HUMAN KNOWLEDGE: AI & CAD

Aart Bijl
EdCAAD, University of Edinburgh, Department of Architecture
20 Chambers Street, Edinburgh EH1 1JZ
Scotland

Perceptions of design and CAD are discussed, leading to a distinction between human knowledge and machine representations of knowledge. A strategy for "mechanistic" symbol processors is presented, employing "mechanisms" of formal logic to manipulate written and drawn expressions of designers' knowledge.

Keywords: knowledge, notions, representations, formalisms, symbols, drawings, design.

INTRODUCTION

We have now experienced two decades of CAD. In that period, the promise of CAD has met with obstacles presented by: firstly, the essentially idiosyncratic nature of design practices, particularly in loosely constrained fields; secondly, the prescriptive nature of conventional computer technology; and, more recently, the assumptions underlying machine intelligence.

It is time to return to two fundamental questions:

1. What can we know about design that will inform our efforts in CAD?

2. What can we know about human intelligence that will inform our efforts to link AI and CAD?

The field of CAD requires theories of design that embrace human designers. By designing CAD systems - focusing on machine systems - we are in effect designing designers. All CAD systems have in-built anticipations of the behaviour of designers who will be invited to use them. The orthodox argument is that these systems relieve designers from the routine and uninteresting tasks involved in designing. The assumption underlying this position is that we know what is tedious to designers. To test this assumption, consider each of us as designers and then ask whether we would be happy to let other system designers decide what each of us finds interesting when we design CAD systems?

A theory of design does not need to explain all that goes on within individual designers when they design particular things. CAD is not committed to producing design machines. Yet a theory does need to explain enough about the whole of design activity in order to define intended relationships between CAD machines and human designers. CAD is targeted at generalised representational environments that people will find useful when they design things. The question then is how much design knowledge needs to be incorporated within a system in order to make it useful?

Many design theories variously rest on a distinction between function and object, in which function refers to design requirements and object refers to something that possesses attributes or properties that satisfy the requirements. Typically, designing is then conceived as some process of problem solving, employing problem decompositions followed by aggregation of partial solutions. Along this same line of thought, design is also seen as search and refinement. The effectiveness of this approach to design appears to be related to application fields in descending order of success:

1. Electronic Engineering: "using computers to design computers", which has the advantage that system designers and users use (more or less) the same language.

2. Mechanical Engineering: system designers and users have different perceptions, but application tasks are regarded as being predominantly functionally determined.

3. Architecture (or building design): system designers and users have different perceptions and functionality is implicit or is subject to apparently arbitrary redefinition.

In previous papers [1,2] I have argued that the view of design presented by experience of architectural applications is more representative of design in general. Design activity is different from other activities that are already amenable to computerised techniques. Design is more a process of purposeful event generation and event exploration which refers to individual human intuitions and judgments, with no separate criteria for correctness of results. Good designs are decided, never proved.

This latter approach to design should condition our expectations for CAD. Design is not problem solving, it does not decompose into discrete components (abstract or concrete) and it is not defined by prior typing or classification of components (as perceived in a design domain). Instead, components have a transient existence and get redefined during instances of design activity. Design is evolutionary in the sense that changes occur in imperceptible steps, not directed by prior goals, that become apparent over periods of time - a normal and on-going characteristic of human knowledge.

Finally, in this brief review, the problems that need to be addressed by CAD are not simplified by the familiar distinction between design assistants or apprentices and automated design machines. Human design assistants share the same world as the designers whom they serve, share too much knowledge and share the designers' ability to evolve knowledge. Thus the ambition of machine assistants is not substantially different to that of design machines. Any imposed simplification will have the effect of redefining designers.

REPRESENTATIONS AND KNOWLEDGE

To make headway in CAD, we need to consider how people represent what they know. Figure 1 depicts a person as a bounded human being. We attribute properties to these beings, which we indicate by such words as *intelligence* and *knowledge.* We think that these properties are acquired through some combination of physiological "mechanisms" and world experience within people - without explaining how or what, we accept that knowledge does exist within people.

The same figure also shows objects that are made by people, that exhibit some of people's inner knowledge, or intuition. Such objects include text and drawings executed in some medium, on paper or a computer display screen. We read such objects as symbols and symbolic constructs, in the case of text, or more as analogic depictions in the case of drawings. Analogic depictions here refer to objects that exhibit properties which are the same as corresponding properties of other objects which they depict, such as shape properties. Interpretations of symbols and analogues within ourselves then tell us something about the knowledge which we exhibit to each other.

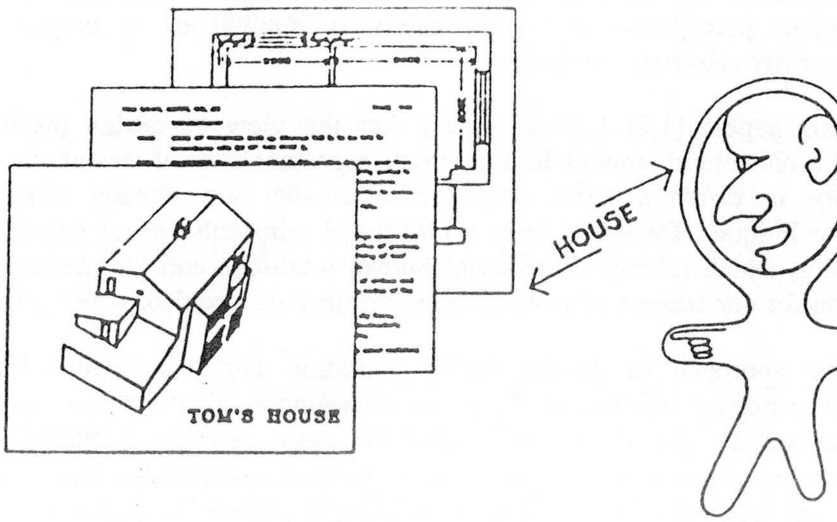

Figure 1: REPRESENTATIONS ARE NOT KNOWLEDGE
Externalised representations are artefacts which people make in order to express some part of what they have in mind. Such artefacts, at any level of formalisation, are not the same as human knowledge - they remain partial representations of all that humans might sense.

The key point that now needs to be recognised is that all overt expressions are conditioned by formal systems that are employed in their execution. Their formalisms constitute bounded worlds that limit the scope of possibilities for expressive objects.

We tend to lose sight of such limitations when humans exercise interpretive ability. We forget we are dealing with objects and think only in terms of interpretations, and somehow we adjust to differences between interpretations within each of us.

We then proceed to do something rather odd. There are people who believe that our interpretations, our knowledge, can somehow exist separately from us within these expressive objects. The oddity of this position becomes apparent when we consider the possibility of symbolic objects being animate and exhibiting design knowledge that you or I might variously know, as in the ambition of AI. Usually researchers depersonalise this position by targeting computers at other people's intelligence.

For certain purposes, this belief in objective knowledge seems to be practical. It is practical when we can satisfy ourselves that overt knowledge includes all that is necessary to produce useful effects. Overt knowledge, as formalised thought, applies when we believe that individual thought from people is no longer required.

EXPRESSIVE ENVIRONMENTS

Figure 2 offers a reassessment of the role of symbolic expressions related to human knowledge and machine processes. Knowledge refers to human notions. Here, notions are thought of as something extending beyond symbols, as part of human being, without implying anything about the actual substance of notions. Notions can be considered to exist as interpretations between symbols and anything else that might occur in the human mind, including states of mind that may be shaped by the totality of human experience (including sight, sound, touch, taste and smell). Domain knowledge then refers to specialised knowledge that people attribute to other people.

Expressions occur as externalised physical manifestations of people's knowledge, as representations of human knowledge. In the normal case of symbolic expressions, prior to computers and in the familiar forms of words and mathematical notations, expressive objects are used to evoke notions, to prod thoughts within people. We should not expect symbolic constructs to possess notions about themselves. This implies, for example, that mathematical notions exist *not* in text books but in the minds of mathematicians and other people who write and read the books. We can make two further general observations. Human validation of knowledge, in the sense of knowledge receiving variously accepted meanings with reference to human notions, cannot be contained wholly within overt expressions. As a consequence of this dependence on interpretations within people, symbolic representations of knowledge do not need to be complete. For example, we do not need an overt description of the syntax and semantics of natural language before we can know how to use it.

Worlds of symbols include relationships and functionalities that can be applied to symbol types, constituted as formalisms for producing compositions and supporting analyses of instances that might represent human knowledge. Processing of symbols then employs the functionality of symbolic logic, with mathematics as a highly developed example. Such processing is commonly executed by people who are literate in the symbolic language. When this processing is executed autonomously within

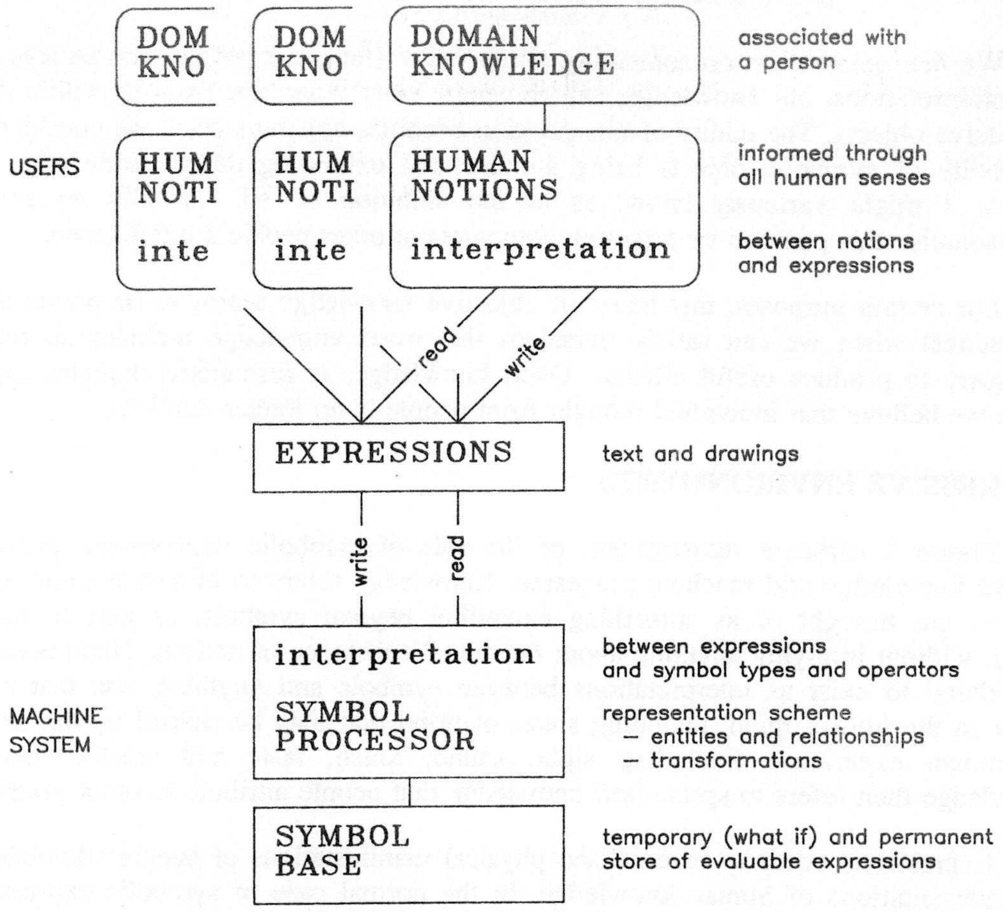

Figure 2: SYMBOL PROCESSORS
Normal literacy is illustrated in the upper half of the diagram. Domain knowledge associated with individuals and like-minded people is exercised through human notions and is made overt in externalised expressions. These expressions evoke notions within the same or other people. Computers, in the lower half of the diagram, offer functionality that is directed at the form of expressions, supporting the formal environments in which expressions can be generated and manipulated.

machines, we have computers.

As a general strategy for computers, we should try to differentiate between knowledge required to process symbols, and people's domain knowledge that might be represented by instances of symbolic expressions. This is not an easy distinction, and it is confused by the use of highly evocative words to indicate abilities of computers, such as *artificial intelligence* and *knowledge engineering*. To redress the balance, we should think of computers more as mechanical devices for operating on symbolic objects. The position that I am advocating appears to be in tune with criticisms of AI by Dreyfus [3] and Searle [4] but, in our case, we are not seeking to resolve the debate in AI. Instead we are looking for advances that might be appropriate to CAD. Such advances should employ formal logic to provide the "cogs and levers" for manipulating symbols, as an extension of the kind of functionality provided by "dumb" word processors. What is being suggested here is that we should maintain a clear distinction between the internal structure or syntax of a system and a user's semantics for expressions that are processed by the system. The system's semantics should be limited to interpretations of the user's demands for edits to its current syntactical representation of the user's expressions. Users, knowing the syntax of the system, might then be able to use it to construct and manipulate expressions of their own design knowledge..

WE ALL USE FORMALISMS

The upper part of figure 3 illustrates a familiar formalism that each of us uses every day. This formalism consists of a symbol type, and aggregations of instances that each have a single uni-directional "next to" relationship. Aggregations are related by empty spaces and special symbols that have meanings like *and* and *stop,* to form compositions. Further relationships can be identified by matching similar instances of symbols.

The lower part of this figure reveals this formalism to be the basis for a "writing machine", or word processor. This is *not* a natural language system; it deals only with those parts-of-speech that occur as parts-of-writing. The basic symbol type is a character and aggregations of instances form written words that approximate to spoken words. Special symbols occur as punctuations only in writing and are not spoken, though they might be implied by intonations or pauses in speech. The ability to translate between spoken utterances and written expressions, plus the ability to make further interpretations into meanings within other forms of knowledge representation, is the goal of natural language systems - an ambitious goal, beyond our present consideration of CAD.

A writing machine deals only with written objects. Thus, in the illustration, THINGS that appears as a character string is very different to the thing that is a character. We can use the writing machine to manipulate characters to produce words to signify different things that are not words. Examples of writing machines are pen and paper, mechanical typewriters and electronic word processors. This spread indicates a progression from artefacts that embody very little knowledge about writing objects, to artefacts that represent quite a lot of knowledge about writing objects. In all cases, they stop short of knowing what is being signified by any instance of writing. This lack of interpretive responsibility, and their obvious dependence on humans for any interpretation of expressions passing between people, makes these machines widely useful to many people.

The virtue of writing machines is that they share a common formalism which consists of few symbol types and relationships. We have learned to employ this formalism to construct partial representations of a vast range of human knowledge, from scientific methodologies and problem solving techniques to the rich ambiguities and contradictions of evocative poetry. This full range is important to all interactions

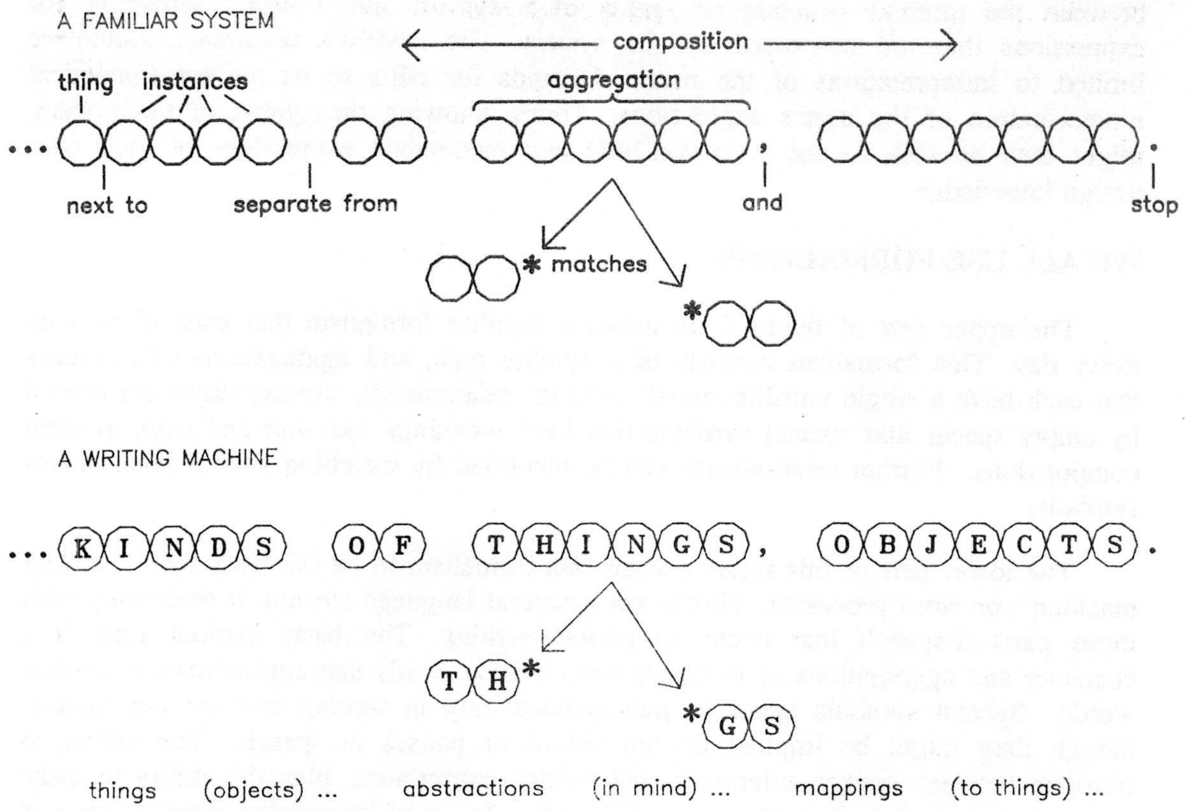

Figure 3: FORMALISM OF A WRITING MACHINE
The formalism shown in the upper part of the diagram consists of characters and instances of characters held together by a *next to* relationship. Aggregations, which we recognise as words, are differentiated by a *separate from* relationship. Punctuations identify further relationships between parts of compositions. Words that are instanced in writing then are different things to the constituents of written expressions - they receive interpretations in human language outside the writing system.

between people, and to CAD.

Writing machines illustrate the meaning intended for symbol processors in figure 2. The question that now presents itself is: Can we envisage a development of writing machines that will handle more varied and multi-directional relationships between words, and can we link them to drawing machines? Can such a development preserve the generality of these machines and what new kind of literacy will they require from people?

AN EMERGING FORMALISM

Figure 4 shows the structure of expressions supported by the MOLE logic modelling system [5,6] - a tentative and modest extension of a writing machine which supports multi-directional relationships between symbols. The MOLE system forms part of a theoretical exploration of fundamental CAD strategies which is being undertaken at EdCAAD and is supported by the UK Science and Engineering Research Council.

The concept that is being developed here sees words as symbols that stand for things and parts of things in the minds of people, and sees other symbols as standing for relationships between words, which can be used to model descriptions of things. These relationships have general definitions for users and they have interpretations down into machine procedures, giving the system its ability to manipulate descriptions. The form of expressions generated by the system and its users is the same, with special (non-verbal) symbols alerting users to system functions. All expressions generated by users can be read as being declarative, telling the system what it has to do, and expecting the system to produce intended results.

Expressions consist of three basic symbol types: kinds (K, for any kind of thing); slots (s, denoting a part of the thing); fillers (F, referring to any other thing that instances a part). This representation has an affinity with frame systems [7], but without any definition of frames as distinct from arbitrary collections of slots; and it also has an affinity with semantic networks [8], but without the system reading significance in names of arcs. Any instances of MOLE's symbol types can be indicated by any user-declared words, and the system's functionality is applied only to instances of types, without reading significance in the words that instance them. The further special symbols denote the relationship of a slot as being a *part of* a kind, and a slot's filler as an *inheritance* relationship to something else. Punctuation of composite expressions occurs in a similar manner to normal writing.

A kind name (in upper case characters) refers to something that a user has in mind. A slot label (in lower case) attached to a kind denotes a part of that kind. A filler (may include references to further kinds and slots, as path names) attached to a slot denotes something else that instances a part, which may be matched to another kind name. Matching has the effect of linking expressions, to form virtual hierarchies of parts into composite descriptions of things. Through inheritance relationships, descriptions can include views of parts of other things, resulting in interconnected parts hierarchies.

The central idea here is that we should be able to devise a machine for processing symbols so that expressions can be read by people as being meaningful, and so that expressions make the functionality of the machine visible to users. Words that form parts of expressions remain the responsibility of users, and relationships between words are maintained by the system. We should then be able to make the system do things with words and, if we can satisfy ourselves about mappings of words to other things (such as parts of drawings), we should then expect the system to perform useful

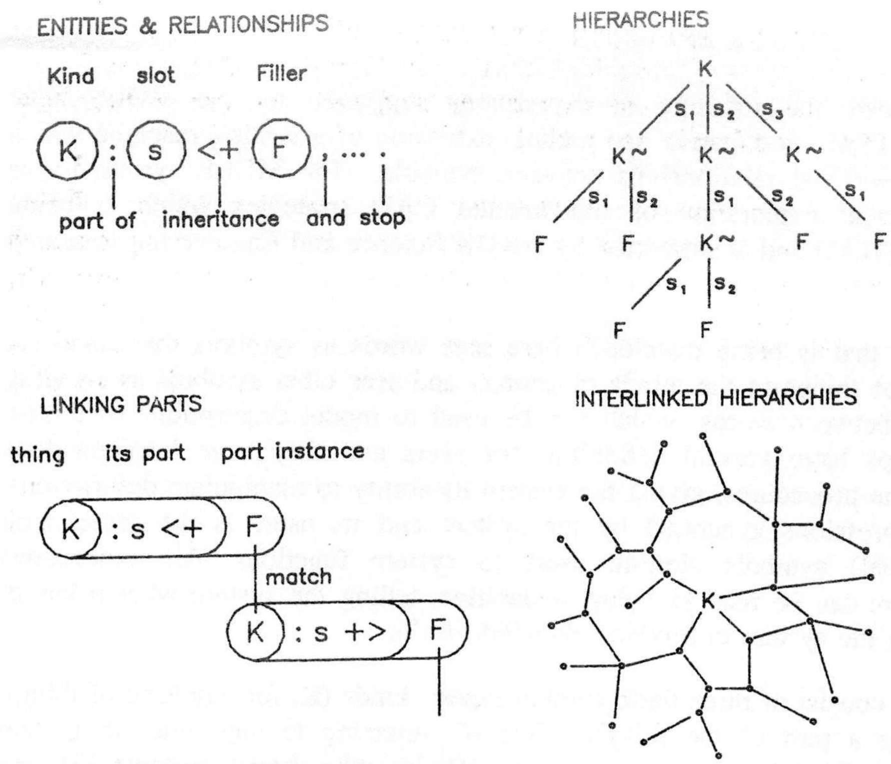

Figure 4: PARTS HIERARCHY
MOLE's representation scheme employs three main symbol types that are instanced by user-declared words, which are kinds (for any kind of thing), slots (for their parts) and fillers (for instances of parts). Triples of these types are held together by system-defined relationships selected by users, which determine matching across triples to build up hierarchies of parts that describe things. Inheritance results in interlinked hierarchies.

tasks on other things.

SOME SYMBOLS KNOWN TO THE MOLE SYSTEM

<+	slot filled by a new *instance* of F.
+>	slot filled by F.
>>	slot filled by *view* of F.
:	whatever follows instances a *part-of* K.
::	expands to a *path* through a sequence of part-of instances of K.
?	denotes queries and identifies *search space* for candidate answers.
{ }	denotes queries and sets *conditions* on candidate answers.
|	exclusive *not* conditioning answers.
<=	*description* of K gets *replaced* by description of F.

This is a fairly comprehensive sample of system-defined symbols included in the system as presently developed. The first three referring to inheritance relationships are the most critical. Each qualifies how a kind sees a part through its slot, where a part is likely to be some instance of a previously declared kind, and each also qualifies how a change to a part may be declared from a kind.

Instance inheritance, <+, makes the part a *new instance* of its former self, which continues to inherit its previous description, but subsequent changes to sub-parts have the effect of masking corresponding inherited sub-parts. Reverse inheritance, +>, makes the part the *same as* the inherited instance (of some other kind), and subsequent changes to sub-parts will be seen by all kinds that are partially described by the same instance. Indirection, >>, serves as a *read-only* instruction, to view an instance of a part of another kind.

The penultimate three symbols are used to express queries to the system, to find out what is already stored in the knowledge base and incorporate answers as parts of further assertions declared by users. Question marks at the end of path names are used to control the extent of search space for answers, and curly brackets contain conditions that have to be matched by answers.

WHAT MOLE KNOWS OF PARTS OF DRAWINGS

Expressions can include words which map onto drawing parts, and relationships that represent the general structure of line drawings. They refer to drawing parts as: construction lines with angle values, construction points at the intersections of construction lines, drawing line segments as portions of construction lines delimited by construction points, shapes as chains of segments (open or closed), and compositions as connected shapes (under varying conditions of attachment).

MOLE's symbolic representation of drawings includes no coordinate point values, yet this representation is sufficient to regenerate instances of drawings - a logical representation of the general structure of drawings, attachments and transformations, is being developed as a PhD by Szalapaj [9]. The chosen drawing primitive is a construction line, more in keeping with conventional uncomputerised drawing practice. Coordinate values might be a property of a drawing space (gridded paper or a computer display), in which case they need to come into play only to locate a drawing on a particular drawing space.

A separate drawing machine, a computer (or a person), then passes symbolic representations of instances of drawn objects to MOLE, and these can reflect edits and transformations applied to drawings. Representations of drawings can be linked to further representations of other things that they depict, so that drawings can form parts of descriptions of other things. Similarly, changes made to representations in MOLE can have the effect of driving the drawing machine, to result in changed drawings.

A WORKED EXAMPLE

To illustrate how MOLE expressions can be used to make and change descriptions, we can return to earlier experience of integrated CAD systems [10] and consider the example of walls and junctions between walls. We might want to avoid the condition of orthogonality, as shown in figure 5 below.

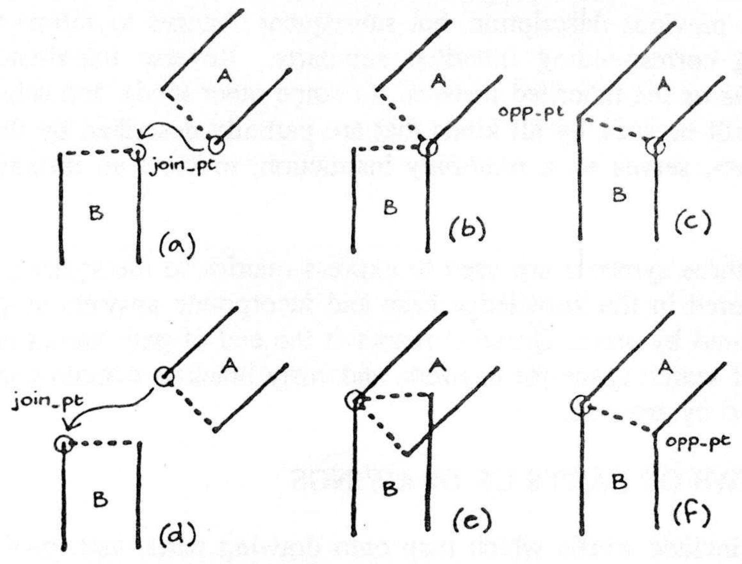

Figure 5: PROBLEM OF JUNCTIONS
Drawings of wall components, stored with square ends, need to be modified when walls are brought together to form junctions. Faces and ends have to be rotated and stretched, to result in continuity of wall parts, for any angle of junction.

DESCRIBING JUNCTIONS

```
JUNC_END:
    [join_pt +> fix_wall:join_pt,
     opp_pt:
        [conline1 +> join_wall:face?:|{conpt?{origin}}:bearer,
         conline2 +> fix_wall:face?:|{conpt?{join_pt}}:bearer].
    junction:
        [dwg:
            [segment:
                [conpt1 +> join_pt,
                 conpt2 +> opp_pt,
                 style <+ DOTTED]]],
    change:
       [join_end +> join_wall:end?:{conpt?{origin}}:
           [move_pt +> join_end:conpt?|{origin}:
               [move_to +> move_pt<=opp_pt]],
        fix_end +> fix_wall:end?:{conpt?{join_pt}}:
           [move_pt +> fix_end:conpt?|{join_pt}:
               [move_to +> move_pt<=opp_pt]]]].
```

This is a composite MOLE expression which describes the general case of wall end-on junctions. It expects that walls and their drawings, and the fact that two walls are going to take part in a junction, have been or will be described. We then declare a junction to be a kind of thing, giving it any name: JUNC_END. The junction is described by any number of parts, indicated by slot labels. These parts are a *join point* an *opposite point,* a *junction* part, and a *change* part. Each of these parts inherit descriptions of other kinds indicated by the slot fillers.

Thus the *join point* has a reverse inheritance relationship with the kind that fills the *join point* part of the kind that fills the *fixed wall* part of something that is not as yet identified in this description. The filler of this junction description's *join point* is, in effect, a path name to some other kind. The reverse inheritance means that this description can effect changes to the other kind.

The *opposite point* part is filled by an un-named kind that is further described by two *construction line* slots that are filled by kinds that are identified by the following conditions. The first *construction line* is filled by the same kind that fills the *bearer* part of any *face* of a *joining wall,* provided that it does not include within its description any *construction point* that is the same as the *origin* part of the *joining wall.* Notice, once more, that the *joining wall* is a slot label indicating a part of something that is not as yet identified in this description. The second *construction line* is treated in similar fashion.

We have now described the *join point* and *opposite point* of any junction as shown in figure 5.

The *junction* component part identifies the mitre at the junction as a component that is separate to the two wall components - this might not be necessary, but follows the earlier precedent. One might want to attach further information to the *junction,* independently of the walls. The description of the *junction* component says that it has a *drawing* part which has a *segment* part which is described by two *construction points* (the same as the *join point* and *opposite point* described earlier) and its line *style* is dotted.

The *change* part propagates changes to the descriptions of the adjoining walls, to complete the junction. The *change* part has a *joining end* and a *fixed end.* The *joining end* part is filled by the same kind that fills any *end* of the *joining wall,* that includes within its description any construction point that is the same as the *origin* part of the *joining wall.* The *change* part's *joining end's* inherited description then receives a *move point* part which is filled by the same kind that fills any *construction point* of the *joining end* part, provided that the *construction point* is not the same as the *origin* part of the *joining end.* The *move point* part then receives a *move to* part which is filled by the same kind as fills the *move point* part, and the description of that kind gets replaced by the description of the *opposite point* part (declared earlier) using the <= change operator. The *change* part's *fixed end* part gets treated in a similar fashion.

These paragraphs have given a conversational interpretation of the composite MOLE expression describing end-on junctions, using far more words. All these conversational words are needed to express this knowledge about junctions, if we want to do so conversationally. If we then want a machine to represent this knowledge, we have to re-express it in terms of system-defined relationships. Notice that the words included in MOLE expressions can refer to anything that a user might have in mind, to objects, events or tasks. None of the words included in these expressions have any meaning to the system, other than to signify different instances of kinds and slots. Yet, by using system-defined relationships to link words, the system can be made to exhibit behaviour as though it knows what it is doing, by effecting changes to drawings of walls in order to form junctions.

MOLE does things by evaluating descriptions, by using a left-to-right, depth-first search procedure (working on virtual tree structures implicit in composite expressions, as illustrated in figure 6). As indicated at the beginning of this explanation, this description of junctions includes references to parts that are not yet identified within the description. Something more has to happen before the system can produce instances of junctions.

INSTANCES OF JUNCTIONS

```
JUNC_END_AB:
    [join_wall +> WALL_A::dwg,
     fix_wall +> WALL_B::dwg,
     join <+ JUNC_END].
```

If we want to join two particular walls, say walls A and B, then we have to declare a kind name for this instance, any name: JUNC_END_AB. We can then say that an instance of junction has three parts. These are a *joining wall* part, a *fixed wall* part and a *join* part. The kind that fills the *joining wall* part is declared to be the same as the kind that fills the *drawing* part of wall A. Similarly, the *fixed wall* part is filled by the kind that fills the *drawing* part of wall B. The *join* part then is filled by an instance of the JUNC_END kind, inheriting all the description of that kind. The *fixed wall* and *joining wall* parts referred to in the earlier general description of junctions can now be found as kinds included in the description of this instance of junction.

With this description of an instance of junction, the system is able to evaluate the description and effect changes to the drawings of walls, to produce the completed junction. This approach to describing junctions can then apply to all cases of end-on junctions at any angle and for walls of unequal thickness (with the exception of a straight end-on junction between walls of different thickness) as shown in figure 6 below.

What has been shown by this example is the use of MOLE to model a perception of parts of buildings (not necessarily a correct perception). This has been done by using words that map onto those parts, plus words that map onto drawings of those parts, and by using system-defined symbols to declare relationships between parts and effect changes to descriptions. The example of wall junctions has been chosen because, in previous experience of CAD, it presented severe problems. Previously, we were

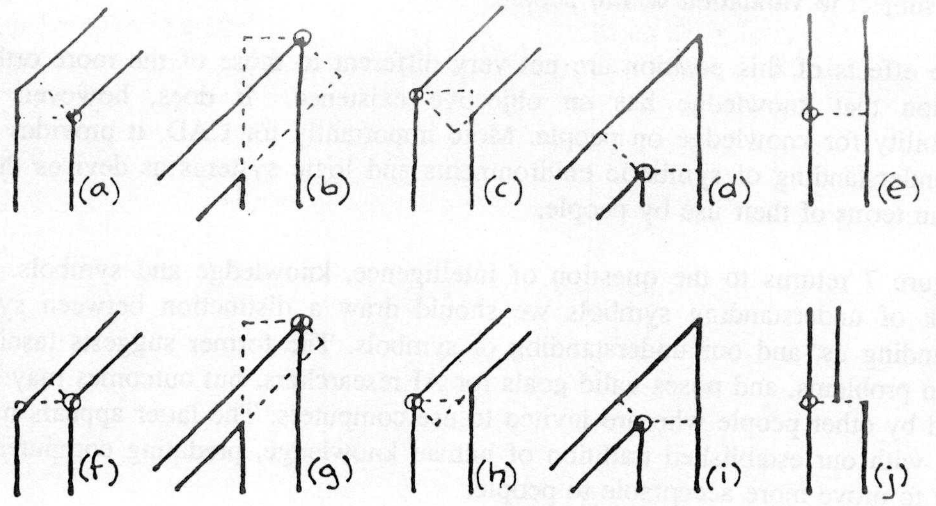

Figure 6: JUNCTION CONFIGURATIONS
Configurations for any angle of end-on junction between pairs of walls of equal and unequal thickness.

constrained to orthogonal arrangements of walls, in plans of buildings. This was especially true in those cases where a system was expected to interpret drawings in order to evaluate the effects of junctions on construction materials, and on other aspects of designs for buildings. Now this case of junctions is no longer particularly difficult. The point of the example is that it illustrates a use of a system which does not rely on domain knowledge being held in some prior and separate way within the system.

CONCLUSIONS

MOLE can be regarded as being on a path towards computer literacy, a development of our familiar written language. It offers logical constructs for composing expressions without requiring prior domain knowledge - no prior definitions for part/whole distinctions, discreteness of parts, typing of parts and correctness of results. However, this flexibility is gained at the expense of users having to know and work within the logic of the system. In effect, users have to construct their own definitions of objects and tasks which they see as being relevant to their own applications. The ability to do so is what is meant by computer literacy.

More generally, by differentiating between human knowledge, symbolic expressions that emanate from such knowledge, and logic systems that support manipulations of expressions, we can then conceive that human knowledge exists only within human beings - not in expressions, nor in logic systems. This position admits that knowledge can be represented symbolically, but that a representation is not the same as knowledge within people. A representation can be validated only by people. A representation may come to be accepted by many people, thus acquiring the status of conventional knowledge, but even then it is not the same as human knowledge. It remains subject to validation within people.

The effects of this position are not very different to those of the more orthodox assumption that knowledge has an objective existence. It does, however, focus responsibility for knowledge on people. More importantly for CAD, it provides scope for an understanding of symbolic environments and logic systems as devices that are defined in terms of their use by people.

Figure 7 returns to the question of intelligence, knowledge and symbols. When we think of understanding symbols we should draw a distinction between symbols understanding us, and our understanding of symbols. The former suggests fascinating and deep problems, and poses valid goals for AI researchers, but outcomes may not be accepted by other people who are invited to use computers. The latter appears more in keeping with our established tradition of human knowledge, predating computers, and is likely to prove more acceptable to people.

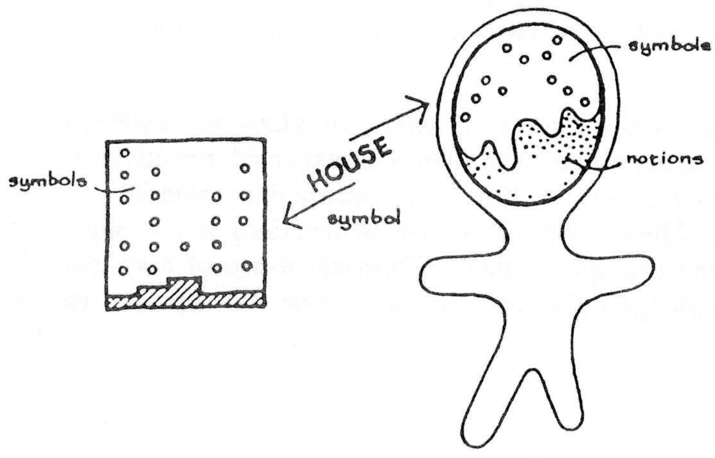

Figure 9: UNDERSTANDING SYMBOLS
We employ human notions to understand symbols, but should we expect symbols to understand us? Perhaps the ambition of AI needs to be rephrased as "intelligent use of dumb systems."

Acknowledgments

Research at EdCAAD, which forms the background to this paper, is being supported by the UK Science and Engineering Research Council and the European ESPRIT Programme, project P393 (ACORD).

References

1. Bijl, A., Stone, D. and Rosenthal, D.S.H.; *Integrated CAAD Systems,* EdCAAD Report for the Department of the Environment, Edinburgh University, UK, 1979.

2. Bijl, A.; *An Approach to Design Theory,* Proc. IFIP WG 5.2 Working Conference on Design Theory for CAD, Tokyo, Japan, 1985.

3. Dreyfus, H.L.; *What Computers Can't Do - The Limits of Artificial Intelligence,* (revised edition) Harper Colophon Books, NY, 1979.

4. Searle, J.; *Minds, Brains and Science,* 1984 Reith Lectures, BBC Publication, UK, 1984.

5. Krishnamurti, R.; *Representing Design Knowledge,* submitted to Environment and Planning B, Planning & Design, UK, 1987.

6. Bijl, A., *Architecture in Mind, Computer Discipline and Design Practice,* in preparation, Wiley, UK, 1988.

7. Minsky, M.; *A Framework for Representing Knowledge,* in *Psychology of Computer Vision,* 211-277, Winston P.H. (ed) McGraw-Hill, USA, 1975.

8. Woods, A.W.; *What's in a Link: Foundations for Semantic Networks,* in *Representations and Understanding,* 35-82, Bobrow, D.G. and Collins, A.M. (ed) Studies in Cognitive Science, Academic Press, NY, 1975.

9. Szalapaj, P.J. and Bijl, A.; *Knowing Where to Draw the Line* (IFIP WG 5.2, Hungary, 1984) in *Knowledge Engineering in CAD,* 149-169, Gero J.S. (ed) North Holland, 1984.

10. Bijl, A.; *Computer Aided Housing and Site Layout Design,* proc. PARC79 (Planning Architecture & Computer): Int. Conf. on Application of Computers in Architecture, Building Design and Urban Planning, 283-292, Berlin, West Germany, 1979.