Analogy, Exploration and Generalization: Three Activities for Knowledge-Based Architectural Design Systems

François Guéna
Khaldoun Zreik

Laboratory for Research into Design and Information Technology
Lutèce'IA
9, rue Barbanègre
75019 Paris, France

We propose in this article a system architecture based on reasoning through analogy with past cases or situations. Starting with a project and a sketch provided by the user, the system locates analogous situations in the past and uses these to improve a problem's description. A sufficiently improved description will in turn activate a constraint-satisfaction mechanism. Previous situations are stored in a memory bank of objects that match the description of past problems to the generic descriptions of past solutions. Three mechanisms can be distinguished within the system: an analogy mechanism collects hypotheses about the variables and constraints to be satisfied in past situations, an exploratory mechanism searches through the solution space, a generalizing mechanism looks at experiences and memorizes only what is needed to collect hypotheses.

Keywords: knowledge-based system, case-based reasoning, constraints satisfaction, explanation-based learning, object-oriented representation.

1 Introduction

Until very recently the use of CAAD tools was limited to the production of conventional design documents: plans, cross-sections, perspectives, images, etc. Today, in spite of the efforts of many researchers and designers, architects still do not have computer systems that can help them during the most important stage of their work— the process of searching for solutions to design problems. This limitation motivates our research into knowledge-based architectural design systems.

Our objective is to construct a system that would aid the architect in the preliminary phases of design. A possible way to achieve this objective would be to specify a CAD system architecture capable of design knowledge reasoning. Artificial Intelligence offers models and techniques that enable one to realize these types of knowledge-based systems. Choosing techniques depends first on identifying the type of task the system must realize.

A task we could delegate to this system would be helping architects to validate their hypotheses, their first sketches of a solution. The computer would attempt to complete the architect's partial, crude solutions, and then make the optimal adjustments vis-à-vis the project's requirements, while respecting the subjective preferences of the designer. In other words, the system would transform a hypothesis into a blueprint, through interaction with the designer. Such a system could also explore diverse solutions derived from different hypotheses, something that the designer does not always have time to do.

Nonetheless, identifying this task is not the same as defining a knowledge-based system architecture capable of realizing it. It is still necessary to identify the nature of the knowledge on which the system's reasoning ability is based.

The architectural sector does not rely very much on explicit knowledge. Most of the knowledge present in the solutions provided by architects is implicit; it is a form of knowledge that they cannot clearly explain. Architects often base their choices on subjective preferences. On the other hand, they are often confronted with situations which are unique. A new project always requires exploration based on hypotheses. This is even more pronounced if the project requires innovation.

We propose in this article a system with which architecture can take into account the particular nature of design knowledge: a system founded on analogy, exploration and generalization. After having presented an example of a task that could be delegated to our system, we will present our reflections on the nature of design knowledge. We will then introduce the reasoning models and techniques that will be used to construct the system.

2 An Example of a Design Task for a Knowledge-Based System

To illustrate the mechanisms we are going to describe in this article, we have chosen as an architectural problem the design of an office building floor. We have greatly reduced the complex nature of this activity for purposes of this presentation.

Let us suppose that:

- A certain percentage of area allocated for private and shared office space is required.
- The designer can, if he desires, furnish a crude preliminary solution for certain elements: for example, the physical form, the dimensions of the building envelope, or the partitions of which it is composed.
- The system's role consists in defining, connecting, and calculating the dimensions of the vertical partitions (solid or glass walls) and the horizontal partitions (floors), with the goal of assembling office spaces according to the percentage of surface required for the building envelope, circulation, natural lighting, and to meet whatever regulations are applicable.

For example, a statement given to the system by the user could be the following: The length of the floor will be 20 or 40 meters, the depth of the floor must not exceed 14 meters, and only private offices are required. Obviously, there are innumerable solutions to this problem; the system must determine which solution best suits the user, taking into account his preferences and experiences. Figure 1 presents two possibles solutions that the system could propose in response to this problem.

It is important to point out here that the user delegates to the system a task he knows how to perform, but that is fastidious and time-consuming; the type of task usually delegated to an assistant. The knowledge with which the system reasons must be provided by the user himself. In this way, the designer prepares a system that is suited to his specific needs.

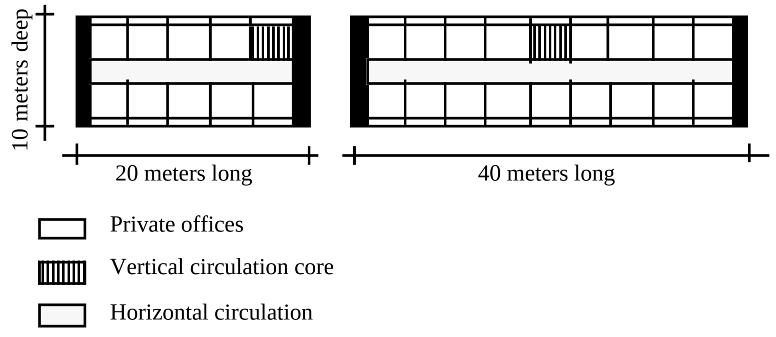

Figure 1. Two possible solutions for an office building floor with two different positions for the vertical circulation core.

This article will examine AI models and techniques which we believe could be used and combined to create the type of system we are describing. However, to better comprehend our choices, we will first make some general remarks about the nature of design knowledge.

3 The Nature of Design Knowledge

We will divide the knowledge used during a design act into three categories: explicit knowledge, implicit knowledge, and knowledge that is "unknown."

3.1 Explicit Knowledge

Explicit knowledge is easy to identify. It can be perfectly represented and faithfully transmitted to others, and it describes primarily a domain's technological knowledge. Yet we do not always know when, where and how it is used in the design process. One of the advantages of knowledge-based systems is their ability to dynamically determine which forms of explicit knowledge to use (Brown, 1988).

Among different types of explicit knowledge, we distinguish between foreseeable knowledge and unforeseeable knowledge. Foreseeable knowledge is knowledge that is valid in every designing situation. In our office building floor design example, the knowledge described below, related to fire regulations, is a type of foreseeable explicit knowledge.

> The distance that users must travel to reach a vertical circulation core, no matter where they are positioned in the building, cannot exceed 30 meters.

Unforeseeable explicit knowledge cannot be identified until part of the solution has been determined, that is, not before the designer has made certain design choices. For example, if the designer has chosen to design a floor with enclosed offices served by only one horizontal circulation route, he must not only consider the minimum length of the passageway, but also keep in mind what length would be sufficient to allow for the safe evacuation of occupants in the case of a fire. On the other hand, the constraints to consider will be different if he has instead chosen to install a single large workroom and open offices.

3.2 Implicit Knowledge

Technological know-how by itself will not enable a designer to come up with the solutions to a project. For example, numerous solutions exist to satisfy the requisite regulation constraints involved in the design of an office building floor. During the design process, the designer's "savoir-faire" always plays a privileged role; unfortunately, it cannot be easily rendered explicit. Either it is dependent on a particular context about which it is difficult to generalize, or it is founded entirely on the designer's subjective preferences. This knowledge is implicitly present in the solutions produced by the designer.

In the given context, implicit knowledge is an unstated way of doing things, a form of practice that is not described, which is inarticulate. It represents those situations where the designer does not have the means available to explain an action or describe an object he has put under his control. In the design process, the role of implicit knowledge can be very important. For Heath (1989), innovative and original thoughts and decisions can intervene at any stage of the design process, while the nature of the knowledge that leads to an innovative or original act must be by definition implicit, in as far as the act has not yet been achieved. Zeisel (1981) confirms the importance of this type of knowledge when he defines the act of design in three activities: imagination, representation, and experimentation.

3.3 The Unknown

Beyond the explicit knowledge of technological understanding and the implicit knowledge of design "savoir-faire," there remains the unknown, the ensemble of knowledge types that will be used at sometime in the future, and about which we cannot even speculate.

Clearly, in the architectural domain each new project is unique. Each generally requires part of their exploration be done on the basis of hypotheses, especially if the project requires some innovation. This exploration can push the designer toward a refinement of existing knowledge, as well as lead him to the discovery of new knowledge.

Today, most design-assistance systems only consider the explicit aspects of knowledge. This is why we seek to give precision to the architecture of knowledge-based systems, thus allowing them to function in cases where the participation of implicit and unknown knowledge in design activity are very important.

The complex and diverse nature of design knowledge leads us to believe that several reasoning models and techniques will have to brought together in order to produce a system capable of realizing all of our objectives.

4 Exploration, Analogy, and Generalization

A formal solution to a design problem requires specification of objects and satisfaction of multiple constraints, as fully as possible. In this light, the constraint-satisfaction techniques arising from AI research (Waltz, 1972; Mackworth, 1987) seem particularly useful (Fox, 1987; Navinchandra, 1991). In principle, this is a two-step process which involves first stating the problem, and then providing solution description variables, their value domains, and the constraints at work between the variables. Exploration algorithms determine consistent sets of values for the variables. Knowledge is reprented here primarily as constraints. Nonetheless, it is difficult to apply these constraint-satisfaction techniques directly in most real-life design cases (Janssen, 1990), as the variables that allow a situation to be described cannot be clearly defined *a priori;* this also holds true for the constraints that bear on these variables.

First, as we have stated, there is the presence of unforeseeable explicit knowledge. Consequently, variables and constraints are discovered during the design process; that is, they are discovered only after part of the solution has been found.

Secondly, there is the presence of knowledge that is difficult to express precisely: preference constraints (e.g., it is preferable that the office surface not exceed 12 m ≤); approximate constraints (e.g., the wall must be approximately 18 cm ≤ thick); and symbolic constraints that can only be defined in a broad sense.

Finally, the number of highly important variables at play require recourse to two other forms of implicit knowledge: decomposition (separating the problem into sub-problems) and planning. These knowledge forms are integral parts of the designer's savoir-faire and thus they are also difficult to explain.

A line of recent research consists in combining constraint-satisfaction techniques with other AI techniques, such as analogical reasoning, or more precisely, Case-Based Reasoning (Reisbeck, 1989).

A mechanism resulting from this combination has been described elsewhere (Guéna, 1991). It is based on reasoning through analogy with past cases or situations. Starting with a project and a sketch provided by the user, the system locates analogous situations in the past and uses these to improve a problem's description. A sufficiently improved description will in turn activate the constraint-satisfaction mechanism. Previous situations are stored in a memory bank of objects matching the description of past problems to the generic descriptions of past solutions. These take the form of architectural schemas to which performances are linked.

Three mechanisms can be distinguished within the system:

- An analogy mechanism obtains solutions to design problems by collecting and storing hypotheses about the variables and constraints to be satisfied in past situations. This mechanism uses aspects of both unforseeable explicit knowledge and implicit knowledge in the problem-solving process.
- An exploratory mechanism searches through the solution space, which is the complete set of variables and hypothetical constraints. This mechanism is necessary as it remains partially unknown.
- A generalizing mechanism looks at experiences and memorizes only what is needed to collect hypotheses. Its objective is to improve on the explicit knowledge and to clarify the implicit knowledge present in the solutions.

Figure 2 presents the system's global architecture. Three levels are clearly distinguished, one for each of the mechanisms described above. We will now discuss in detail the function of each of these levels.

4.1 Analogy

The analogy mechanism has two levels. The first searches through a memory bank of past situations for the solution schema best satisfing the requirements of the new problem in terms of the user's order of preferences. The second stage adapts this past situation by combining it with certain aspects of other situations. The analogy mechanism thus states a new constraint satisfaction problem (CSP).

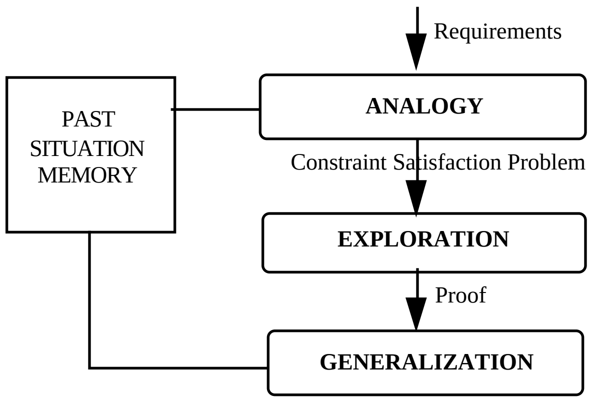

Figure 2. Global architecture of the system and its three levels.

Figure 3 presents the analogy mechanism's different modules. The techniques used by these modules will be described further on in the article.

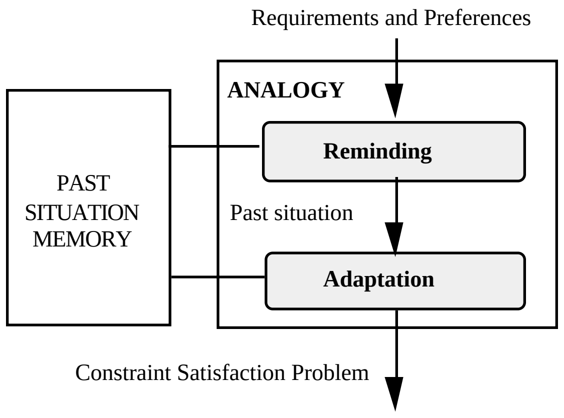

Figure 3. The Analogy Mechanism.

4.1.1 Representation of Past Situations

A past situation links the functional and/or qualitative description of a category of solutions to its structural description. Describing a category of solutions structurally entails describing a particular configuration of components (Figure 4). The components are typologically and geometrically described with the aid of constraints on their parameters. The position of components relative to each other can be described by a network of spatial constraints through which geometric information used in reconstructing the scene can be propagated.

To represent past situations, a type of frame that consolidates descriptive attributes is used. What differentiates these frame entities from the abstract models employed in *refinement and instantiation techniques* (Gero, 1988; Oxman, 1990) and the prototypes employed in reasoning by case (Kolodner, 1988) is that the values of the attributes are the domains. Therefore, the term scheme has been chosen for this type of representation

(Guéna, 1992), a term psychologists use to designate the thought structures which bridge concepts and schemas (Estival, 1988).

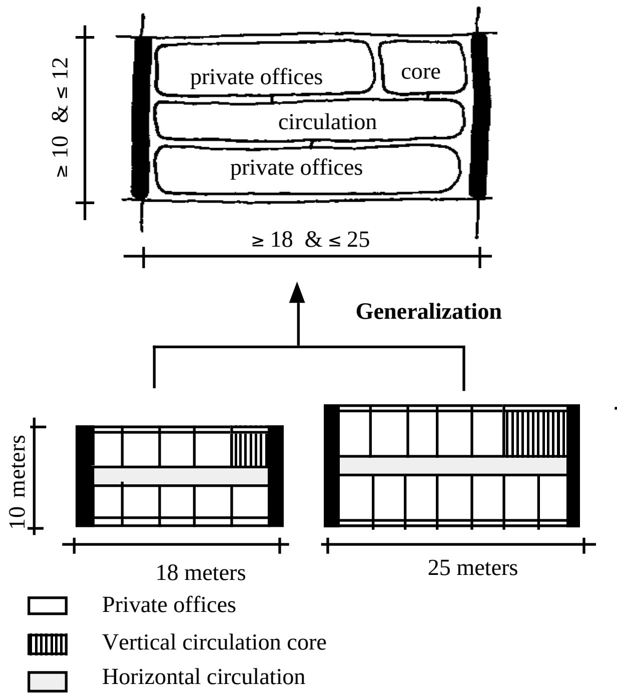

Figure 4. A composite configuration resulting from the generalization of two similar solutions.

Attribute domains are recursively constructed by combining more simple domains. They can also contain variables. They attempt to describe a category of objects, a set of wall-partitions, for example, that will be used either to generate the extention during synthesis or to verify an object's presence in this category during the reminding process (section 4.1.3). The figure below illustrates a scheme whose task is to describe the arrangement in Figure 4.

This scheme includes performance and structural attributes.Performance attributes define the boundaries for the percentage of surface to be allocated to different office categories. Given the domain constraints, the parameters here define the admissable limits for the length and depth of the floor which, if not adhered to, invalidate the arrangement (e.g., fire safety requirements). For example, the domain defined by the depth attribute is: *(and int (bounds 10 12))* that corresponds to an integer value between ten and tweleve meters. Structural attributes define domains containing variables that belong to the set of physical elements included in the arrangement (walls, windows, or partitions), and also domains that assemble sets of sub-spaces. For instance, the domain defined by the outer wall partition attribute is: *(and set (marked-list (? px1 wall) (? px2 window) (? px3 wall) (? px4 window)))*. This corresponds to a set consisting of an ordered list of four marked elements. Topological attributes describe positions relative to the different subspaces between them. There are many ways to state such spatial relations(e.g., (Flemming, 1989; Fox, 1987;

Hégron, 1990). We will demonstrate here one possible way, based on an analogy between temporal and spatial relations. The relative position of sub-spaces can express themselves by projections along two or three axes, depending on whether one wants two or three dimensions. For example, along the X axis an office space is followed by the circulation core (co follows o1) which itself begins with the horizontal circulation (o1 starts c) etc. And along the Y axis the circulation follows an office-space (c follows o1). The relationships shown in Figure 5 represent the constraints between the parameters and the structural elements, and also the configuration constraints of the elements themselves.

```
FLOOR106
 performances
  % private offices (and int (bounds 100 100))
  % shared offices (and int (bounds 0 0))
 parameters
  depth:(and int (bounds 10 12))
  length:(and int (bounds 18 25))
 structure
  outer-wall-partitions:
    (and set (marked-list (? px1 wall)
                          (? px2 window)
                          (? px3 wall)
                          (? px4 window)))
  sub-spaces:
   (and set (marked-list (? co Core-11)
                         (? o1 Office-124)
                         (? o2 Office-128)
                         (? c Circulation-134)))
 topology
  X-axis: { (o1 starts c) (co finishes c) (co follows o1)
            (c equals oself) (o2 equals oself) }
 Y-axis: { (o1 starts oself) (co equals o1) (c follows o1)
            (o2 follows c) (c finishes oself) }
 relations
  (perpendicular px1 px2)
  (perpendicular px2 px3)
  (perpendicular px3 px4)
  (equal (distance px1 px3) depth)
  (equal (distance px2 px4) length)
  ...
```

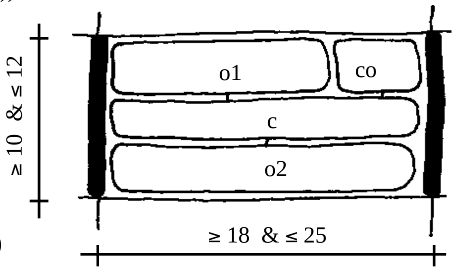

Figure 5. An example of a scheme (where oself has the value of floor106).

The schemes are linked together into a net that describes *functional decompositions* of arrangements (Figure 6). The most complete description is located at the leaf; in other words, the description that defines the most specialized attribute domains. On moving up the net to the root, the schemes define more general domains. A scheme can either determine a specific characteristic or inherit some of the characteristics of the schemes from which it is derived. If the same attribute is inherited from several schemes, a new domain is produced at the point where domains intersect on the inheritance graph. Thus, the system possesses several abstract levels for describing the performances and structure of an arrangement.

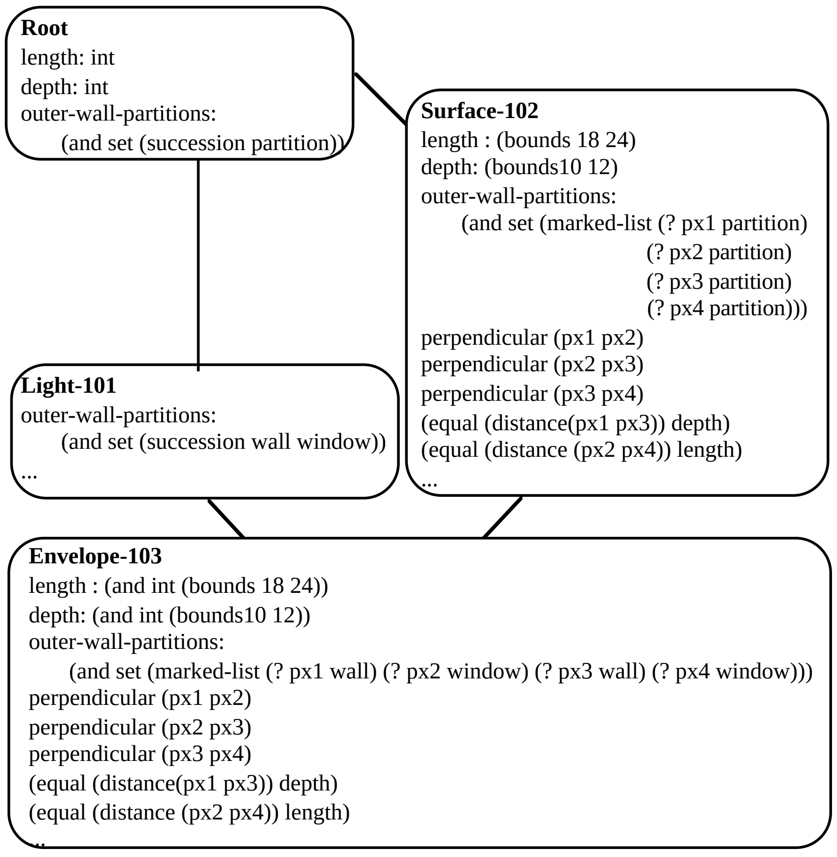

Figure 6. An example of functional decomposition for an office building envelope arrangement. The task of the schematic offshoot is to describe the elements in terms of both the light supplied by the outer wall partitions and the configuration of the floor.

Similarly, schemes can be linked together to form a network describing the structural decomposition of an arrangement (Figure 7). Instead of a functional tree that describes the decomposition of the same arrangement from different points of view, a network describes an arrangement decomposition into sub-arrangements. Thus, the system possesses several levels of decomposition for describing the performances and the structure of the arrangement.

This representation of past situations in a network of schemes allows the system to register, from the time of analogy onwards, levels of abstraction and decomposition where hypotheses can be collected and, once learned, used to refine and clarify descriptions.

4.1.2 Requirements and Preferences

For stating the problem the user must construct an arrangement model that defines the requirements by order of preference. For example, if it is required that the maximal building length be less than 30 or equal to 32 meters, and the project requires that 100%

of the surface be reserved for private office-space, the problem can be stated in the following manner:

```
(Floor-model
    (length (bounds 30 32)
    (%private-office-space (bounds 100 100))).
```

A preference is therefore accorded to the length constraint.

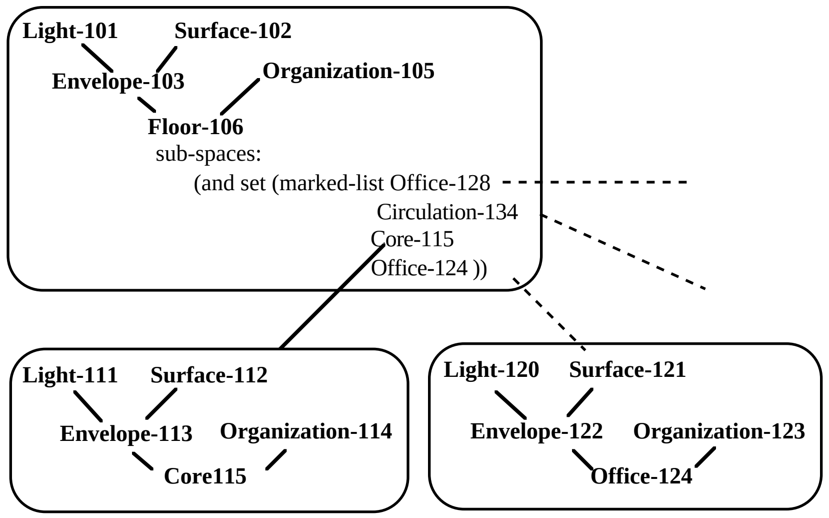

Figure 7. A structural decomposition of an office building floor arrangement. The scheme floor 106 defines, through the intermediary of subspace attributes, a domain that samples and cross-references the different sub-arrangements of the assembled spaces. In this case two arrangements for different types of offices, an arrangement for the core, and an arrangement for the circulation.

4.1.3 Reminding

The reminding mechanism is based on a type of classification algorithm (Brachman, 1985) in the sense that one tries to match the arrangement model required with an existing arrangement. Yet it differs in the sense that the arrangement best suited is not necessarily the one in which the greatest number of properties are matched with the problem's requirements. The best arrangement model matches itself with those requirements judged to be the most important by the user who states the problem. The algorithm searches for the most specific arrangement best conforming to the required model in terms of the stated preferences. One moves from the more general arrangement and for each descendant one calculates a mating score or grade as a function of the number of verified characteristics present and their rank in the order of preference. Only those descendants with the best mating scores are conserved.

The figure below presents a possible state for an arrangement memory bank. If, as we have stated above, a length preference is accorded, then the past arrangement which best matches the “floor-model” arrangement (section 4.1.2) would be Arrangement B.

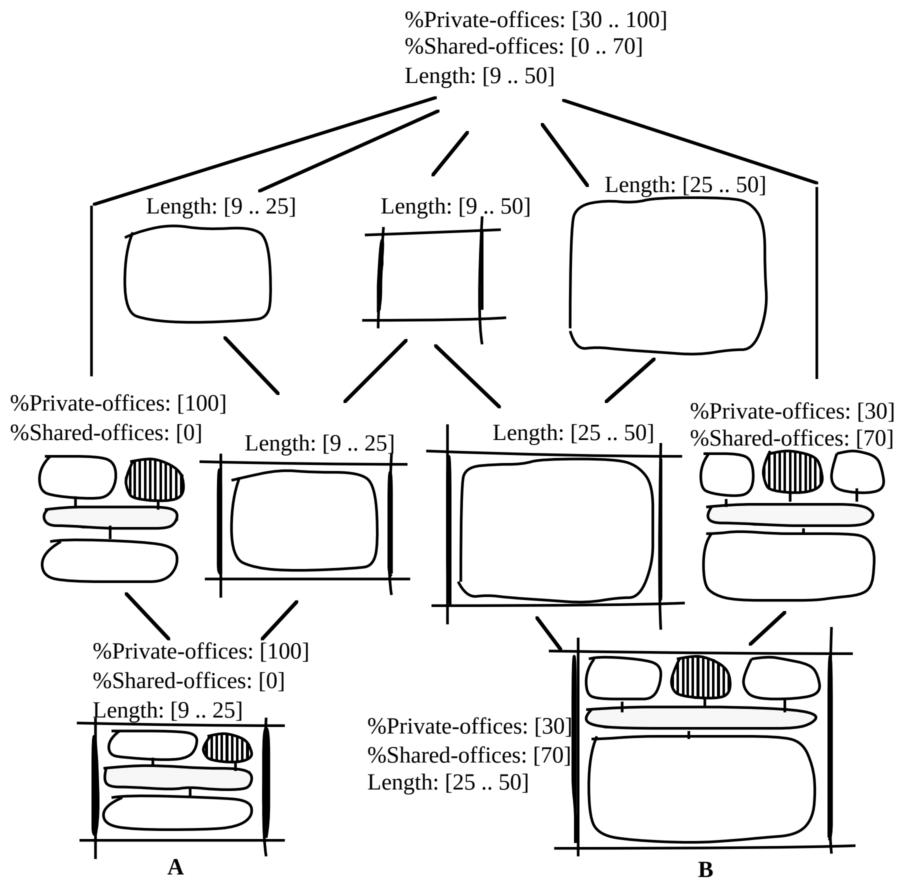

Figure 8. A possible memory state. Matching (scheme characteristics) is generally partial, as it has been acknowledged the ideal arrangement does not exist, but must be constructed. In our example the “floor-model” arrangement does not completely match up with the arrangement vis-a-vis the percentage of office spaces required.

4.1.4 Adaptation

Adaptation takes place in three stages. The first stage selects from the arrangement's operational or structural net one of the schemes that causes the dissatisfaction of the requirements to be satisfied. The second stage selects another scheme from the same level, to which the performances seem better suited. The third level consists in replacing the defective scheme with the new one, recalculating the domains of structural attributes, collecting the relations, and propagating the modifications down to the roots of the arrangement. We are trying here to construct the most specific generalization possible for the structural descriptions of the different schemes assembled.

Figure 9 presents the adaptation of Arrangement B: the system replaces its interior organization scheme with Arrangement A, which, in terms of required office-space, appears to be more appropriate. The newly constituted Arrangement C could be used as a hypothesis by the exploration mechanism.

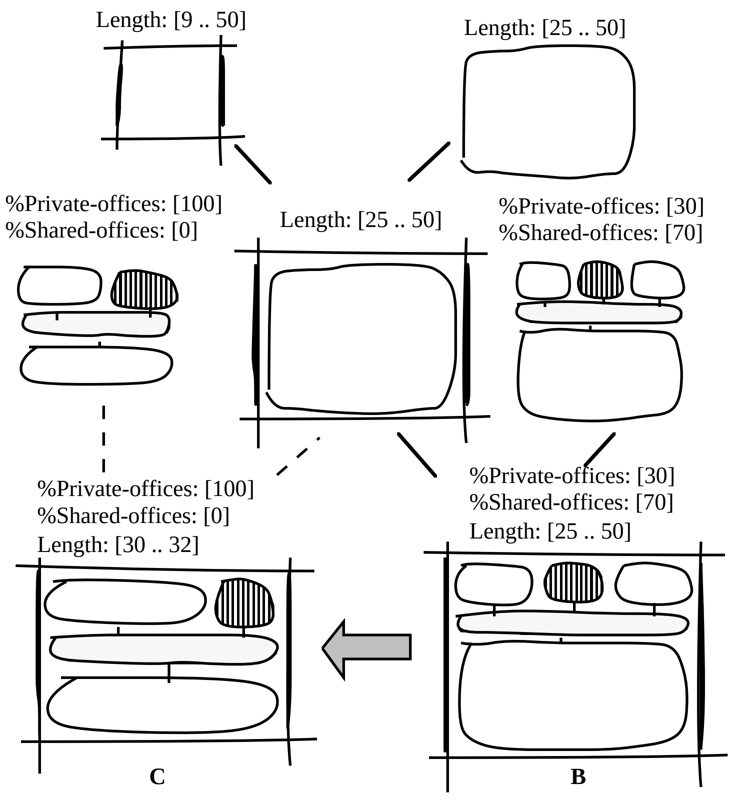

Figure 9. An example of adaptation.

4.2 Exploration

This level requires three modules. The first receives the CSP of the new hypothetical arrangement and produces one or several possible solutions for it in accordance with the user's preferences; the second module verifies that the solution or solutions produced are valid and then produces a proof of failure or success; and the third module searches through this proof for the explanations for failure or success.

4.2.1 Generation

This involves instancing and determining the parameters of all physical elements described by the new arrangement to produce the possible solution or solutions to satisfy the problem's constraints. Propagation and constraint-satisfaction techniques are used here, but as the number of variables can be very high, the architecture of the problem-resolver must be a blackboard type (Hayes-Roth, 1985). The blackboard's levels correspond to the structural decomposition levels of an arrangement separated into sub-arrangements. However, and contrary to the blackboard architecture usually encountered, the levels here are not defined beforehand, *a priori.* The organization of these levels depends on the nature of the adapted (or re-utilized) arrangement. The blackboard is therefore constructed dynamically in terms of the specificities of the problem. Each level can propose hypotheses to the other levels and several solutions can be constructed in a parallel manner.

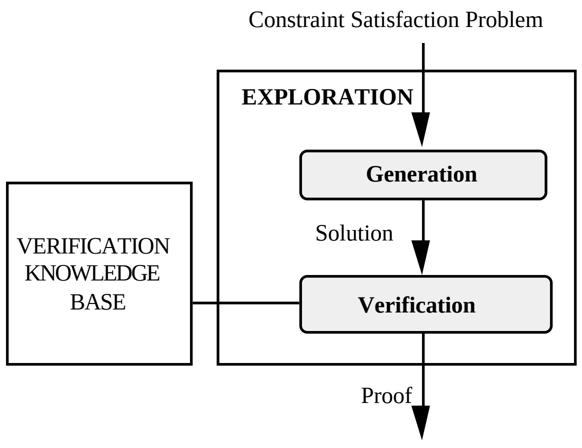

Figure 10. The exploration mechanism.

4.2.2 Verification

It may be that the problem of satisfying constraints described by the new arrangement will be insoluble vis-a-vis the domain's constraints. This stage performs the task of verifying conceivability of the solution's components, and ensuring all regulations and conditions have been met. It does so by activating a database of verification rules. The portion of this database that detects failures caused by the violation of fire-safety regulations, is presented below.

If (core a) accessibility > 30
-> failure

If (core a) finishes (circulation c)
& (circulation c) length (L)
-> (core a) accessibility (L)

If (floor e) length (L)
& (circulation c) equals (floor e)
-> (circulation c) length (L)

As well as activate rules like the one above the module also constructs a proof tree through the activation of these rules. The figure below, for example, demonstrates the proof that the arrangement constructed in the analogy stage does not provide a valid solution.

4.3 Generalization

In its present form, the learning mechanism is analytical (Explanation Based Learning) (Minton et al., 1989), and based solely on the explanations of failure and success determined by the proof produced in the verification stage. Learning therefore involves incremental improvements of *a priori* known concepts: past situations.

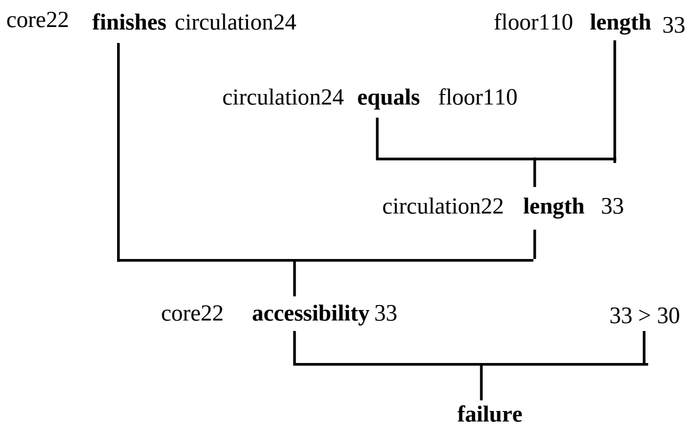

Figure 11. A proof of failure.

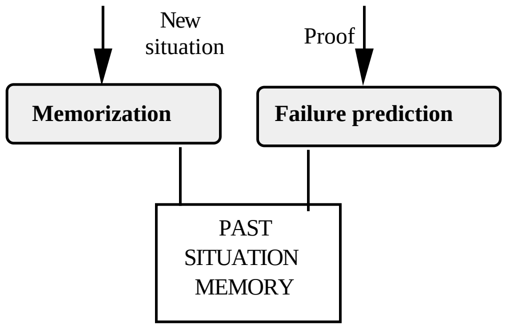

Figure 12. The learning mechanism.

4.3.1 Failure Predictions

If a system decides a solution is incorrect, it then analyzes the proof provided by the verification mechanism to determine the cause or causes of the failure. We use here an algorithm which allows us to retrieve explications from a proof. A detailed description of this algorithm is available in (Dejong, 1986). The figure below demonstrates the explications produced by the module.

Explications:
(core a) finishes (circulation c)
& (circulation c) equals (floor e)
& (floor) length (L)
& (L) > 30

The system uses the explanations to restrict possible ways in which these schemes can be reutilized. In such a way, repetitions of faulty adaptations can be avoided. The performance attributes of schemes where interaction is detected become duly restricted.

Once failure has been predicted, the explications are reused to refine the problem's description and thus commence to construct another CSP at the analogical level. In our current example, the system will no longer be able to make changes to Arrangement B. It will be reutilized, but this time a valid solution will be produced.

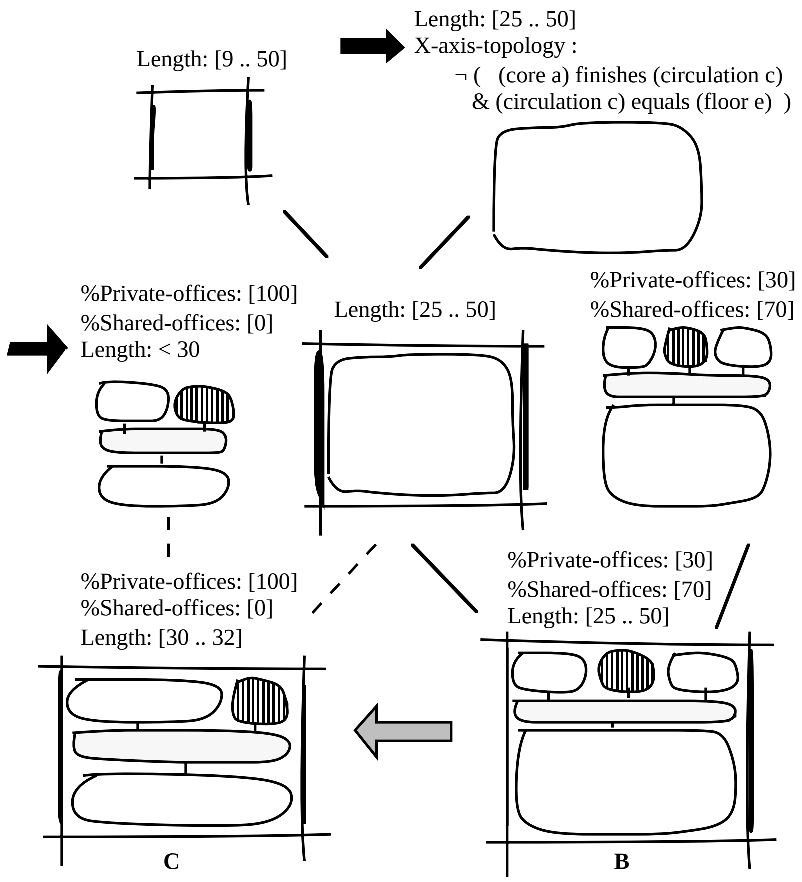

Figure 13. New failure-predicting characteristics.

The figure above, drawn from our current example, demonstrates the prototypes involved and domains modified by the failure-predicting module.

4.3.2 Memorization

If no failure is detected at the verification stage, the system takes advantage of this experience by increasing the number of possible reutilizations of these schemes. The scheme's performance attribute domains, which are reutilized in the construction of an arrangement, are enlarged. More solutions are brought into play.

The figure below depicts the system's memory after successful reutilization of Arrangement B. It is possible that the system will not produce any solution, or that it will produce a solution the user does not validate. In this case, the system's knowledge has proven insufficient and only the user can incorporate new knowledge by providing his own solution, or correcting the solution offered by the system. This is the only way new arrangements can be added to the system's memory.

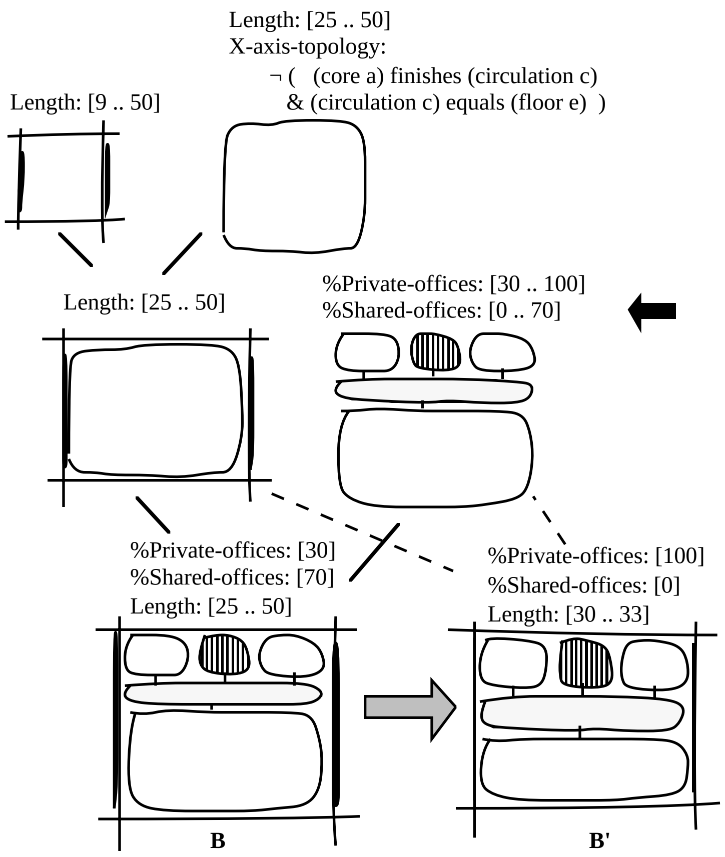

Figure 14. New characteristics for increasing prototype reutilization.

5 Conclusion

These results constitute the first stage in the specification of a system capable of aiding a designer during the preliminary phases of design. Not only will these future systems aid designers in problem solving, the knowledge of these systems, or “assistant designers,” will be improved through contact with their user (Guéna and Zreik, 1989). However, for this type of design activity, in which savoir-faire is implicitly present in the solutions produced by the designer, learning comes about through experience rather than through theoretical teaching. This is why we believe it is necessary to add to an exploration mechanism an analogical reasoning model capable of drawing insight from past situations, and a generalization mechanism that memorizes information gained from past situations. The problem solving mechanism we are proposing selects, adapts, and instantiates abstract prototypes (called schemes). Schemes store problem solving from past situations, and with this forms the basis of a system offering solutions to new problems. Schemes are intermediary knowledge structures that dwell between concepts and solutions. They are dynamic structures, adapting to the problems submitted to the system, and evolve according to the system's behavior, and the designer using the system. Functional and structural decomposition of arrangements into different schemes is used to clarify different levels of generalization and decomposition. The system can thus move from one point of view to another, and from one sub-part to another, in its search for the explanations necessary for the reminding, adaptation, and, if a failure should occur, the repairing of an arrangement.

A system maquette is now being developed with the aid of AIRELLE object language (Nicolle, 1990) and INRIA's Le-LISP v15.2 (Chailloux, 1985). We will soon be able to observe the system's behavior and have a better understanding of its limitations.

References

Brachman,R.J., and Schmolze, J.G., 1985. “An Overview of the KL-ONE Knowledge Representation System,” *Cognitive Sciences* 9, pp. 171-216.

Brown, D.C., and Chandrasekaran, B., 1988. “Expert Systems for a Class of Mechanical Design Activity,” *Expert Systems in Engineering,* Springer-Verlag, 1988.

Chailloux, J., 1985. *Lelisp version 15, le manuel de référence,* Documentation INRIA,

Dejong, G., and Mooney, R., 1986. “Explanation Based Learning: An Alternative View,” *Machine Learning* 1, pp. 145-176.

Flemming, U., 1989. “More on the Representation and Generation of Loosely Packed Arrangements of Rectangles,” *Environment and Planning B: Planning and Design,* 16, pp. 327-359.

Estival, R., 1988. “Le Vocabulaire de la Schématisation,” *Revue de bibliologie: schéma et schématisation.*

Fox, M.S., and Baycan, C., 1987. “An Investigation of Opportunistic Constraint Satisfaction in Space Planning,” in *IJCAI 87,* Milan, p. 1035.

Guéna, F,. and Zreik, K., 1989. “An Architect Assisted Architectural Design System,” in J.S. Gero (ed.), *Artificial Intelligence in Design,* Springer-Verlag, pp. 159-179.

Guéna, F., 1991. *Réutilisation de solutions génériques pour résoudre des problèmes de conception*. Thèse de doctorat de l’université Paris VI.

Guéna, F., 1992. “Représentation et utilisation de pre-solutions pour la résolution des problèmes de conception,” *Journées Représentations par Objets,* La Grande Motte.

Gero, J.S., 1988. “Prototypes: A Basis for Knowledge-Based Design,” in *Knowledge - Based Design in Architecture, TIPS 88,* Preprints, Otaniemi, Finland.

Hayes-Roth, B., 1985. “A Blackboard Architecture for Control,” *Artificial Intelligence* 26(3), pp. 251-321.

Heath, T., 1989. “Social Aspects of Creativity and their Impact on Creativity Modeling,” in *Preprints,* International Round-Table Conference, Modeling Creativity and Knowledge-Based Creative Design, Heron Island, Australia, pp.1-12.

Donikian, S., and Hégron G., 1990. *Towards a Declarative Method for 3-D Scene Sketch Modeling*, Rapport de recherche INRIA-Rennes.

Janssen, P., 1990. *Aide à la conception: une approche basée sur la satisfaction de contraintes*. Thèse de doctorat de l’université de Montpelier.

Kolodner, J.L., 1988. “Retrieving Events from a Case Memory,” in J.L. Kolodner and M. Kaufmann (eds.), *Proceedings*, DARPA Workshop on Case-Based Reasonnings.

Mackworth, A.K., 1987. “Constraint Satisfaction,” in S.C. Shapiro (ed.), *Encyclopedia of Artificial Intelligence,* 1, New York: John Wiley, pp. 205-211.

Minton, S., Carbonnel, J.G., Knoblock, C.A., Kuokka, D.R., Etzioni, O., and Gil, Y., 1989. “Explanation-Based Learning: A Problem Solving Perspective,” *Artificial Intelligence* 40(1-3), pp. 63-118.

Navinchandra, D., 1991. *Exploration and Innovation in Design.* Springer-Verlag.

Nicolle-Adam, A 1990. “Le métalangage AIRELLE,” *Revue d’intelligence artificielle* 4(1), Hermes.

Oxman, R., 1990. “Design Shells: A Formalism for Prototype Refinement in Knowledge Based Design Systems,” *Artificial Intelligence in Engineering* 5(1), pp. 2-8.

Riesbeck, C.K., Schank, R.C., 1989. *Inside Case-based Reasoning*. Lawrence Erlbaum Associates.

Waltz, D.L., 1972. “Understanding Line Drawings of Scenes with Shadows,” in P.H. Winston (ed.), *The Psychology of Computer Vision*, New York: McGraw Hill, 1975.

Zeisel, J., 1981. “Inquiry by Design: Tools for Environmental Behavior, Research, Brooks-Cole,” Monterey, CA.

Zreik, K., 1991. “What Could AI Know About Knowledge Involved in Design Process,” in G. Schmitt (ed.), *CAAD Futures’91*, Vieweg, pp. 395-410.