Knowledge-Based Computational Support For Architectural Design

Gianfranco Carrara — *University degli Studi di Roma, Italy*
Yehuda E. Kalay — *University of California, Berkeley, U.S.A.*
Gabriele Novembri — *Cartesiana Consortium, Rome, Italy*

The process of architectural design aims to define a physical form that will achieve certain functional and behavioral objectives in a particular context. It comprises three distinct, but highly interrelated, operations: (1) Definition of the desired objectives; (2) production of alternative design solutions; (3) evaluation of the expected performances of the solutions and their comparison to the predefined objectives. Design can be viewed as a process of search for a solution that satisfies stated needs, while at the same time adapting the needs to the opportunities and limitations inherent in the emerging solution.

Computational techniques were developed to assist each one of the three operations, with varying degrees of success. We propose to integrate all three operations into one whole, by developing a computational model that will facilitate smooth transition from one operation to another. The role of computers in supporting this model will include providing a knowledge base of prototypical design objectives and solutions, storing project-specific design goals and solutions, and predicting their expected performances.

This paper discusses the rationale and background for developing such a knowledge-based design system, and presents the parameters for implementing it as a computational tool to support architectural design. Examples from a prototype implementation serve to illustrate the discussion.

Introduction

We consider architectural design a goal-directed search process, whose purpose is to define an object (or an environment) that achieves some desired behavioral and spatial characteristics, while conforming to and relying upon cultural, social, environmental, and other norms. We refer to the object that is being designed as the *solution*, and to the desired behavioral and spatial characteristics it strives to achieve as the *goals*. We view the process of design as a *dialogue* between the goals and the solutions within the particular social and cultural context of the project. The dialogue adapts and modifies the initial goals and solutions until they converge, such that a solution is found which achieves an acceptable set of performance characteristics (what Simon called a "satisficing" solution [1969]). This process is depicted schematically in Figure 1.

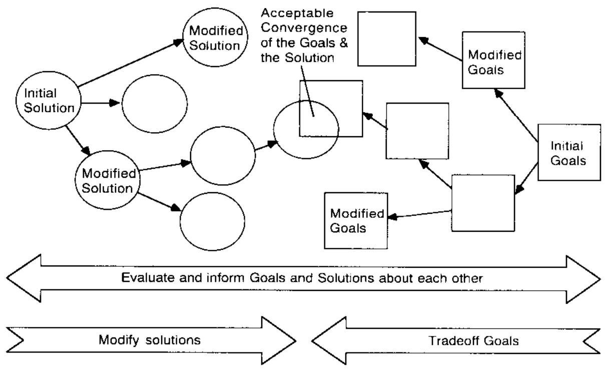


Figure 1: Design as a bi-directional search for a "satisficing" solution.

As we near two decades of research focused on computer aided architectural design, it is appropriate to take stock of the impact of computers on architectural education and practice. The research to date has been exciting, diverse, exploratory, and productive. Pilot projects, experimental labs and development programs have become a common feature of architectural schools and it appears

The process by which goals are generated is, essentially, *analytic* and *deductive*. It starts by defining a set of needs representing the wants and wishes of the sought solution. These are translated into statements that define more specifically the expected *behavior* of the solution, expressed as structured sets of *requirements*. The process by which solutions are generated, on the other hand, is *intuitive* and *inductive*. It begins by inventing or adapting a solution that appears to have performance characteristics that achieve the requirements, as well as other desired attributes (form, style, etc.). This initial solution is analyzed by means of deductive reasoning processes, which generate an abstraction whose expected behavior can be compared to the desired behavior stated by the requirements. At the same time, the requirements are modified to overcome irreconcilable conflicts, and to accommodate emerging opportunities discovered as the process unfolds.

Design activities that are based on intuition and induction cannot, presently, be substituted by computational means. Computers can only *assist* designers by providing examples, precedents, case studies, prototypes, and their derivatives [Flemming]. However, choosing and adapting the appropriate solution to the context of the problem remains a human prerogative. On the other hand, design activities that are based on deductive reasoning can (theoretically) be fully supported by computational means.

To implement this approach, we have developed a computational model of the design process which conforms to the Partnership Paradigm (discussed in [Swerdloff & Kalay 1987]), and is based on the method of design knowledge representation discussed in [Carrara et al 1992].

A Computational Model of Architectural Design

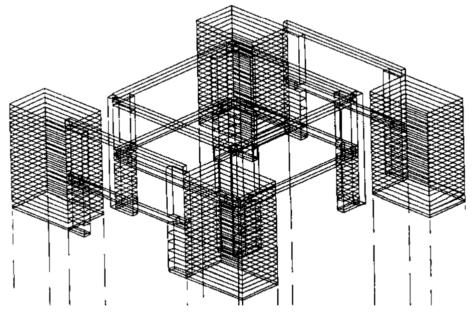
The objective of any architectural design process is to define a building that will achieve certain functional and aesthetic needs. This is done through two means: (1) by defining the conditions the building and its constituent components ought to satisfy, and (2) by defining a specific set of spaces and enclosures (building objects) along with the methods of their construction and use.

The two kinds of entities rely on different representations: the first uses representations of *specific building objects* (e.g., walls, spaces, and materials), while the second uses representations of *sets of desired performances*. Such sets are defined not by listing all their individual members, but rather by the functional needs they fulfill.

In order to arrive at a generally acceptable and useful representation of needs, they must be defined objectively and unambiguously. This eliminates most, if not all, aesthetic criteria from qualifying as design objectives in our implementation. Functional criteria, on the other hand, can be generalized into objective representations of needs, which we call *requirements*. The requirements establish alternative sets of qualitative (and quantitative) preferences that ought to be achieved by the solution. We call the specific values that satisfy a particular requirement in a particular situation *performance*, and the set of all requirements and performances the *behavior* of the represented class.

Requirements can be classified according to their effect on the building as a whole. Two main classifications are common: (1) Requirements that affect the *spatial* aspects of the building; and (2) requirements that affect the physical or *technical* aspects of the building. We have represented both kinds of requirements explicitly and made them user-accessible. The system evaluates the proposed solutions by comparing their expected performances to the stated requirements.

We have chosen to represent spatial requirements in terms of Space Units (SU) and Building Units (BU). Space Units define the requirements that are associated with individual rooms (or their equivalents), such as dimensions, type of use, environmental conditions, and so on. Building Units define classes of requirements that are associated with *structured sets* of Space Units or, recursively, other Building Units. Similarly, we have chosen to represent the requirements associated with the technical aspects of the building in terms of Functional Elements (FE) and Functional Systems (FS), which are structured sets of FEs, or, recursively, other FSs.



The stated requirements can be achieved by many different but functionally equivalent objects which possess the desired performance values. Objects are organized in the database according to different structured properties, which include hierarchy, topology, geometry, and function. *Hierarchical structures* provide an organizational schema of parts and wholes, where parts may be composed of sub-parts. *Topological structures* help organize the reciprocal interrelationships between objects. They provide a model of building use, both spatially and technologically. *Geometrical structures* are, of course, the primary instrument used by designers to represent objects. Not only does geometry help "materialize" objects in ordinary space, but it also provides a framework onto which properties, such as dimensions, materials, cost, and even aesthetics, can be attached. *Functional structures* help establish performances that result from combinations of objects according to specific hierarchical, topological, and geometric relationships.

Implementation

To demonstrate our proposed model of architectural design we have developed a system called KAAD (Knowledge-based Assistant for Architectural Design), which is intended to help architects specify design objectives, adapt existing or create new design solutions, evaluate their expected performance and compare them with the stated objectives in the context of designing health care facilities for treating infectious diseases.

Adaptation and refinement of design solutions is assisted by a knowledge base comprising prototypical design solutions. It includes much of the information pertinent to generic building objects such as walls, windows, doors, rooms, and corridors in the context of hospitals. Particular solutions are derived from the prototypes by adapting them to the specific context of the problem, adding dimensions, relationships, orientations, and specific performance characteristics.

To support the generation of new solutions, the system manages the representation of design requirements and includes a host of evaluators for predicting and comparing the expected performances of the emerging solutions with the stated objectives. A graphical and analytical user interface facilitates the communication between the system and the designer.

The frame-based implementation of the knowledge base enables us to use a single set of operators to manage all the data and to guarantee its integrity, while saving us much effort and greatly reducing the overall size of the system. Following frame conventions, each entity in the system is either a prototype or an instance. Prototypes constitute the system's knowledge base, and instances constitute its database. Each object, whether a prototype or an instance, is represented by a set of attributes (slots), which may assume a value selected from one of several types (facets). Such values may consist of a number, a text string, another frame, or a procedure. Several slots and facets are shared by all the objects. Shared slots include:

ISA	("is a") designates an object as an instance of the prototype referenced by the slot.
AKO	("a kind of") designates a prototype as special sub-class of the more general class of objects which are defined by the prototype referenced by the slot (the AKO slot is used to define hierarchies of prototypes).
IMS	("immediate successor") establishes an assembly (part-whole) relationship among instances (the referencing instance is considered part of the objects referenced by this slot).
SHAPE	contains procedures to compute the geometric representation of the object.

Shared facets include:

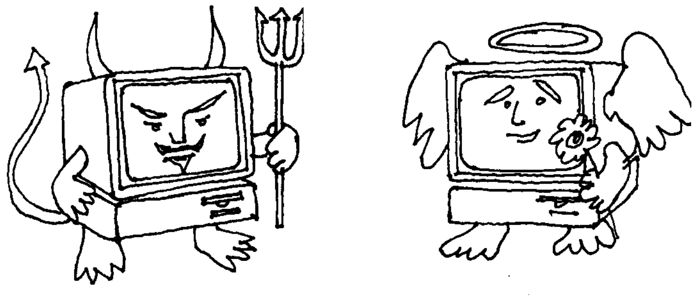
VALUE	contains the value associated with the slot.
DEFAULT	establishes the value which will be used if no other value has been specified.

inevitable that computers will become the dominant media for design in the near future. Architectural applications are moving from experimentation to mass adoption. As architectural schools balance needs and resources, assessing the past will play an important part in facing the immanent critical decisions.

MAX, MIN	defines the max/min values for the slot (a range).
ALLOW-SET	defines the set of allowed values for the slot (an enumeration).
CHECK-LIST	establishes the list of mandatory values for the slot.
TO CALC	("to calculate") provides the object with a method to calculate the value of the slot when the system requires it. This facet contains a procedural attachment, implemented as a Lisp procedure, which is invoked when the value is required. The facet also implies the existence of an additional facet, C-TO-CALC ("compiled to calculate"), which contains the compiled version of the method. KAAD compiles the method the first time the procedure is invoked and stores it in the C-TO-CALC facet for future use.
TO VERIFY	contains methods used to verify that the assigned values are allowed values, as defined in the previous facets.
TO DELETE	contains the method to delete the value associated with the slot.
TO DRAW	contains the procedural attachment used by the system to draw the object.

As an example, consider the implementation of a simple SU (space unit) prototype of a nursing room in a hospital, which includes the slots of ADS ("adjacent"), establishing a spatial relationship between the SU and the other SUs which make up the whole building; COM ("communication"), defining a system of routes among SUs; FAR, establishing that the given SU cannot be near or adjacent to some other kinds of SUs; SUP, defining the maximum and minimum values of net surface; WTEMP and STEMP, defining the values of the internal winter and summer temperature, REWH and RESH, defining absolute humidity conditions; and VENT, defining the number of air exchanges per hour:

```
(dg3 (ako value su))
        (description (value "space unit for patient's
         nursing room"))
        (com (value conn1))
        (adj  (value dg6 dg7 ))
        (far  (value dg2 dg11 dg15 conn2))
        (ims  (value hfur3 ite ))
        (sup  (min 22)
              (unit mq)
              (max 28)
              (unit mq)
              (description "minimum and
               maximum net area")
        (wtemp (range 19 21)
              (unit °C)
              (description "interior winter
               temperature")
        (stemp (range 25 27)
              (unit °C)
              (description "interior summer
               temperature")
        (rewh (range 40 60)
              (unit %)
              (description "winter relative humidity")
        (resh (range 40 60)
              (unit %)
              (description "summer relative humidity")
        (vent (value 2)
              (unit vol/h)
              (description "number of air exchanges")
        (vela (value 0.2)
              (unit m/sec)
              (description "air velocity")
        (pura (value 4)
              (unit % )
              (description "air purity level")
        (press (value 10)
              (unit Pa)
              (description "pressure")
        (sound (max 42)
              (unit dbA)
              (description "maximum noise level")
        ..................
        ..................
        ..................
```



Other prototypes used by KAAD have been structured in a similar manner and will not be elaborated upon here.

How the System Works

To demonstrate the structure of the knowledge base and how the system works, we will use as an example a small, real case comprising the nursing module shown in Figure 2, in which the hatched arrows represent the communication between rooms and the white ones represent adjacent rooms. For the sake of simplicity the adjacency with the outside and with other modules of the hospital have not been represented. The case study, albeit of reduced complexity, is significant because buildings for treating infectious diseases must respond to a considerable number of constraints, many of which are often very important. The designer must fulfill specific requirements related to the treatment of symptomatic seropositive or AIDS-infected patients, and must guarantee an adequate protection of the patients against the risk of crossed or opportunistic infections. At the same time the designer must guarantee an adequate level of protection to the visitors and the staff by carefully evaluating paths, entrances, filter and reclamation areas and dressing rooms.

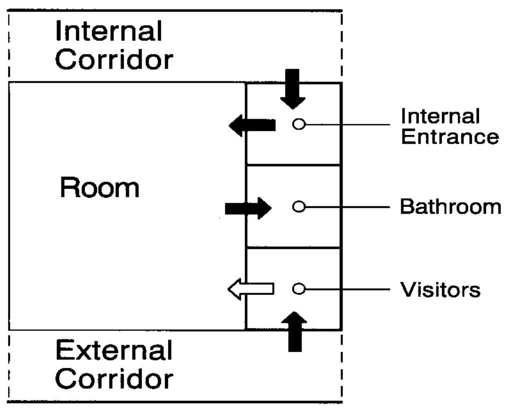


Figure 2: BU (Building Unit) representing a nursing unit module in an infectious diseases health care facility.

The design process is based on the instantiation of knowledge base prototypes. The designer may start by defining several characteristics and constraints that KAAD uses to instantiate the appropriate prototypes of the FEs (Functional Elements). Typically not all the information needed to completely define the FE instance is specified. KAAD begins a slot-filling process that attempts to complete the FE, asking the designer for the necessary values and the specification as needed. When the designer has specified the geometric information, the system calculates many of the values needed to complete the instantiation process, such as adjacencies, paths, areas, and so on.

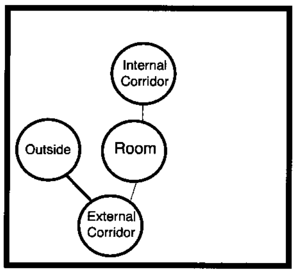
This design process may be characterized as Top-Down design and is depicted in Figure 3. KAAD can also operate in a Bottom-Up design mode. In that case, the designer may start by drawing lines and graphical objects representing a design solution. Starting with such graphical primitives, the designer may tell the system that one of them represents the instance of a particular physical object. KAAD will look up the corresponding prototype, and if it exists in the knowledge base it will instantiate the information it contains and attribute it to the instance. This bottom-up mode of design is depicted in Figure 4.

Along the way, KAAD performs checks to verify that adjacency, size, and other requirements are met. If it discovers a violation, KAAD warns the designer.

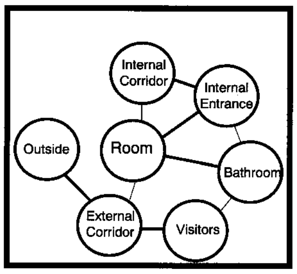
KAAD has been implemented in Lisp and C. All the components concerning the knowledge base have been implemented using the Allegro Common Lisp 3.0 from Franz, Inc. The parts of the system concerning the graphic and the database management components have been implemented using the C language. The user interface was developed under X11R3.

The first implementation of the prototype was developed in the UNIX environment. At present, a PC version under Microsoft Windows 3.1 using Allegro CLl/PC 1.0 and Borland C^{++} is in the final stages of development.

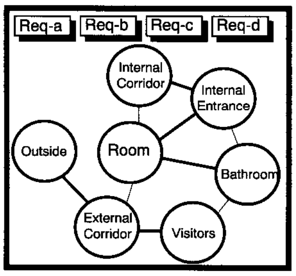
"No matter how completely technics relies upon the objective procedures of the sciences, it does not form an independent system like the universe: it exists as an element in human culture and it promises well or ill as the social groups that exploit it promise well or ill. The machine itself makes no demands and holds out no promises: it is the human spirit that makes demands and keeps promises." *Lewis Mumford*[1]



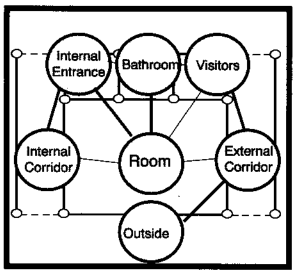
The designer can create some instances of Space System prototypes, define their characteristics and their requirements.



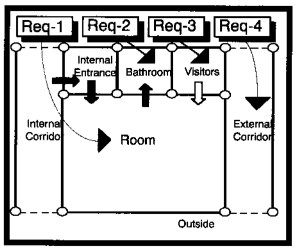
SUs and BUs can be structured, defining the Space System of the Building.



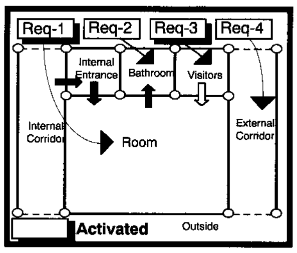
The requirements related to the Space System can be checked.



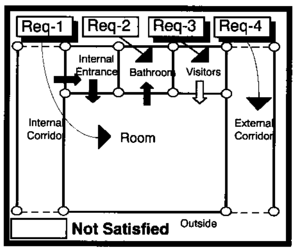
Links between the Space System and the Building System must be defined.



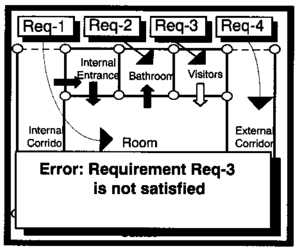
The designer can define requirements over either a specific object or classes of physical objects and/or space entities.



The requirements can be grouped, activated, or turned off.

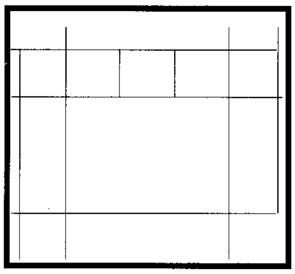


The designer can launch a verification process at every phase of a design process, or . . .

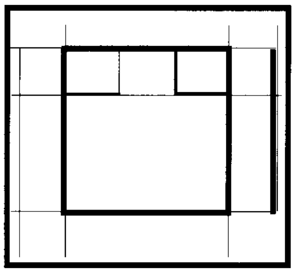


. . . ask for a continuous check of design choices.

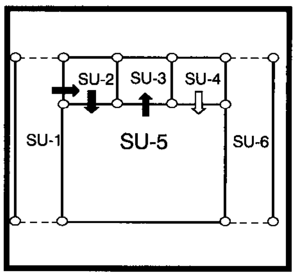
Figure 3: Top-down design process.



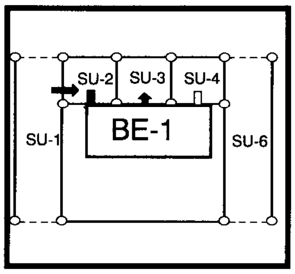
The designer can draw some graphic primitives as references for subsequent work.



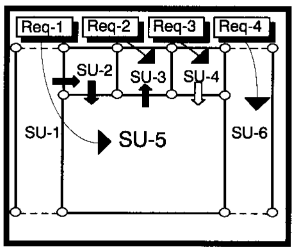
The reference points can be used for assembling components.



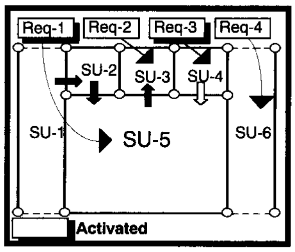
Physical objects can be organized to define Space Units (S.U.) and . . .



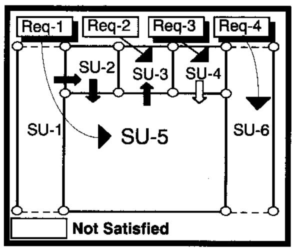
S.U. can be organized in more complex entities called Building Units (B.U.).



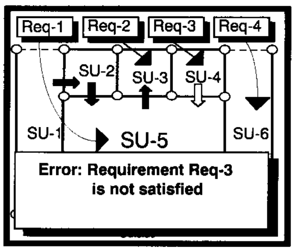
The designer can define requirements over either a specific object or classes of physical objects and/or space entities.



The requirements can be grouped, activated, or turned off.

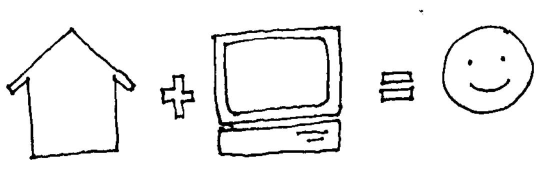


The designer can launch a verification process at every phase of a design process, or . . .



. . . ask for a continuous check of design choices.

Figure 4: Bottom-up design process.



Conclusion

Recent developments intended to model the architectural design process and to represent the knowledge it relies upon promise to become significant forces shaping the future of computer-aided architectural design. Nonetheless, the development of *integrated* knowledge-based design systems is still a matter for much research and development. The difficulty is due, in part, to a lack of distinction between design processes that can be represented as overt knowledge (in some formal way) and those that cannot. Whereas *learning*, *creativity*, and *judgment* are the hallmarks of architectural design, they are, and for the foreseeable future will continue to be, the prerogative of humans. On the other hand, there are many design processes which have already been successfully computed, such as a host of analyses, visual representations, and even certain solution generating algorithms. The difficulty, then, lies in developing design frameworks that can integrate the computable aspects of architectural design with the ones that are not currently computable, in a smooth, transparent way.

We have proposed such an integrative approach, which *facilitates* design but does not fully automate it. It is based on the observation that designers are able to cope with and manage complex design processes, and have for centuries achieved outstanding results doing so without the aid of computers. It is our contention that it is not necessary to fully automate each and every one of the design process activities in order to significantly improve design productivity and quality. Rather, it is more prudent to develop a practical *symbiosis* between the capabilities of designers and machines.

The implementation of our approach is based on viewing architectural design as a process of search, which aims to reconcile the differences between a set of *requirements*, given by the client, and the expected behavior of a design *solution*, proposed by the architect. In the course of negotiating the differences between the two ends, tradeoffs must be made on both sides. The process also contributes to better understanding the problem itself, and informs both the architect and the client of initially hidden opportunities and irreconcilable conflicts.

The prototype we have developed forms a framework for implementing our proposed *design partnership* paradigm. In such partnership, the role of the computer can be shifted dynamically between passive representation/evaluation and active generation/evaluation of design solutions. Such dynamics would allow the designer and the system to respond to changing requirements, unforeseen problems, and emerging opportunities as they arise during the design process. The system demonstrates the feasibility of implementing this paradigm. Its utility for practicing architects is, nonetheless, yet to be tested.

Acknowledgment

This paper is a shortened version of a paper on the same topic, published in [Carrara et al 1994], and is published here with the kind permission of Elsevier Science Publishers, Amsterdam.

References

Carrara G., Y.E. Kalay and G. Novembri, “Multi-Modal Representation of Design Knowledge,” *Automation in Construction*, September 1(2):111-122, 1992.

Simon H.A., *The Sciences of the Artificial*, MIT Press, Cambridge MA, 1969.

Swerdloff L.M. and Y.E. Kalay, “A Partnership Approach to Computer-Aided Design,” Computability of Design (Y.E. Kalay, ed.), John Wiley & Sons, New York, 1987.

Flemming U., *Case Based Design in the SEED System*, Knowledge-Based Computer-Aided Architectural Design (G. Carrara and Y.E. Kalay, eds.), Elsevier Science Publishers, Amsterdam, The Netherlands, 1994.

The basic question, "Are we better off because of computers?", supposes that computers are one of the key factors in our future as architects. A perusal of current opinion about the state of the architectural profession suggests that either computers have been detrimental to architecture or the application of computers, in itself, has not been a major factor. While one can find isolated