted by our view of design as something

people do, as a kind of process rather than a class of objects. We are interested in the principles that emerge from Al's search for formalisms to represent human intelligence, and in the resulting systems that can reflect and support people's activities in a world domain. The link between CAD and Al poses fundamental questions. Whilst accepting the central role of formalisms, what, in general, can we know about what is excluded by formalisms? For activities that occur between people, how do we develop confidence in accepting any exclusive effect of a chosen formal system? Syntax can be understood as formal structures within single formalisms, and semantics as interpretations of results between different formalisms. What can we expect of a formal treatment of semantics that will improve communication between a computer system and the informal behaviour of people? These questions imply unattainable goals, but they are useful in ensuring progress towards more widely acceptable practical systems.

In CAD we need to become much more deeply involved in research, less as borrowers of techniques from other fields and more as contributors to new advances that will also be valid in Al and related fields. We need to give more recognition to the role of theory and formalisms in determining the practical utility of systems.

Acknowledgements

The work on logic modelling described in the last section of this paper is being supported by the UK Science and Engineering Research Council. Thanks are due to Sam Steel, a logician who set us on the path of relative naming, and the Ramesh Krishnamurti for adding mathematical rigour to our thoughts.

References

BIJL, A. (1978). Machine Discipline Practice. *Proc. CAD 78,* Brighton, UK

BIJL, A. (1979). Computer Aided Housing and Site Layout Design. *Proc. PARC 79,* Berlin, FRG

BIJL, A. (1984). Computer Literacy: Designers' Despair or Hope? *Proc. Design Research Society Conf. on the Role of the Designer,* Bath, UK

CLOCKSIN, W. and MELLISH, C. (1981). Programming in Prolog, Berlin: Springer-Verlag

HOSKINS, E. (1977). The OXSYS System. In Gero, J. S. (ed.) *Computer Applications in Architecture,* London: Applied Science

KAMP, J. W. (1980). A Theory of-Truth and Semantic Representation. *Report Cog. Sci.,* Univ. Texas, Austin, USA

KOWALSKI, R (1979). *Logic for Problem Solving,* New York/Amsterdam: Elsevier/North-Holland

LANSDOWN, .J. (1982). *Expert Systems: Their Impact on the Construction Industry,* RIBA Report, UK

MICHIE, D. (ed.) (1979). *Expert Systems in the Micro-Electronic Age,* Edinburgh: Edinburgh University Press

POPPER, K. R. (1963). *Conjecture and Refutation: The Growth of Scientific Knowledge,* London:
Routledge and Kegan Paul

SHANK, R. C. (1975). *Conceptual Information Processing,* Amsterdam: North-Holland

SZALAPAJ, P.J. . and BIJL, A. (1984). Knowing Where to Draw the Line. *Proc. IFIP Working Conference on CAD,* Hungary