Engineering with Computers 7, 209–224 (1991)

Engineering with Computers



# Toward the Integration of Knowledge for Engineering Modeling and Computation

**Clive L. Dym**
Department of Civil Engineering, University of Massachusetts, Amherst, MA, USA

**Raymond E. Levitt**
Department of Civil Engineering, Stanford University, Stanford, CA, USA

**Abstract.** Recent developments in computer science, especially in the area of artificial intelligence (AI), have made possible the representation of knowledge in symbolic terms. This, in turn, has made it possible to represent and integrate a broader range of engineering knowledge so as to provide new kinds of computer support for both analysis and design. This paper presents a typology of engineering knowledge to provide a conceptual basis for its computational integration. A review of the roles of numerical, geometric, and symbolic representations is also given within the context of the knowledge typology. Illustrations are provided from the domain of structural engineering.

## 1 Introduction

This paper explores the ways in which recent advances in computational paradigms and representation affect the ways in which engineering is done. The central thesis is that new styles of programming based on symbolic representation allow us to represent much more and significantly different engineering knowledge than ever before, especially descriptive, heuristic, and causal knowledge that heretofore could not be readily—if at all—expressed numerically or in formulas [1]. The integration of numerical and symbolic programming styles, together with graphics, provides unparalleled opportunities for engineering modeling, analysis, and design. This paper outlines in conceptual terms the specifications for such integrated computing environments. In so doing, we examine what engineers actually do, how they do it, how computers are currently used in engineering, and what kind of knowledge they bring to bear in their problem-solving activities. We also describe some recent approaches to the integration of representations of engineering knowledge.

*Offprint requests:* Clive L. Dym, Department of Civil Engineering, University of Massachusetts, Amherst, MA 01003, USA.

### *1.1 The Tasks of Engineering*

In order to describe what might be required for an integrated approach to engineering knowledge representation, we need to answer some very basic questions: (1) What do engineers do? (2) How do engineers do what they do? (3) How do engineers use computers? and (4) What kind of knowledge do engineers bring to bear on their work? One reasonably broad response is that engineers design, plan, build, and operate artifacts. In so doing, they also communicate a variety of types of information about the form, functions, and behavior(s) of these artifacts. This view is consistent with dictionary definitions and it uses comfortable and familiar terminology, words like *design* and *build*. Engineers, at least, know what they mean, even if they are not quite precise enough. For example, the verb *design* means, among other things, to "draw the plans for" and to "create, fashion, execute, or construct according to plan." However, we need better specifications of these engineering tasks in order to think about replicating them in any sort of computer program.

The spectrum of problem-solving tasks outlined by Amarel [2] and depicted in Fig. 1 provides a framework within which we can identify the engineering tasks of specifying, designing, analyzing, redesigning, and making artifacts. In this model there are two broad classes of problem-solving activity. The first is the set of **derivation** or **classification tasks,** in which solutions are *derived* from given facts and data. This class includes diagnosis, in which we seek to identify the reasons for a failure or malfunction; interpretation, in which we try to discern a pattern among a set of facts or readings so that we can understand their meaning; and monitoring, in which we track the changes in a system dynamically with a view toward intervening if something critical is about to happen. Derivation tasks in engineering

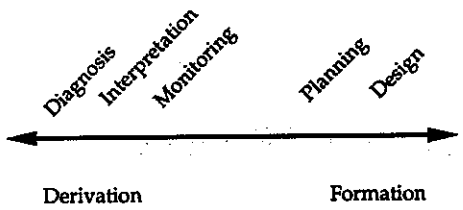


**Fig. 1.** Amarel's spectrum of problem-solving tasks.

would include debugging systems, selecting components in artifact design, and choosing among alternative analysis or control strategies. The following are typical engineering classification tasks [3, 4].

- *Diagnosis and repair.* Identify the causes that might explain the observed symptoms in a failed structural system and generate a plan of interventions to restore the system's failed functions.
- *Selection.* Find an analysis strategy and select the corresponding tool(s) to analyze a structure, given certain constraints on time, cost, and the degree of accuracy required of the answer.
- *Interpretation.* Locate the source and determine the magnitude of an earthquake from ground motion data recorded at several remote seismometers.
- *Monitoring and control.* Decide on the timing and nature of specific actions that must be taken during the processing of polymers in a chemical plant.

The second set of problem-solving tasks are the **formation** or **synthesis tasks,** in which a problem is solved by *forming* or creating an object or a set of plans for making an object. This class includes both planning, in which we seek to lay out the sequence of steps required to achieve some objective, and design, in which we seek to define an artifact in terms of a set of known features or components. Some engineering examples of synthesis tasks are [3]

- *Planning.* Plan the delivery of construction equipment of the appropriate type and capacity to arrive at a building site so as to ensure that there will be no construction delays.
- *Design.* Lay out a path that paper can travel through a copying machine and design all requisite devices needed to move the paper physically along the path in terms of their type, dimensions, materials, and locations.

The examples given here are relevant not only because they are legitimate engineering tasks, but also because each has been encapsulated in a knowledge-based system [1], a central part of the integrated approaches to modeling engineering knowledge discussed herein.

Another way of distinguishing between derivation (classification) tasks and formation (synthesis) tasks is the following [3, 5]. In derivation tasks we are intent on classifying data (which is presumably available to us). However, in general, we must be prepared to handle both unreliable data and time-varying data in order to be able to do diagnosis, monitoring, and so on. When synthesizing something, on the other hand, the task requirements may involve evaluating partial solutions (e.g., incomplete designs); dealing with interacting—or even conflicting—subgoals; and perhaps guessing at or generating multiple candidate solutions, especially where a complete specification of the form of the solution may not exist. The point is that the requirements vary with each type of task in Amarel's spectrum; thus, we expect that the solution prescriptions will vary as well. Far more is known about representing derivation tasks than about representing planning and design, a reflection of the state of engineering practice today, where we can formalize much more of what we know about diagnosis and selection than we can of what we know about design synthesis [1]. Not also that traditional engineering analysis based on mathematical modeling does not readily fit into either category. Mathematical models are useful in engineering analysis and design, and they provide information for both derivation and formation tasks. In the discussion that follows we will provide examples from structural engineering and pointers to some of the related, more general literature on engineering analysis and design.

### *1.2 How Computers Are Used in Engineering*

How are computers used in engineering? The standard answer would no doubt reflect advances in numerical codes and algorithms, graphics, and the manipulation of large spreadsheets and data bases. It can be argued that advances in computer science go hand in hand with advances in engineering; applications often motivate and drive progress in computational technique. For example, the advent of large-scale finite element method (FEM) codes has its roots in applied mathematics and numerical analysis. Their conceptual development proceeded along parallel paths in structural analysis, algorithm development, and hardware advances. We show in

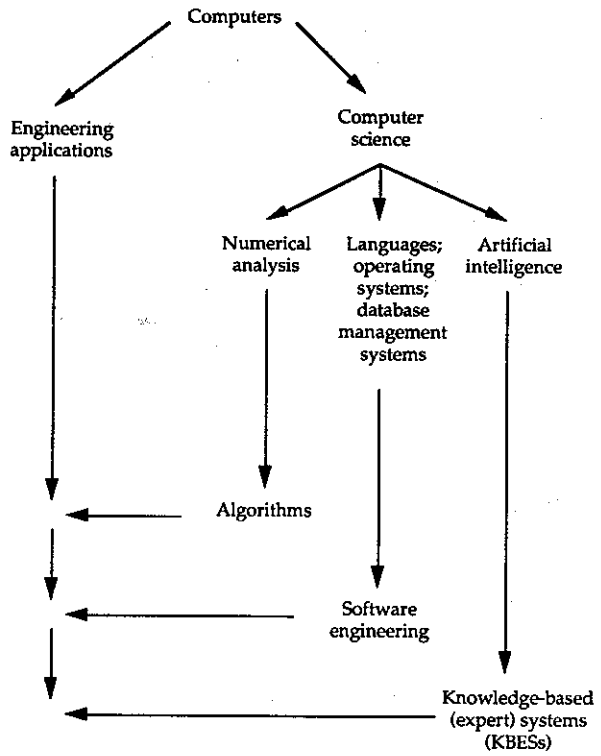


**Fig. 2.** A pictorial view of the relation between developments in computer science and engineering applications (Adapted from [6].).

Fig. 2 a pictorial view of these interactive developments [6].

Computers are now routinely involved in every phase of engineering. The kinds of computers used range from desktop personal computers to mainframe supercomputers. The kinds of tasks range from elementary modeling for analysis and design to the preparation of documents and drawings, to highly developed supercomputer and parallel algorithms used for numerical modeling such things as weather patterns and complex fluid flow problems involving highly nonliner partial differential equations. In all these applications, notwithstanding the use of word processing and graphics, the principal representation for computing is numerical.

From a broader perspective, computers have allowed us—as engineers—to assimilate and exploit much more knowledge than we could just a few years ago, and at a much faster rate. Thus, we can model more complex phenomena in a more satisfying way and, as a result, can design more complex and sophisticated artifacts. In addition, we have an increasing capability to teach the computer knowledge that is representable in general symbolic terms so that we can introduce qualitative aspects of engineering that cannot be captured as numbers or algorithms. That is, we can program deductive reasoning, rules of thumb, judgment calls, and so on. And, in fact, the thrust of this paper is that we now have the ability to do better engineering because we can encode much more knowledge and in much more accessible and useful ways. Thus, we can go well beyond the algorithmic black boxes that are in widespread and daily use. We can now think in terms of *intelligent assistants*, which incorporate useful forms of human expertise that can be applied in different ways and that can produce advice about and explanations for the problems they are called upon to solve. Thus, if anything, computers will become even more central to our ability to perform engineering tasks.

This is especially true for design, which we could view as an iterative recycling of synthesis, analysis, and evaluation. Computing has been effective for automating analysis (namely, analysis algorithms and "black boxes") and for documenting synthesis (e.g., keeping track of syntheses as they unfold). But there has been virtually no role for traditional numerical, procedural computing in design beyond those outlined for analysis and documentation. However, the advent of symbolic representation and declarative programming has opened up the possibility of automating routine design [7] and of providing design assistance for variant design (e.g., see Refs. [8–10]. Recent research into applications of knowledge-based (expert) systems (KBESs) in engineering is finding its way into practice [1, 11]. Thus, as noted earlier and depicted in Fig. 2, there is a synergistic interaction between developments in AI-based programming styles and rigorous engineering applications.

### *1.3 How Computers Might Be Used in Engineering*

We now briefly consider some of the abstractions of knowledge used in computer-aided approaches to structural engineering. It will become apparent through this discussion that substantial leverage can be gained if these separate abstractions were to be integrated into a single "structural engineering modeler's assistant." Many pieces of such an integrated environment are in place, in the sense that working prototypes and demonstrations have been developed for many aspects of structural analysis and design. In this discussion we focus on the knowledge-based approaches and "ignore" the numerical approaches except as they are procedural tools that could—and should—be embedded in an

integrated structural engineering environment. Further, the work cited is not intended to be complete; rather, it is suggestive of the kinds of engineering knowledge that can be made computationally representable and tractable—and thus amenable to inclusion in an integrated structural engineering environment.

Starting with structural analysis, three abstractions are relevant. One is concerned with the representation of structural system behavior at three distinct levels: spatial layout, functional, and behavioral (e.g., load-deflection relations) [12]. Of particular interest is the ability to add and retract assumptions so that basic structural modeling assumptions can be kept in mind through an entire analysis cycle. A second is the extraction of qualitative causal behavior from a description of the structural system in the domain of seismic analysis [13]. The third abstraction concerns the representation of meta- or strategic knowledge in the structural modeling process [14]. Here the focus is on the choice of modeling tool in the context of issues of time, purpose, granularity, cost, and so on. These abstractions have as their focuses the idea of capturing qualitative knowledge about structural behavior, at several levels, to provide powerful analysis support tools. Some of this incorporated knowledge is about fundamental physical and geometric principles, some describes structural phenomena, some pertains to the intentions behind the use of the knowledge, and all seek to link these kinds of knowledge to some mathematical or numerical analysis tools.

In the design of structural systems several useful projects are worth identifying as candidates for integration. The HI-RISE knowledge-based system focused on conceptual and preliminary design for a class of high-rise structures [9]. More recently, a more extensive conceptual design system has been developed to incorporate explicitly functional and behavioral intent, in qualitative terms, as well as conformance with more abstract objectives such as cost [15, 16]. This work also includes the satisfaction of various codes and standards as design constraints. And, at this end of the design process, work on standards processing is worth noting [17, 18]; in this context, the representation of design standards in a computational (object-oriented) form becomes a powerful design aid, facilitating continuous, real-time standard checking as the design process unfolds. These approaches incorporate more abstracted knowledge of structural behavior and some description of intent with experiential knowledge about practice and performance codes. Thus, the kinds of knowledge invoked here depend on but are significantly different from the kinds of analysis knowledge described in the previous paragraph.

The point, again, is simple—although its implementation is not. The kinds of knowledge that are now being incorporated into various computer-based tools is both far deeper and far broader than what has been available through traditional numerical tools. These kinds of knowledge include the fundamentals of structural mechanics, specific modeling knowledge, different abstractions of behavior, knowledge of experience from practice and specifications, and meta-knowledge about how and where to invoke the other kinds of knowledge. Thus, there exists the opportunity to identify and integrate the different kinds of knowledge we bring to bear in doing engineering, to identify the appropriate representations for these knowledge types, and to integrate them into a powerful computational engineering modeling environment.

## 2 Representations of Engineering Knowledge

Section 1.1 has provided a vocabulary for talking about the process or tasks of engineering and section 1.2 has outlined the traditional uses of computers in engineering while hinting at the potential for new approaches to engineering computation. However, we have yet to express a clear idea of the knowledge we use in solving engineering problems or of the language(s) in which that knowledge is cast. Therefore, the next step in our discussion must be to focus on *modeling,* on how we actually represent or describe something so that we can describe a phenomenon or object, make predictions about it or with it, analyze it, design it, and so on. The idea of modeling is inherent in the scientific method and is a central feature of engineering [19, 20]. We use models all the time, generally in an unconscious fashion in which we take them for granted. We use verbal models to describe thoughts and feelings as well as objects and plans. And in engineering we use a technical vocabulary and various kinds of mathematical and numerical models to do our work. For example, we model structures, circuits, chemical processes, and mechanical parts, and for all of these we have a set of terms, both verbal and mathematical, that we use to describe the immediate objects of our attention. However, the main point is that our most familiar models are mathematical ones in which we describe objects in the language of continuous mathematics, such as differential and integral calculus. These models serve us quite well in many ways, and they have

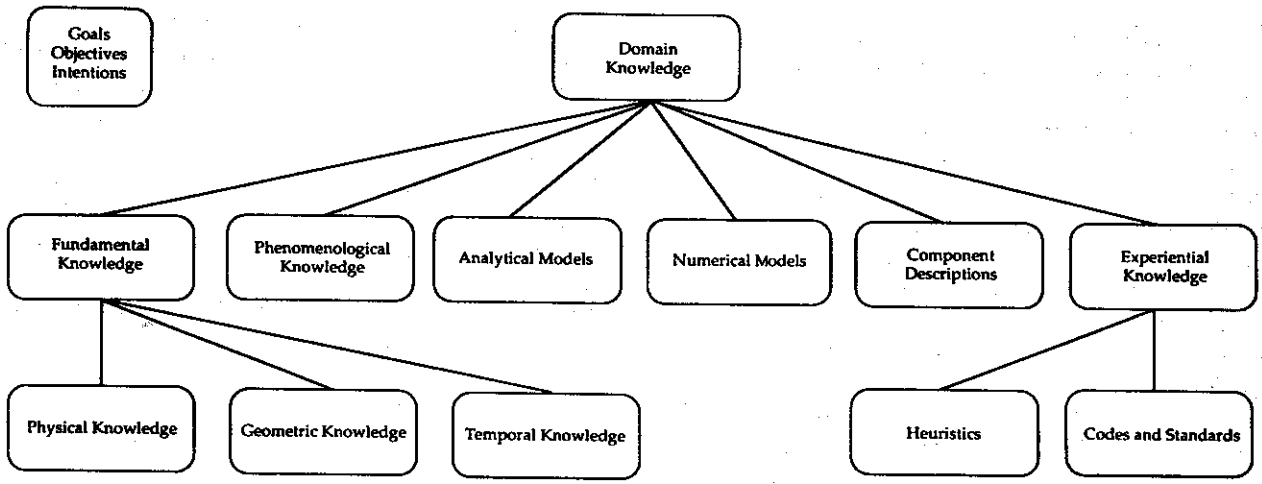


**Fig. 3.** A typology of engineering knowledge.

also been extended into algorithms that can be effectively manipulated on computers by processing symbols that represent numbers.

### 2.1 *A Typology of Engineering Knowledge*

In this paper we endeavor to recognize and incorporate new ways to describe and solve engineering problems in terms of "languages" for representing the problems and their constituent pieces and methods. We do this because there is much more to engineering than can be described in (continuous) mathematical and numerical terms. The typology of engineering knowledge outlined below (and exhibited in Fig. 3) identifies the different kinds of knowledge brought to bear on engineering tasks. The various knowledge types are illustrated with examples drawn from the field of structural analysis and design. The types of engineering knowledge include:

***Engineering Domain Knowledge.*** This category incorporates all the domain knowledge that can be applied to an engineering problem, regardless of its origin—theoretical, experimental, or experiential—and independent of its form of expression or its intended use.

- *Fundamental knowledge.* This class subsumes the most basic and fundamental knowledge of physics, geometry, and time. In a definition that departs from the customary usage, this is **first principles** knowledge because it is about the relevant fundamental physical laws *and* the spatial and temporal references for their expression.
  —*Physical principles.* These are the basic physical laws on which stands the domain in which a particular engineering problem is being solved. In applications to structural engineering the discipline of engineering mechanics would embody the first principles in statements of Newton's laws of motion, conservation of energy, and so on. Note that these principles, while typically expressed and applied in mathematical or formulaic terms, are also expressions of concepts that we may wish to reason about in qualitative or symbolic terms.
  —*Geometry.* Geometrical knowledge includes descriptions of the shapes and forms of objects and the spatial relation of system components and other objects to each other. In traditional practice, geometrical knowledge was typically displayed in drawings and blueprints. Then technology evolved to include, first, computer-aided drafting (CAD) and, now more intensive and interactive geometric modeling tools.
  —*Time.* This knowledge is about the order of events, precedence, and sequences of actions.
- *Phenomenological models.* These are models often based on experimental result or on compiled high-level extrapolations of the fundamental laws. Examples of such models include pressure-volume relations, stress-strain laws, and generalized force-deflection and mass-stiffness-frequency relations. These phenomenological models can be expressed in analytical terms, either as formulas or in some numerical model, but they are often also amenable to representation in qualitative terms. Symbolic formulations of such phenomenological models permit—and even encourage—causal reasoning.
- *Analytical models.* These are formulas, both exact and approximate, used to model specific "cases" or subsets of the more general first prin-

ciples and phenomenological models. These models typically employ continuous mathematics as their language for stating and manipulating the models. For example, in modeling elastic continua we use variational principles to derive consistent, case-specific models for beams, columns, plates, and shells [21]. The case-specific models could be very general in terms of the kinds of behavior they admit; for example, they might incorporate time-dependent (dynamic) effects, stability and buckling, and nonlinear constitutive behavior.

- *Numerical models.* These are the mathematical formulations of the computational versions of the analytical formulas just described. Perhaps the best-known examples in structural mechanics are matrix representations that underlie the finite element codes that are in widespread use in structural and mechanical engineering [21].
- *Component descriptions.* The design of a complex system often involves selecting existing components and configuring them to meet certain objectives, as in the design of computer systems [22]. The identification and representation of the attributes of these components is important engineering knowledge, as is the relationship of individual components to assemblies of components. Thus, component descriptions necessarily include hierarchies of relationships of parts to a whole.
- *Experiential knowledge.* In this category are listed the kinds of knowledge that derive from our experience. This knowledge can take the (familiar) form of heuristics, but it also includes the codes and standards that serve as design objectives for many engineering artifacts. Structural codes, for example, derive from our experience about what makes a safe or serviceable structure.
  —*Heuristics.* Heuristics are typically written as rules because they represent those "rules of thumb" that arise out of experience. However, it is important to remember that a rule is a representation device that can be used to represent many different kinds of knowledge. In one dichotomy, rules may be used to express truly heuristic *surface knowledge* as well as *deep knowledge* that reflects a compilation of first principles [23]. For example, in the SACON system that was designed to act as a preprocessor for application of the MARC finite element package [24], rules are used to express some aspects of first principles, compiled versions of both phenomenological and analytical models, and heuristic preferences and assumptions about the use of numerical codes. In the MUMS system, rules are used to express *strategic* knowledge about using any or all available tools in searching for an appropriate way to solve a given problem, subject to constraints about granularity, time requirements, cost, and so on [14]. Heuristic knowledge—as defined here—can incorporate both strategic knowledge and more specific experiential knowledge.
  —*Codes, standards, and constraints.* A complete engineering design is expected, based on our experience, to meet certain specifications or conform to various of codes of behavior. For example, the design of a steel-framed building would be expected to conform to (among others) the AISC Specification for its structural components [17, 25] and the NFPA Life Safety Code for its floorplan [26, 27]. In this context, no distinction is drawn between specifications and constraints, although one can choose to identify specifications with attributes that are to be optimized in a design and constraints with attributes for which satisficing is adequate [1].

***Goals, Objectives and Intentions.*** This is meta- or process knowledge, that is, it is concerned with the reasons why the problem is being solved. This knowledge is in some sense analogous to the definition of the "outer environment" of a design as the surroundings within which a designed artifact will operate [28]. It often includes knowledge outside the specific problem domain and represents the intentions of the engineer as much as the artifact being engineered.

This typology of engineering knowledge is not a formal taxonomy, and the boundaries between the categories are, in some instances, rather fuzzy. For example, the statement that "a post-and-frame is regarded as unstable" could be viewed as first principles knowledge [15, 16], but it could also be viewed as a phenomenological description because it refers to a phenomenon observed through an application of first principles (i.e., a stability analysis) to a specific structural configuration. In fact, in one ordering of structural design knowledge, it appeared to be useful to categorize qualitative knowledge of behavior in one category and quantitative knowledge of behavior in another category [15], that is, the categories were defined—at least to this extent—by the representation rather than by the underlying knowledge. Similarly, the description of artifact function in a design problem could be reflected in the meta-knowledge as well as in the component descriptions.

Indeed, each of the different kinds of knowledge in

the typology could be—and often is—expressed in different forms, depending on the problem at hand. Momentum considerations for particles (and structures), for example, can be written in several forms, depending on the type of information sought and the uses to which that knowledge will be put. Momentum, however, cannot be measured directly but is calculated from velocity measurements; yet the most common form of motion detector (the accelerometer) measures acceleration. On the other hand, if we were modeling the dynamic behavior of a structure in order to deal with changes in structural form, information about the relative motion of components will be needed. Such *multiple perspectives* are important in engineering problems and so representation issues will arise across the entire typology (cf. section 2.3).

### 2.2 *Other Characterizations of Engineering Knowledge*

There are other ways to order or characterize the knowledge, some of which have been referred to earlier. For example, in a discussion of a methodology for conceptual structural design seven categories of knowledge are identified [15]:

1. *Structural elements and systems knowledge* contains information about the attributes (e.g., function, qualitative behavior, etc.) of generic structural elements.
2. *Behavior and performance knowledge* includes mathematical models of structural behavior and formulas derived from performance criteria.
3. *Product knowledge* about specific products.
4. *Concepts* or abstractions such as cost, floorplan layouts, and so on.
5. *Strategy knowledge* about problem-solving approaches.
6. *Reasoning with constraints* about the knowledge contained in the previous categories.
7. *Evaluation knowledge* to assess and rank candidate conceptual designs.

Although there is overlap between this ordering and the typology presented here, there are some very clear differences. The most obvious one is that geometry does not appear in the preceding categorization, but perhaps more important is the fact that these categories are based in large part on the representations used in each rather than on intrinsic definitions. For example, behavioral knowledge appears qualitatively (as a slot in the frame representation of a structural member) in category 1 and quantitatively (as an appropriate formula) in category 2. Similarly, the application of constraints in category 6 is a restatement of the knowledge contained in categories 1 to 5 in a form amenable to constraint propagation and satisfaction [15]. For the present typology, as we discuss later in greater detail (cf. section 2.3), different representations may be useful for the knowledge contained in any single category.

The inclusion of "strategy knowledge" as a separate category is similar to the distinction drawn between strategic knowledge and substantive knowledge as applied to medical diagnosis [29] and structural modeling [14]. In these applications substantive knowledge was characterized as "the rules of the game" and strategic knowledge as "knowing how to play the game." In the structural modeling application the strategic knowledge was obtained from knowledge acquisition sessions with the domain expert, whereas the substantive knowledge was typically derived from texts, papers, or other reference materials. Both kinds of knowledge were represented as rules (and frames), although (in the typology of section 2.1) the strategic rules would be categorized as heuristics and the substantive rules as either first principles or phenomenological knowledge. Again, inasmuch as it seems useful to separate the knowledge from its representation, in a kind of "declarative knowledge paradigm," the approach taken in [14] seems more consistent with the present typology.

It is also interesting to look at the typology from the dichotomy of analysis knowledge versus design knowledge. Analysis work would generally require the following kinds of domain knowledge: fundamental, phenomenological, analytical, numerical, and heuristic knowledge. For design, the complete typology of knowledge would have to be invoked because component descriptions and codes and standards are inextricably involved in the design process. Needless to say, both analysis and design work require the application of relevant process knowledge.

### 2.3 *Representational Implications of the Engineering Knowledge Typology*

One reason for outlining such a typology is to emphasize that while mathematical and numerical representations are useful—if not absolutely essential—to the intelligent application of some kinds of engineering knowledge, they are inadequate for other kinds of knowledge. For example, choices made by an expert about *how* to solve a particular structures problem, choosing among analytical

methods, numerical schemes, back-of-the-envelope calculations, experiments, and so on, are choices (with corresponding attributes) that cannot be described in formulas or with a numerical code [14]. And while we might have a finite element model of a part we are designing, we typically cannot represent the function of that part within the product to which it belongs in such an algorithm [12], and there are other attributes (e.g., color, fit, manufacturability) that also cannot be easily described or about which we cannot reason if they are represented with numerical symbols. Further, much of the knowledge that we categorize as heuristic, whether it is shallow or deep, is simply not expressible in continuous mathematical formulas, although they can be expressed in the "discrete" mathematics of formal logic (see [1], chap. 3). And we have suggested that even first principles knowledge can often be usefully written in some form of rule. Therefore, we must look to other ways to represent the different kinds of knowledge that engineers use to solve problems.

Thus, we need some additional language or representation tools beyond those of continuous mathematics and their numerical realizations. First, for representing problems and methods available to solve problems, we can use the ideas of search and of logical inference as ways of formulating problems (see [1], chap. 2–4). These techniques for formulating problems and solution approaches move use away from the idea of seeing every problem as the solution to a differential equation, whether analytical or numerical, and toward representing problem states as discrete elements in a space that must be searched in order to find a solution. *Search* gives us a new way of formulating problems so that we can look at problems in a way that classical methods do not permit. Similarly, the exercise of *logical inference* allows us to apply formal techniques of deduction to derive new facts from those currently believed to be true, or to test the consistency of a proposed new fact with respect to an existing base of facts.

Second, we use *symbolic representations* of the detailed engineering domain knowledge that we use in search and logical inference. These relatively new representation ideas include *rules* to describe situations and consequent actions, *semantic nets* to order elementary objects into a meaningful organizational network, and *frames* or *structured objects* to characterize more complex objects and concepts and their attributes and relationships (see [1], chaps. 5–7).

Third, as we apply different representation schemes to encode fully all varieties of engineering knowledge, several problem-solving paradigms will be used. The most familiar is the *procedural paradigm* used in conventional (numerical) analysis programs in which the problem-solving knowledge is "hard-wired" into the procedures written to solve a particular problem. Symbolic representation techniques are applied in *declarative paradigms* in which knowledge is "declared" independently of how it will be used [1]. This stands in direct contrast to the intertwining of knowledge and subroutines in procedural programming. An important instance of the declarative paradigm is *object-oriented programming* [30, 31], which facilitates the description of objects, their attributes and behaviors, through their membership in classes of objects. Another declarative paradigm is a simulation-like extension of *model-based reasoning* that allows us to integrate: symbolic representation, including aspects of rule-based programming and object-oriented programming; qualitative reasoning about the structure and function of systems and their components; and graphical descriptions and tools to tie these pieces together [32]. In this paradigm we can, in effect, simulate the behavior of a system in a symbolic—rather than numerical—sense [33].

Implemented in symbolic programming languages such as Lisp [34], symbolic representation, and declarative paradigms offer a host of advantages. For one thing, symbolic representation makes *abstraction* much more explicit as we look at descriptions of objects at various levels of detail (e.g., in an "inheritance lattice" [1]) and try to abstract at a given level of an hierarchy just that set of details or attributes or relationships or links that are needed to answer a given question. Further, symbolic representation techniques provide richer, more meaningful data structures than are available in data base records or tables. In model-based reasoning, for example, we formalize and communicate data and information about both the state and the behavior of components; we can reason about complex assemblies, their topologies, and their parts; and we can then describe components in terms of function and role, as well as in terms of conventional physical attributes. Thus, we have more powerful ways of modeling artifacts and of reasoning about them.

Rules can be used to represent not only associative heuristics and some compiled knowledge; they are also a representation tool that, when combined with other ideas such as qualitative reasoning, can transcend the limitations of heuristic reasoning, especially for first principles and phenomenological knowledge—which are generally applied through

formulas and computer codes. For example, while Newton's law of motion for a particle is often written as

$$F = m * \frac{d^2x}{dt^2}$$

it can also be written as

$$\text{Force} = \text{mass} * \text{acceleration}$$

This version seems more amenable to qualitative reasoning, for example, *causal reasoning,* in which we can relate cause and effect at a *conceptual level,* as in the statement that an unbalanced force produces a nonzero acceleration. That is, in addition to calculating force magnitudes, we can reason in *qualitative* terms about their presence and any resulting effects. Even in such a simple example it is clear that we can add an interpretive element to a structural programming environment so that in the absence of a zero sum of forces we can reason about whether there is some physical cause that effects a nonzero sum, in addition to the possibility that there may be arithmetic or other reasons for a nonzero right-hand side of the force balance equation.

The symbolic modeling of causal reasoning processes has been the province of *qualitative physics.* It originated with the desire to model qualitatively the behavior of devices and components, and an elaborate calculus was devised for this purpose [35–37], although to date this work has not been applied to serious engineering problems. It is easy to confuse reasoning within and about a discipline with causal reasoning. For example, a routine judgment in structural mechanics relates to the issue of when behavior is sufficiently nonlinear that linear models are no longer applicable. But to suggest that a large strain, depending on its relative size, might require consideration of a geometrically nonlinear model is *not* causal reasoning because it is a judgment about what "large" means rather than being a statement about a causative physical agent. At the same time, however, the statement that buckling should be considered in the presence of in-plane forces is an example of physical cause-and-effect. In both instances the representation would be symbolic and rule-like, but the meaning and applicability are rather different.

Note, too, that although the notion of qualitative reasoning has generally been tied to applying the qualitative physics calculus mentioned previously [37], such reasoning could be performed by executing rules whose arguments refer to physical quantities such as force, mass, and so on, rather than by applying that calculus [13]. Such rules, of course, would then be viewed as representing first principles knowledge and not as experiential heuristics.

Similarly, with regard to the analytical models (formulas) and numerical representations (and their computer implementations), we can do computations with them, but any associated reasoning must be done by the engineer who is applying them. That is, the calculations can be rendered computable, but not the associated interpretation and reasoning, which are largely the province of human intervention. Thus, if we were interested in automating the reasoning about these models for formulas, we would need some additional qualitative capability that does not now typically obtain. (An interesting question arises here [38]: Is it perhaps possible to utilize the representations of a symbolic manipulator such as MACSYMA [39] to do such associated qualitative reasoning about the physics of the objects being mathematically manipulated?).

Beyond *identifying* the various kinds of engineering knowledge that can be brought to bear in any given application, what is of most interest is recognizing that we gain the most problem-solving leverage by *integrating* these different kinds of knowledge, ideally within a single "engineer's assistant" workstation. Further, besides gaining enormous problem-solving power through such integration, we will also be better able to handle the entire modeling process, including being able to track the invocation of assumptions and restrictions. An integrated engineering environment can keep us from violating our assumptions and premises so we can ensure that our results are consistent with our starting assumptions.

Finally on representation, we note that the abstraction or typology of knowledge in an integrated engineering could be fixed or "hard-wired," or it could be locally modifiable. In a typical procedural program, which may have built-in numerical choice points or either-or-else tests, the user cannot easily go into the program and modify values in the choice points or tests, and we accept this as a standard part of using such numerical black boxes. Thus, although there are choices that might be regarded as universal (e.g., a shell is thin if its thickness-to-radius ratio is less than 0.10), others are more likely to be programmed in locally, such as the preferences for different modeling tools, which in turn might depend on locally available computing power, expertise, and so on. Therefore, tools such as we envision herein would be adopted by the engineering community only if they can be *customized* by the

user. Every design group has its own culture and its own approach to analysis-in-design. To the extent that local culture and lore are easily integrated, the adoption of this integrated approach will be hastened.

## 3 Current Work on Integrating Engineering Knowledge

We now describe some recent work that implements some of the ideas outlined in section 2. In particular, we outline first the representation integration issues that were identified and solved in the building of a knowledge-based system for architectural code checking [1, 26]. Then we outline in somewhat greater detail the model-based reasoning (MBR) paradigm [32, 33].

### *3.1 Integration of CADD and Symbolic Representations in the LSC Advisor*

The LSC Advisor is a prototype KBES designed to review an architectural design for high-level conformance with the Life Safety Code (LSC) of the National Fire Protection Association [27]. The LSC Advisor focuses on those aspects of the code that relate specifically to egress and fire protection features and equipment as they apply to new health care facilities [26]. Egress requirements specify the type and number of exits and maximum distances from points in the building to the nearest exit. Fire protection features include, for example, the specification of minimum fire resistance ratings for walls and doors, depending on their location and use. Requirements on fire protection equipment include rules for specifying the number and locations of fire extinguishers and exit signs. The egress and fire protection requirements were chosen for encapsulation in the prototype because they are central to the LSC and because violations in those areas are among the most expensive to remedy.

The LSC Advisor was designed to mimic the code-checking activities of an experienced architect as he or she reviewed the architectural drawings (floorplans) and specifications to confirm their conformance with the LSC. The architect must locate and check distances to exits, check the fire and smoke ratings of various partitions, and check the location of exits and their signage. The LSC review process can thus be thought of as having three distinct aspects: identification of building elements, selection of applicable requirements, and application of selected requirements.

For the most part, the identification problem has been left up to the user of the computer-aided design and drafting (CADD) system, which acts as the front end to the LSC Advisor. The CADD system user must explicitly identify the floorplan's different elements, that is, the rooms, doors, and walls. This is accomplished partly by using specialized drawing functions provided by the CADD system and partly by following certain conventions with regard to the use of symbols and the organization of the floorplan into layered drawings. The user must also select room labels from a special list so that occupancy can be properly determined. This allows automatic translation of the CADD data base into the frame-based floorplan representation that the LSC Advisor can analyze.

The process of selecting applicable requirements from the LSC is handled mainly by putting the necessary clauses into the antecedents of the rules that represent LSC requirements. It is also handled implicitly in that the knowledge representation facilitates the handling of objects as groups. This is done by representing all objects of the same types as instances of a class, for example, all rooms are instances of the class ROOMS, all doors are instances of the class DOORS, and so on. The application of requirements to particular building elements is accomplished by the insertion of appropriate clauses in rule antecedents.

Many aspects of the application of LSC requirements are implemented through algorithmic procedures, for example, calculating travel distances from rooms to exits. However, the determination of the required travel distance and the comparisons of the actual distances to those specific requirements are performed by application of rules representing code requirements. One interesting representational aspect of the LSC Advisor is its combination of complex geometrical calculations and algorithms with symbolic knowledge representation. The most interesting (and complex) of these algorithms are those that, in the precalculation phase, measure travel distances between points in the building and the building's exits. These algorithms are needed because the LSC sets maximum limits, referred to as corridor travel distances, on the travel distances from the doors of rooms to the nearest exit. Maximum limits are also set on the distance from the farthest point in a room to the nearest door leading into a corridor and on the sum of the in-room and corridor travel distances.

The LSC also specifies the corridor travel distances be measured along corridor centerlines so that tracing these centerlines becomes a significant

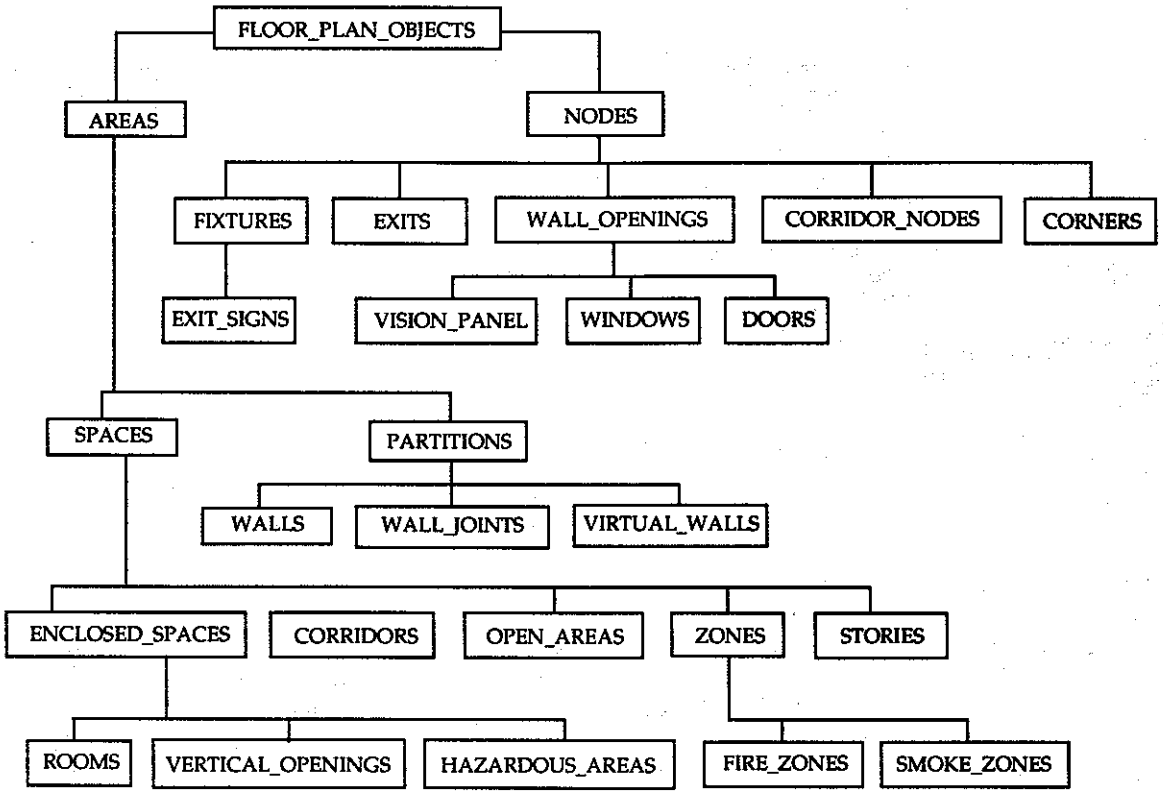


**Fig. 4.** Inheritance lattice of LSC Advisor classes (After [26]).

activity in the travel distance algorithm. This requires identifying and defining corridors that, for those of irregular shape, is nontrivial. The algorithm entailed the construction of a connected graph of points along the edge of the corridor that are visible from one another and then performing a carefully optimized depth-first search of the connected graph to identify the shortest path between each pair of will midpoints (see [26] for the details). Once all corridor centerlines have been formed, another connected graph is formed of nodes representing the building's exits, the doors for each room, and points along the centerlines of the corridors. A shortest-path algorithm determines the shortest distance from each room's doors to the nearest exit. In much the same way as was done for corridors, each room is then examined to see if any points in it are too far from its doors. All the results are stored in appropriate frame slots where they are available for evaluation by the various rules.

The floorplan representation developed for the LSC Advisor was intended to facilitate description of all the objects in a typical floorplan of blueprint; to enable reasoning about these objects in the context of analyzing and applying the LSC; and to provide an economic and efficient implementation for these descriptive and reasoning tasks. Inasmuch as the typical objects in a floorplan (e.g., fire zones, rooms, doors, walls, etc.) are fairly abstract, a rule-based approach in and of itself would not provide an adequate representation because, as we have stated repeatedly, rules alone are inadequate for describing objects with a large number of often complex attributes.

The LSC Advisor *inheritance lattice* (or *abstraction hierarchy*) is displayed in Fig. 4. The class frames are organized into a single hierarchy of subclasses of the class of FLOOR_PLAN_OBJECTS. Examples of subclasses of FLOOR_PLAN_OBJECTS are WALLS, ROOMS, CORRIDORS, and EXIT_SIGNS. Subclasses have all the slots of their superclasses, as well as additional slots that are applicable to it but not to other subclasses of the superclass. The slot LSC_PROBLEM is the only one that *all* the objects have in common; it is defined in the class FLOOR_PLAN_OBJECTS. NODES, one of the immediate subclasses of FLOOR_PLAN_OBJECTS, defines a COORDS slot for storing a single $(x, y)$ location. The subclasses of NODES include CORNERS and EXIT_SIGNS.

| Slot | Values |
|---|---|
| CORNER | ( #[Unit : CRNR99 BUILDINGS] #[Unit : CRNR104 BUILDINGS] #[Unit : CRNR94 BUILDINGS] #[Unit : CRNR32 BUILDINGS]) |
| EXTERIOR_NODES | ( #[Unit : CRNR97 BUILDINGS] #[Unit : CRNR102 BUILDINGS] #[Unit : CRNR92 BUILDINGS] #[Unit : CRNR37 BUILDINGS]) |
| GROSS_AREA | (2408.0) |
| OCCUPANT_LOAD | (6200.0) |
| SUBSPACES | (ROOM180 ROOM151 ROOM152) |
| ZONE_CORRIDORS | (CORRIDOR1 CORRIDOR2) |
| ZONE_DOORS | (DOOR79 DOOR86 DOOR89 . . . ) |
| ZONE_EXITS | (DOOR79 DOOR86) |
| ZONE_EXIT_SIGNS | (EXIT_SIGN1 EXIT_SIGN3 . . . ) |
| REQ'D_ENCLOSURE_RATING | 1.5 |
| NET_AREA | (1956.0) |
| LSC_PROBLEM | NIL |

**Fig. 5.** Frame representation of an instance of the class FIRE_ZONES (After [26]).

The superclass SPACES is used to organize building stories, fire zones, smoke zones, rooms, and corridors. All members of the class SPACES have the same general attributes: DOORS, NET_AREAs, GROSS_AREAs, BOUNDARY_WALLS, and a REQUIRED_ENCLOSURE_RATING. Rooms are then defined in this representation as having all the attributes of spaces in addition to those specific to rooms alone (e.g., INROOM_TRAVEL_DISTANCE, NUMBER_OF_REQUIRED_EXITS, etc.)

To illustrate how the floorplan representation all comes together, consider the application of a rule about the fire resistance rating of a door in a wall intended to serve as a fire barrier. Given that a floorplan may contain several thousand objects, we want its representation to facilitate rapid and efficient identification of the door(s) in question. How does this happen here? The process begins with the identification of all instances of fire zones that are direct descendants of the class FIRE_ZONES (Fig. 5). This could be a list of objects with between 2 and 5 entries. As the class FIRE_ZONES is itself a subclass of the class SPACES, it has a slot for the attribute BOUNDARY, which lists all the walls that make up the boundary of the fire zone. This list of walls, with anywhere from 4 to 30 entries, is used first to eliminate exterior walls by excluding those with the value EXTERIOR in their WALL_TYPE slot. Now the class WALLS (Fig. 6) contains the slot WALL_OPENING, which holds all the DOORS in a given wall, so that it is possible to identify all the specific doors to which the relevant rule applies. The RATING slot of each door so identified can now be examined and the relevant rule applied.

| Slot | Values |
|---|---|
| CORNER | ( #[Unit : CRNR113 BUILDINGS] #[Unit : CRNR112 BUILDINGS] #[Unit : CRNR139 BUILDINGS] #[Unit : CRNR138 BUILDINGS]) |
| WALL_TYPE | INTERIOR |
| RATING | 2.0 |
| LSC_PROBLEM | NIL |
| WALL_OPENING | (DOOR153) |

**Fig. 6** Frame representation of an instance of the class WALLS (After [26]).

One significant advantage of this hierarchical abstraction of the different types of objects in a floorplan is that general operations can be defined on a superclass. These operations will work then correctly on any object that is an instance of one of its subclasses. For example, a generic Lisp function that requires access to a COORDS slot can be defined for the superclass NODES; it will then work equally well, and without modification, for instances of CORNERS and EXIT_SIGNS. The attachment of a Lisp function to a slot of a superclass and its activation by message passing is an example of method inheritance in the tradition of object-oriented programming [31]. Additionally, many of the same economies can be derived when the operations are defined separately from the objects, either as Lisp source files or in the form of rules.

The floorplan information represented in the LSC Advisor goes well beyond the purely geometric data that is the province of the typical blueprint. Typically, detailed information on construction materials, wall finishes, and product names for prefabricated items must be obtained by reference to other documents, either detailed drawings or materials lists. Since information from all these sources is needed for detailed LSC review, a representation that could unify these diverse types of information had to be developed.

Unifying these diverse data with a single representation represents a significant step forward beyond most of the CADD systems used by architects today, in which there are no unified data bases for materials and floorplans. In the original version of the CADD system within which the LSC Advisor was built there were unique identifiers in the materials or finishes data bases and individual records for objects such as doors and rooms. Floorplans, however, were generated by use of a drawing program that maintained a separate data base of lines, arcs, and text strings. This graphical data base contained no explicit indication, for example, as to which lines were meant to be understood as repre-

senting the two sides of the same doorway. This information is (subconsciously) deduced by the architect looking at the complete drawing, based on general knowledge about how to interpret blueprints. Conversely, although materials and finishes data bases may contain some dimensional information, they typically contain no information about the positions of objects relative to one another. Therefore, as input to the LSC Advisor, a floorplan representation was specified in which geometrical information is explicitly related to high-level objects such as walls and doors, requiring that the architectural CADD system provide such a unified description of the building.

Several different kinds of knowledge are incorporated in the LSC Advisor, including experiential knowledge, geometric fundamentals, and component descriptions. The experiential knowledge is in part reflected in the architect's approach to code checking, that is, in the process itself, and in part in the very code that is being applied as a constraint on the design. The geometry is reflected in both the layout of the design and the algorithms developed to compute and minimize travel distances. Knowledge about component descriptions is incorporated in the basic representational hierarchy (cf. Fig. 4) and the related frame descriptions (cf Fig. 5 and 6) as they define the makeup and relationship of the floorplan components themselves and of more abstract components such as fire and smoke zones.

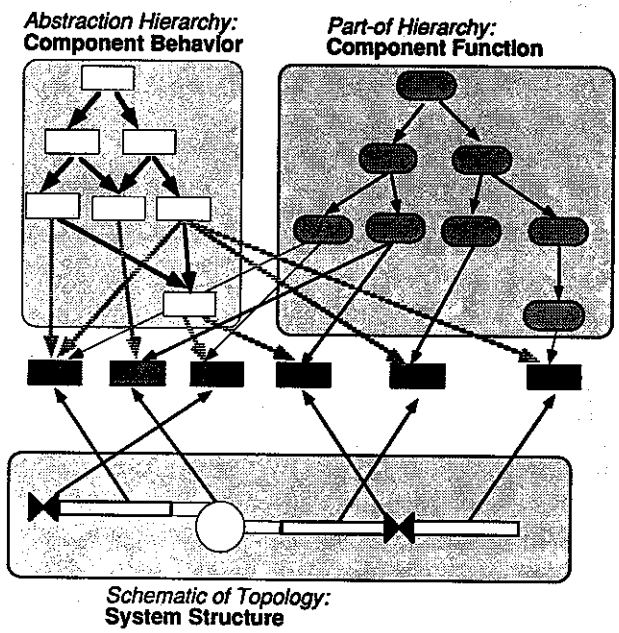


**Fig. 7.** The elements of model-based engineering. Components (shaded boxes) are inserted into a design model by a human designer or a knowledge-based system and attached as instances of the appropriate parent subclasses in a frame hierarchy termed a "component library" (white boxes) from which they inherit appropriate behavior. Components derive their function from their position in the part-of hierarchy (shaded ovals) and their topology in the schematic diagram. Design synthesis is completed by propagating attribute values among components with design rules. System behavior is obtained by exercising the inherited component behavior along the topological links in the schematic diagram with local component attributes as arguments.

### 3.2 Integration of Qualitative and Quantitative Behavior in MBR

Model-based reasoning (MBR) is a way to model an engineering system in a way that is intuitive and efficient to implement and that permits multiple kinds of reasoning about the system. It employs the AI-inspired representation and reasoning techniques of rules, frames, and object-oriented programming. Also, MBR encompasses notions of qualitative reasoning about structure and function, and it extends the reach of KBES techniques from classification problems, such as diagnosis (for which they are already well established), to problems that involve the formation of solutions from primitive elements, that is, design and planning.

KBESs developed in this spirit of model-based reasoning are intended to be models in the usual engineering sense (cf. section 1). They explicitly model the behavior of components—either quantitatively or qualitatively—and propagate effects among topologically linked objects representing system components or subsystems in a manner that models *causality* in the system as it is understood by the modeler. In the course of their reasoning, therefore, such KBESs generate descriptions of intermediate and final states of the modeled system (in terms of patterns of the system or component attribute values) that can be checked to validate the descriptive power of the model. Thus, KBESs developed in the MBR style represent structure, function, and behavior explicitly; they deduce the states of the modeled system by qualitative or quantitative simulation of component behavior [33]. (The four basic points of the model-based reasoning approach to engineering KBESs are illustrated in Fig. 7.)

Inasmuch as designs are increasingly developed on-line, much of the data needed to support MBR for configuration, diagnosis, simulation, and planning tasks can be extracted from data bases (as we have seen in the LSC Advisor application) rather

than entered at the keyboard in response to prompts. The architecture and methodology that we describe here use AI-based techniques to interact with CADD, data base, and algorithmic software currently used by engineers for design and analysis. The way in which AI and CADD systems can be integrated to support MBR is described in more detail in [32]. We review the main points of the approach here.

Further, for all but the most unique products, designers at CADD workstations can now extract components from a company, vendor, or industry data base of standard components, stored in an appropriate CADD format, and insert them into a particular synthesis. A series of components that has been selected, resized, or adapted in some other way, as well as appropriately connected to other components in the model, defines a unique *instance* of a product. For products designed in this way the component records or "blocks" in the CADD data base contain—or point to records in an external data base that contain—several kinds of descriptive information (e.g., geometry, material properties, manufacturing specifications, and contractual data) that can be used by a KBES to support design, manufacturing, and operation of the product.

Typically, each component is represented in a single frame at the level of detail at which the engineer wishes to reason about the engineered system. A single component frame can thus be used to represent a subassembly consisting of hundreds of separate parts or an individual keyway, hole, or washer in a product. The frames in such a product model can represent geometric, physical, and administrative attributes of a product's components together with their topological structure. All this information about the structure of a product and the local values of its component attributes is then available in a representation easily accessible to KBESs for several kinds of engineering tasks.

The specific components that make up an engineering system can be viewed as instances of more general classes of components. Some component attributes and their values—especially those describing component behavior—can be defined at the most generic class level to which components belong (e.g., in a class called STRUCTURAL_COMPONENTS). Additional attributes and their values representing descriptive properties or behavior of more specific subclasses of components (e.g., FLOOR_SYSTEM_COMPONENTS or WALL_SYSTEM_COMPONENTS) can then be added at each level of specificity and inherited down the abstraction hierarchy to particular instances of each type of component.

A KBES can deduce a substantial amount of knowledge about the roles that individual components (such as beams, joists, or columns) play in the functioning of a product by reasoning about the part-of hierarchy for the product and the topological links among its components. Thus, by noting that a beam is PART_OF a structural subsystem and is CONNECTED_TO a generator platform, for example, we might conclude that its role or function in the product is to provide structural support for the generator. This allows the beam to deduce that it supports the generator and to reference the weight of the generator in computing its size.

MBR can be used to predict the behavior of the modeled system by simulation. Inherited methods give components the correct behavior in the context of local descriptive data. The topological information stored with each component allows such a model to deal with interactions among its components (e.g., current flow between connected components of an electronic circuit, fluid flow through a piping system, load paths through a structure, or the propagation of vibration through an airframe).

The simulation of system behavior can be quantitative, as in conventional engineering analysis programs, which use mathematical formulas or numerical algorithms to model system behavior. And while such simulation packages have traditionally been written in conventional procedural programming styles, we may also anticipate that object-oriented programming (OOP) styles well be used in the future. For example, the case has already been made that an OOP approach may offer substantial advantages for modeling construction robotics [40], as well as for finite element modeling [41].

One of the principal advantages of the MBR approach to engineering KBESs is the extensive use of generic component libraries represented as frames, whose attributes and behavior can be inherited by instances of the components in engineered products or systems. This provides two important results. First, it drastically reduces the number of new rules that need to be employed in a given MBR engineering application. As a company develops new products that incorporate generic components, much of the system behavior is already captured in the inherited behavior of the components. The volume of new rules to be encoded in order to define configuration or diagnosis knowledge for new products will thus increase linearly—or more slowly, if new products share configuration or diagnostic knowledge with past products—rather than exponentially, as would be the case with a rule-based system.

Second, frame-based inheritance is a much easier

form of deduction to program and maintain then rule-based inference (see, e.g., [1], chap. 6). Thus, MBR systems implemented in this way are far easier for an organization to develop and maintain than corporate knowledge bases consisting only of production rules.

The MBR approach has been applied to building design in the RATAS system [42] and has been used to synthesize designs in a product model in the intelligent boiler design system [43]. Model-based systems can also provide excellent support for automated diagnosis of faults in engineered systems. A set of system failure modes can be established as goals to be investigated by firing a set of production rules or demons that propagate failure conditions and effects forward and backward along topological links. Since we model all the components of a system (at whatever level of analysis we wish to conduct), along with their behavior and their topology, intermediate values of other component properties can be deduced, then checked manually or by automated sensors linked to the KBES to confirm or disconfirm a line of reasoning. Domain-specific applications include the simulation of a $CO_2$ recycling system for a space station [44]; the Guardian system for monitoring and control of intensive care patients [45]; and the OARPLAN system that encodes product topological data using a high-level CADD interface [46] and imports it into a KBES via a generic, object-oriented product model to automate construction planning for buildings [47].

## 4 Conclusions

We have seen in the foregoing discussion that recent advances in AI programming techniques have significantly enhanced our ability to do intelligent computer-enhanced engineering. The advent of symbolic representation and declarative programming techniques has facilitated the representation of heuristic, experiential, and descriptive knowledge that could not be made computable with numerical representations. These advances derive directly from the fact that the symbolic representation schemes use frame data structures that are far richer data structures than data base records or tables. These developments have encouraged the development of tools that could help us better use and interpret the results of our traditional number-crunching programs.

However, beyond the development of KBESs to act in support of numerical and graphical work, new paradigms such as model-based reasoning will more drastically change the role of engineering workstations. These are new ways of thinking about typing and encapsulating engineering knowledge. They will facilitate the formalization and communication of data and information about component behavior as well as component state, and of part-of hierarchies, topological networks, precedence diagrams, and other semantic networks important to engineering reasoning. Thus, AI environments based on frames and object-oriented programming can act as a powerful "glue" to connect many kinds of engineering software in ways that permit the sharing of higher-level information—knowledge, rather than data—that is needed to integrate them in more powerful ways than hitherto possible. Hence, AI techniques can and will enhance numerical and graphics applications. They will extend the use of computers to new applications, and stitch all the preceding together in extended and integrated engineering environments.

## Acknowledgments

Professors J. H. Garrett, Jr., and D. R. Rehak (Carnegie Mellon University) provided helpful and insightful reviews of drafts of this paper. The first author is grateful to J. E. A. John (Dean of Engineering) and R. D. O'Brien (Provost) for a sabbatical leave provided by the University of Massachusetts at Amherst and to CMU's Civil Engineering Department for providing a positive and supportive environment during the completion of this manuscript.

## References

1. Dym, C.L.; Levitt R.E. (1990) Knowledge-Based Systems in Engineering. New York McGraw-Hill
2. Amarel, S. (1978) Basic themes and problems in current AI research. In Proceedings of the 4th Annual AIM Workshop (Ed. V.B. Ceilsielske). Rutgers, NJ, Rutgers University Press
3. Dym, C.L. (1985) Expert systems: New approaches to computer-aided engineering. Eng. Comput. 1(1), 9–25
4. Stefik, M.J. (1990) Introduction to Knowledge Systems. Xerox Palo Alto Research Center, Palo Alto, CA
5. Stefik, M.J. Aikins, J.; Balzer, R., et al. (1982) The organization of expert systems: A prescriptive tutorial. Technical Report No. VLSI-82-1, Xerox Palo Alto Research Center, Palo Alto, CA
6. Fenves, S.J. (1982) Expert systems in civil engineering. Invited lecture, MIT Workshop on Microcomputers in Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA
7. Brown D.C.; Chandrasekaran B. (1989) Design Problem Solving. London: Pitman; San Mateo, CA: Morgan Kaufmann
8. Levitt, R.E.; Tommelein, I.D.; Hayes-Roth, B.; Confrey, T. (1989) SightPlan: A blackboard expert system for constraint based spatial reasoning about construction site layout. Technical Report No. 020, Center for Integrated Facility Engineering, Stanford University, Stanford, CA

9. Maher, M.L. (1984) HI-RISE: A knowledge-based expert system for the preliminary design of high rise buildings. Ph.D. dissertation, Department of Civil Engineering, Carneigie Mellon University, Pittsburgh, PA
10. Mittal, S.; Dym, C.L.; Morjaria, M. (1986) PRIDE: An expert system for the design of paper handling systems. IEEE Comput. 19(7), 102–114
11. Dym, C.L. (Ed.) (1985) Applications of Knowledge-Based System to Engineering Analysis and Design. American Society of Mechanical Engineers, New York
12. Turkiyyah, G.M.; Fenves, S.J. (1987) Knowledge-based analysis of structural systems. In: Proceedings of the Second International Conference on Artificial Intelligence in Engineering, II, Cambridge, UK
13. Ganguly, J.; Sriram D.; Kausel, E. (1990) QSEIS: A knowledge-based system for deriving qualitative seismic behavior of structural systems from structural behavior of structural systems from structural descriptions. In: AI in Computational Engineering (Ed. M. Kleiber). London: Ellis Horwood Ltd.,
14. Salata, S.E.; Dym, C.L. (1991) Representation of strategic choices in structural modeling. J. of Comput. Civ. Eng., 5(10), in press
15. Jain, D.; Luth, G.P.; Krawinkler, H.; Law, K.H. (1990) A formal approach to automating conceptual structural design. Technical Report No. 31, Center for Integrated Facility Engineering, Stanford University, Stanford, CA
16. Luth, G.P. (1990) Reasoning and representation for integrated structural design of high rise commercial office buildings. Ph.D. dissertation, Department of Civil Engineering, Stanford University, Stanford, CA
17. Garrett, Jr., J.H.; Fenves, S.J. (1989) Knowledge-based standard-independent member design. J. Struct. Eng. 115(6), 1396–1411
18. Garrett, Jr., J.H.; Hakim, M.M. (1991) An object-oriented model of engineering design standards. J. Comput. Civ. Eng., submitted for publication
19. Dym, C.L.; Ivey, E.S. (1980) Principles of Mathematical Modeling. New York: Academic Press
20. Dym, C.L. (1983) Analysis and modeling in mechanics: An informal view. Comput. Struct. 16, 101–107,
21. Shames, I.H.; Dym, C.L. (1985) Energy and Finite Element Methods in Structural Mechanics. New York: Hemisphere Publishing
22. McDermott, J. (1981) R1: The formative years. AI Mag. 2(2), 21–29
23. Chandrasekaran, B.; Mittal, S. (1984) Deep versus compiled knowledge approaches to diagnostic problem-solving. In: Developments in Expert Systems (Ed. M.J. Coombs). London: Academic Press
24. Bennett, J.; Cleary L. Englemore, R. Melosh, R. (1978) SACON: A knowledge-based consultant for structural analysis. Report No. STAN-CS-699, Department of Computer Science, Stanford University, Stanford, CA
25. Specification for the Design, Fabrication, and Erection of Structural Steel for Buildings. (1980) American Institute of Steel Construction, Chicago, IL
26. Dym, C.L.; Henchey, R.P.; Delis, E.A.; Gonick, S. (1988) Representation and control issues in automated architectural code checking. Comput. Aided Des. 20(3), 137–145
27. Lathrop, J.K. (Ed.) (1985) Life Safety Code Handbook, 3rd ed. National Fire Protection Association, Worcester, MA
28. Simon, H.A. (1981) The Sciences of the Artificial, 2nd ed. Cambridge, MA: MIT Press
29. Gruber, T.R. (1989) The Acquisition of Strategic Knowledge. San Diego, CA: Academic Press
30. Goldberg, A.; Robson, D. (1983) Smalltalk-80: The Language and Its Implementation. Reading, MA: Addison-Wesley
31. Stefik, M.J.; Bobrow, D.G. (1986) Object-oriented programming: Themes and variations. AI Mag. 6(4), 40–62
32. Levitt, R.E. (1990) Merging artificial intelligence with CAD: Intelligent, model-based engineering. Henry M. Shaw Memorial Lecture, North Carolina State University, Raleigh, NC. Reprinted as Working Paper No. 28, Center for Integrated Facility Engineering, Stanford University, Stanford, CA
33. Kunz, J.C.; Stelzner M.J.; Williams, M.D. (1989) From classic expert systems of models: Introduction to a methodology for building model-based systems. In: Topics in Expert System Design (Eds. G. Guida; C. Tasso). Amsterdam: North-Holland, pp. 87–110
34. Touretzky, D.S. (1984) LISP: A Gentle Introduction to Symbolic Computation. New York: Harper & Row
35. de Kleer, J. (1979) Qualitative and quantitative reasoning in classical mechanics. In: Artificial Intelligence: An MIT Perspective (Eds. P.H. Winston; R.H. Brown). Cambridge, MA: MIT Press
36. Bobrow, D.G. (1985) Qualitative Reasoning about Physical Systems. Cambridge, MA: MIT Press
37. de Kleer, J. (1987) Qualitative physics. In: Encyclopedia of Artificial Intelligence Vol. 2 (Ed. S.C. Shapiro). New York: John Wiley
38. Fenves, S.J. (1990) Personal Communication, October
39. Moses, J. (1971) Symbolic representation: the stormy decade, Communic. Assoc: Computing Machinery, 14(8), 548–560
40. Keirouz, W.T.; Rehak, D.R.; Oppenheim, I.J. (1988) Issues in domain modeling of constructed facilities. In: Proceedings of the 5th International Symposium on Robotics in Construction, Tokyo
41. Baugh, Jr., J.W.; Rehak, D.R. (1989) Computational abstractions for finite element programming. Technical Report No. R-89-182, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA
42. Bjork, B-C. (1987) RATAS: A proposed Finnish building product model. Studies in Environmental Research No. T6, Helsinki University of Technology, Otaniemi, Finland
43. Riitahuhta, A. (1988) Systematic engineering design and use of an expert system in boiler plant design. In: Proceedings of the ICED International Conference on Engineering Design, Budapest, Hungary
44. Malin, J.; Lance, N. (1985) An expert system for fault management and automatic shutdown avoidance in a regenerative life support system. In: ISA 1985 Proceedings, ISA Paper No. 85-0333,
45. Hayes-Roth, B.; Washington, R. Hewett, R.; Hewett, M.; Seiver, A. (1989) Intelligent real-time monitoring and control. Proceedings of IJCAI 89, Minneapolis, MN, August
46. Darwiche, A.; Levitt, R.E.; Hayes-Roth, B. (1989) OARPLAN: Generating project plans by reasoning about objects, actions and resources. Artif. Intell. Eng. Des., Anal. Manuf. 2(3), 169–181
47. Ito, K.; Yasumasa, U.; Levitt, R.E.; Darwiche, A. (1989) Linking knowledge-based systems to CAD design data with an object-oriented building product model. Working Paper No. 7, Center for Integrated Facility Engineering, Stanford University, Stanford, CA