JS Gero (ed), *Design Computing and Cognition'04,* 3-22.
© 2004 Kluwer Academic Publishers, Dordrecht, Printed in the Netherlands

AN INTELLIGENT ASSISTANT FOR CONCEPTUAL DESIGN

Informed Search Using a Mapping of Abstract Qualities to Physical Form

KIMBERLE KOILE
Massachusetts Institute of Technology, USA

Abstract. In early stages of design, the language used is often very abstract. In architectural design, for example, architects and their clients use experiential terms such as "private" or "open" to describe spaces. The Architect's Collaborator (TAC) is a prototype design assistant that supports iterative design refinement using abstract, experiential terms. TAC explores the space of possible designs in search of solutions satisfying specified abstract goals by employing a strategy we call dependency-directed redesign: It evaluates a design with respect to a set of goals, uses an explanation of the evaluation to guide proposal and refinement of design repair suggestions, then carries out the repair suggestions to create new designs.

1. Introduction

In early stages of design, the language used is often very abstract. Engineers might talk about designing a piece of equipment that is "easy to maintain". Clothing designers talk of "baggy" clothing. Architects and their clients use experiential terms such as "private" and "open". Throughout the design process these abstract terms are operationalized and translated into physical characteristics of the artifact being designed. The design process can be viewed as one of exploration, trying to turn goals, often articulated only in very abstract terms at the beginning of the process, into an artifact that realizes those goals.

If we are to build programs that help designers during the early stages of design, often termed conceptual design, we must give those programs rich vocabularies and the capability to represent and reason with abstract concepts. The hypothesis put forth in this paper is the following: Computational tools can support conceptual design by providing a mapping of abstract terms to measurable design features and by using that mapping in an informed, exploratory search of a design space. The Architect's

Collaborator (TAC) is a prototype design support system that illustrates these ideas in the domain of architecture. TAC employs techniques from artificial intelligence to explore a space of designs using a technique we call *dependency-directed redesign*. TAC is an intelligent design assistant that focuses on design refinement using abstract terms, leaving to the designer the tasks of providing a starting design, specifying and respecifying goals, and ranking designs.

This paper begins by describing a design problem that TAC solved, then discusses how TAC works and gives results from an experiment with a Frank Lloyd Wright Prairie house and from a real-world design example. The paper then discusses related work, future work, and contributions.

2. An Architectural Example

Architectural design is well-suited to research on conceptual design for several reasons. Most design problems exhibit the difficulties mentioned earlier: They are exploratory in nature and involve the use of terms representing abstract, experiential qualities. Such experiential qualities—e.g., openness, spaciousness, privacy—are not easily articulated or formalized. Yet they are an essential part of the architectural design process: Architects and their clients often describe desired spaces in terms of these qualities; architects use their knowledge from past experiences, from environment behavior research, and from their own theories to create physical form that manifests such qualities. This knowledge can be articulated and structured as general design knowledge (e.g., Wright 1954; Alexander et al. 1977; Zeisel and Welch 1981; Hertzberger 1993). As illustrated in this paper, this design knowledge can be operationalized and used as the basis for a conceptual design support system that reasons about abstract qualities and physical form.

To illustrate the above idea, TAC was given the design of an existing house that the owners and their architects were redesigning. Several problems with the house were identified, one of which is illustrated in Figure 1: The living and dining rooms felt small and isolated from each other.

One way to solve the size problem is to make the rooms larger. Another way is to make the rooms feel larger by creating views to neighboring spaces. Creating views also helps with the feeling of isolation. Given a goal of having the dining room not feel small and isolated, TAC used its knowledge base of architectural concepts to translate this goal into having the dining room visually open from the living room. It calculated a visual openness value, Figure 2, and determined that the value was insufficient.

TAC proposed making the rooms feel larger and less isolated by increasing the visual openness of the dining room. It suggested design

Figure 1. A view from living room to dining

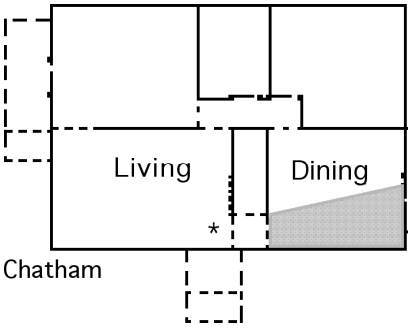

Figure 2. Floor plan showing visual openness of dining from viewpoint *; value is 0.42. Shaded region is visible. Dotted lines are open edges.

modifications to achieve this increase, and created seven new designs by: rotating the stair 90 degrees, rotating the stair 270 degrees, moving the stair to three different exterior edges, removing the stair, replacing the stair wall with a screen. The last solution, with the stair wall "screenified", was implemented by the owners, Figures 3 and 4.

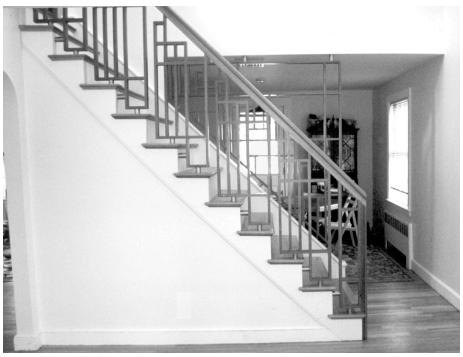

Figure 3. A view from living to dining with screen in place of wall

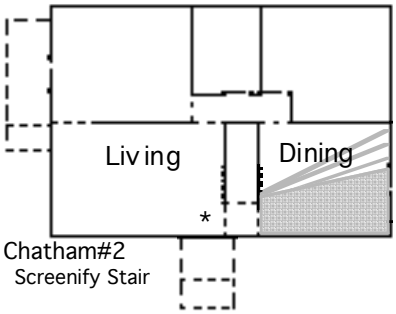

Figure 4. TAC's solution with screen New visual openness value is 0.61.

TAC creates new designs with a visually-open dining room by using a mapping between abstract qualities and operators on physical form, Figure 5. It locates the function that relates visually-open to visual openness, and finds that one territory is visually open to another if at least .6 of its area is visible. Finding this not true, TAC uses a general rule about making an expression of the form "x greater than y" true by increasing y, and proposes increasing visual openness (the value of y in this case). It then finds in its knowledge base techniques for increasing visual openness by modifying the things blocking the view. It determines that the stair blocks the view, then applies each of the techniques to the original design, producing the new designs.

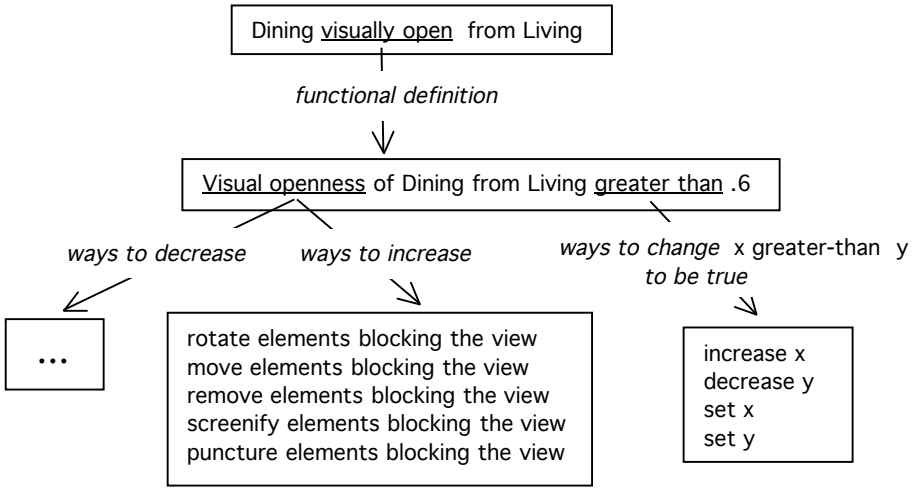

Figure 5. Portion of TAC's mapping of abstract terms to operators on physical form

This example illustrates TAC's behavior: It translates a goal stated in terms of abstract qualities into operators on physical form. It performs this translation using a hierarchy that maps abstract terms to physically measurable design characteristics and design operators for achieving those characteristics. It methodically searches the space of possible solutions by suggesting modifications to the design, pruning suggestions when possible so that it generates only new designs with a good chance of satisfying the desired goal. This informed search of a design space is performed using a technique we call dependency-directed redesign. TAC's intelligence thus derives from two aspects: its hierarchy that maps abstract terms to operators on physical form, and its dependency-directed redesign strategy.

3. Hierarchy: Mapping Abstract Terms to Physical Terms

As mentioned, part of TAC's intelligence derives from its mapping of abstract qualities to details of physical form. TAC knows, for example, that it can make one space more visible from another by removing intervening walls. It knows that it can make a space feel more private by making less of it visible or by making the path to it from a front door less direct. TAC also knows about characteristics of Frank Lloyd Wright's Prairie houses.

TAC represents architectural knowledge—general knowledge as well as a designer's or client's particular preferences—using constructs called *design characteristics*. TAC also contains domain-independent knowledge from geometry, arithmetic, logic, and computation, which it represents using what we call *TAC-functions*. The following sections describe TAC's representation for designs, design characteristics, and TAC-functions, and illustrate how these constructs are used to map abstract qualities to details of physical form.

3.1. REPRESENTING DESIGNS

TAC represents a design as a set of five models, each capturing a different aspect of a design. The *design element model* contains size and location information for walls, windows, etc.; it can be thought of as a primitive computer-aided design (CAD) model. The *edge model* is a two-dimensional geometric abstraction of the design element model, containing points and non-overlapping edges. Edges are either one-dimensional abstractions of design elements (e.g., walls), or one-dimensional projections of design elements. Projections, also called projected edges, are "invisible" edges that extend in a parallel or perpendicular direction from design element edges and help bound two-dimensional regions we call territories (Kincaid 1997). Territories are grouped into a *territory model*, another geometric abstraction of a design element model. (See Figures 2 and 4 for examples.) A *use space model* pairs territories with uses specified by the designer. Finally, a *circulation model* is a graph representing paths between doorways.[1]

TAC's representation for a design differs from most other representations of architectural designs in three significant ways. First, the fundamental vocabulary is that of design elements—walls, windows, etc. Most other knowledge-based architectural design systems that generate new designs represent only spaces, and thus cannot reason about physical form. Second, territories, often called spaces in other systems, are derived from the design elements, not specified independently. Finally, most other systems do not have separate representations for territories and use. A notable exception is (Simoff and Maher 1998). Representing use separately from territories enables TAC to reason about physical form independently of intended use.

3.2. DEFINING DESIGN CHARACTERISTICS

As mentioned above, design characteristics represent architectural properties of a design, including such concepts as visual openness, physical accessibility, and floor plan area. Some design characteristics can be computed directly from design elements, while others are derived from computed design characteristics and are related to physical form via those characteristics. Design characteristics form a decomposition hierarchy, with characteristics computed from physical form at the bottom and those derived from them higher up. In this way experiential qualities are mapped into details of physical form. Four design characteristics, which appear in examples throughout this paper, illustrate this mapping. The decomposition hierarchy for these characteristics is shown at the end of this section.

---

[1] Design elements and their edges are entered by hand using a 2D design editor; projected edges, territories, and circulation paths are computed automatically. Visualization capabilities more sophisticated than 2D floor plans are possible, but are outside the scope of this research, which is focused on intelligent exploration of design space.

Example 1: Visual-openness is quantitative and measures the portion of a territory visible from another territory. Visual-openness is an example of a design characteristic that is computed from physical form elements; its evaluation function is a "black box" computational geometry routine. Figures 2 and 4 show the results of visual-openness calculations.

Example 2: Visually-open is boolean-valued and defined in terms of visual-openness by using a threshold: A territory is considered visually-open from another territory if at least 0.6 of its area is visible from the other territory. Visually-open is an example of a derived design characteristic. Its *evaluation function* is defined in terms of visual-openness using a Lisp-like expression: (gt (visual-openness x from y) 0.6).

Example 3: Perceived-main-entryness is vector-valued and gives a measure of the perception of an exterior door as a main entry. Characteristics that influence a visitor's choice of door when approaching a house are *components* of perceived-main-entryness. These include distance between door and street, straightness of path between door and street, and formality of door. Perceived-main-entryness is a derived characteristic; its evaluation function collects all components into a vector. Perceived-main-entryness also has *necessary conditions*: In order to have a perceived-main-entryness value, for example, an exterior door must be visible from the street.

Example 4: The design characteristic perceived-main-entry is defined in terms of perceived-main-entryness. Its value is the exterior door most likely to be perceived as the main entry. TAC constructs a partial order that ranks exterior doors by their perceived-main-entryness values, and returns the top of the partial order as the value of perceived-main-entry. Alternatively, the evaluation function for perceived-main-entry could combine the components of the perceived-main-entryness vector into a single value and choose the door with the highest value. Notice, however, that the components are incommensurate, and it is not necessarily meaningful, nor obvious how to combine them into a single value.

These examples of design characteristics illustrate TAC's decomposition hierarchy of characteristics. The means by which a design characteristic's evaluation function is defined determines the characteristic's place in the hierarchy. A characteristic that is considered to be directly related to physical form is at the bottom of the hierarchy and has an evaluation function that is a predefined "black box" that operates on one or more models representing a design. A derived design characteristic is higher up in the hierarchy and has an evaluation function constructed using one of three different methods: *evaluation function body* (e.g., as visually-open), *components*, or *components and necessary conditions* (e.g., as perceived-main-entryness). These methods provide the means for constructing the hierarchy, as shown in Figure 6.

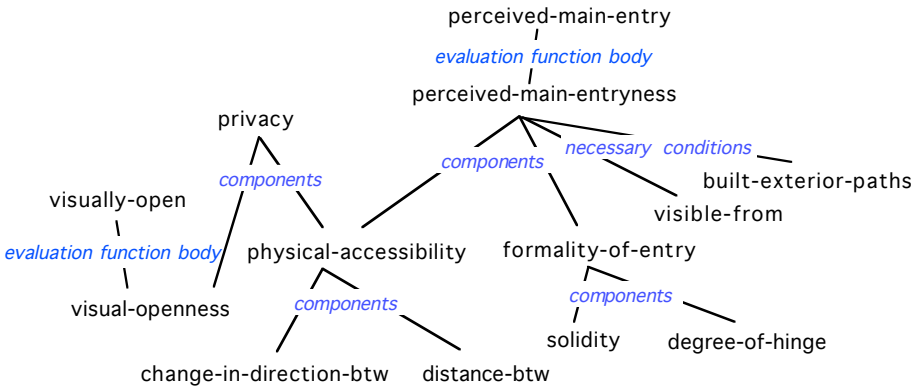

Figure 6. Dependency links for some of TAC's design characteristics

3.3. COMPLETENESS

TAC's knowledge base is complete enough to solve interesting, simple two-dimensional redesign problems, as with the Chatham house design problems described in sections 2 and 5. It contains 30 TAC-functions which represent arithmetic relations, logical relations, computational constructs, and set concepts. These TAC-functions form a basic set of domain independent functions out of which new design characteristics can be built for other architectural design problems. The remaining ten TAC-functions represent geometric concepts, e.g., distance between two things. More geometric concepts could be added, e.g., alignment, overlap (Cui and Randell 1992).

TAC contains 62 design characteristics, which represent architectural concepts such as privacy, visual openness, paths between two design elements. Forty of these proved sufficient for the Chatham design problems. The remaining 22 characteristics were added for a Frank Lloyd Wright Prairie house experiment and included Wright-specific characteristics such as circuitous path, place of prospect, and place of refuge (Hildebrand 1991). More design characteristics could be added easily for design problems involving other architectural types or other architects.

4. Dependency-Directed Redesign

As mentioned earlier, TAC's intelligence derives from its design characteristic hierarchy, which maps abstract concepts to details of physical form, and from its informed search using that hierarchy. Its informed search employs a technique we call *dependency-directed redesign*, which is inspired by artificial intelligence work on dependency-directed backtracking (Stallman and Sussman 1977), plan repair (Sussman 1975, Simmons 1992), and abstraction in search (Sacerdoti 1974). From dependency-directed backtracking, TAC borrows the idea of using an explanation of goal failure to guide search for a solution. From plan repair, TAC borrows the idea of

searching to find an intermediate state in which some goals are satisfied, then repairing that state to satisfy remaining goals. From abstraction in search, TAC borrows the idea of limiting search in a very large solution space by searching in a smaller space: TAC searches in what we call repair suggestion space rather than in design space. Combining these three ideas, TAS uses an explanation to prune repair suggestion space, proposing only those repairs that have a good chance of leading to solutions, and thus decreasing the number and improving the relevance of new designs.

4.1. FROM EXPLANATION TO REPAIR SUGGESTIONS

Dependency-directed redesign uses an explanation of goal failure and a knowledge base of repair strategies to propose suggestions for modifying a design: Given an initial design and a set of goals, TAC evaluates a design with respect to the goals and uses the resulting explanation to propose repair suggestions for any goals not satisfied. It then prunes and refines suggestions, and creates new designs for the remaining suggestions.

Returning to the Chatham house prior to remodeling (Figures 1 and 2), consider the goal of having the dining room visually open from the living room. The goal is represented by the expression (visually-open Dining from Living). TAC evaluates this goal, determines that it is not satisfied, and produces an explanation of the failure in the form of a tree that represents a trace of the goal expression's evaluation, Figure 7. By walking down the tree, TAC can determine why a goal was not satisfied and then use that information to propose suggestions for design repair.

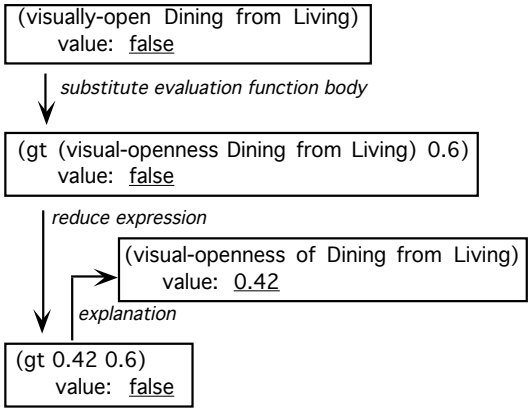

Figure 7. Explanation for (visually-open Dining from Living) for Chatham example

In particular, TAC identifies opportunities to repair the cause of failure by looking for expressions whose value it knows how to change via domain independent routines called *fixers*. Fixers reason about how to get from a current value to a desired value; they propose increasing, decreasing, or setting values. They rely on a characteristic's *increasers*, *decreasers*, and

setters—expressions that when evaluated modify a design, thereby changing the value of the characteristic. (Examples of fixers and increasers were shown in Figure 5, labeled as "ways to change …" and "ways to increase", respectively.) Whether a design characteristic or TAC-function has a fixer, increasers, decreasers, or setters depends on the nature of the characteristic or function. Some characteristics, such as the color of a design element, are directly settable and have setters that change a value. Others, such as visual-openness, are not directly settable, and hence do not have setters; instead their values are changed by modifying the design. Thus, instead of setters, the characteristic visual-openness has increasers, since certain modification operators, e.g., removing an intervening design element, have a good chance of increasing its value.

To repair the Chatham design so that the dining room is visually open from the living room, TAC traverses the explanation shown in Figure 7 until it finds methods for "fixing" a node's expression. When it gets to the gt node, it finds that it knows how to fix a (gt x y) expression: it can set y to be greater than x, decrease y to be less than x, set x to be greater than y, or increase x to be greater than y. In the current expression, y is a constant, 0.6., and cannot be decreased or set; x is the visual-openness expression, and visual-openness cannot be directly set. One option remains: increasing x. TAC checks its knowledge base and finds that it knows how to increase the value of visual-openness by means of increasers associated with that characteristic. So it proposes increasing the value of visual-openness:

(increase-value of (visual-openness Dining from Living)
until visual-openness greater than 0.6)

TAC then retrieves increasers (see Figure 5), which are written in terms of operators on design elements that block the view between things, e.g.,

(remove blocking-elts-btw x y) (screenify blocking-elts-btw x y)

Substituting arguments of Dining and Living from the original goal expression, TAC then proposes specific repair suggestions, e.g.,

(remove blocking-elts-btw Dining Living) (screenify blocking-elts-btw Dining Living)

TAC now checks the design to identify design elements that block the view, finds the stair, substitutes it into the repair expressions, and proposes, e.g.,

(remove Stair) (screenify Stair)

For each suggestion, TAC then creates new designs, one of which was shown in Figure 4.

4.2. FROM REPAIR SUGGESTIONS FOR GOALS TO NEW DESIGNS

The Chatham house example illustrates how TAC works with a single goal, translating a goal expression into operators on physical form, carrying out those operators to create new designs. More realistic design problems have multiple, often conflicting goals. TAC deals with this situation by using a

generate-and-test control structure, generating intermediate designs that satisfy a subset of the goals, then iteratively repairing those designs to satisfy remaining goals. An enabling assumption for this approach is that some goals will be independent, so that working on one goal does not always undo a previously satisfied goal (Sussman 1975). For the goals that do interact, some amount of work to reevaluate and resatisfy goals is necessary. TAC limits the amount of work in two ways. First, as previously described, it separates the proposing of repairs from the performing of repairs, thereby enabling it to avoid designs that it knows will not satisfy goals. Second, its generate-and-test control structure includes a lookahead step: When proposing repair suggestions for a particular goal, it "looks ahead" for potential goal interactions. It looks both for conflict, i.e., when satisfying a goal will undo an already satisfied goal, and synergy, i.e., when a modification will satisfy more than one goal. Three kinds of conflict and synergy were identified: obvious, predictable, unpredictable. TAC handles the first two of these: Obvious interactions are detected by comparing goals, predictable interactions are detected by comparing repair suggestions for goals. Being able to reason about obvious and predictable interactions enables pruning of repair suggestions before creating designs, which helps control search and increases the chances that intermediate designs are closer to solutions. Unpredictable interactions, by definition, cannot be detected ahead of time, and lead to the need for generate-and-test.

An example of TAC's reasoning with multiple goals is shown for one of Frank Lloyd Wright's Prairie houses, the Horner house (Figure 8).

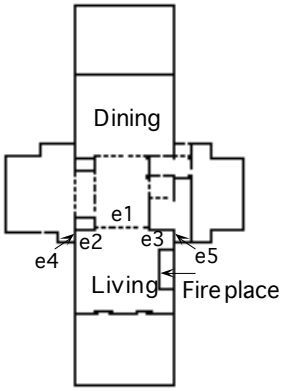

Figure 8. Edge model for Horner house; e1 to e5 are edges

TAC was asked to evaluate the house with respect to five goals usually satisfied in Prairie houses: the center of the Living room visible from the Dining room, the Living room visually open from the Dining room, one fireplace in the Living room and one in the entire design, and the fireplace on an interior edge. The first four goals are satisfied already, but the last goal is not: the fireplace is not on an interior edge. Attempting to satisfy this

goal illustrates some interesting goal interactions.

Figure 9 illustrates TAC's behavior given the five goals. Starting with the unsatisfied goal, TAC proposes six suggestions (s1 to s6): move the fireplace to any of five interior edges (e1 to e5) or add a new fireplace on an interior edge. It notices that adding a fireplace conflicts with the goal of having one fireplace, so it prunes that suggestion. It then creates five new designs, each with a fireplace on one of the specified edges. It checks these designs and finds that in D4 and D5 the fireplace is not entirely on an interior edge, so it discards these two designs. It rechecks the other four goals for the remaining designs, finding that D2 and D3 are solutions. It determines that moving the fireplace to e1 (in design D1) has caused the visibility goals to become unsatisfied: the fireplace has blocked the view between living and dining territories. So TAC proposes removing or puncturing[2] the fireplace. It notices that removing the fireplace will conflict with keeping the number of fireplaces at one, so it prunes that suggestion. It carries out the fireplace puncture operation and creates design D6. It checks the visually-open goal and finds it now satisfied, so D6 is a solution. It has no more designs to check, so it stops, returning solutions D2, D3, and D6 (Horner#2, Horner#3, and Horner#1#1 in Figure 10).

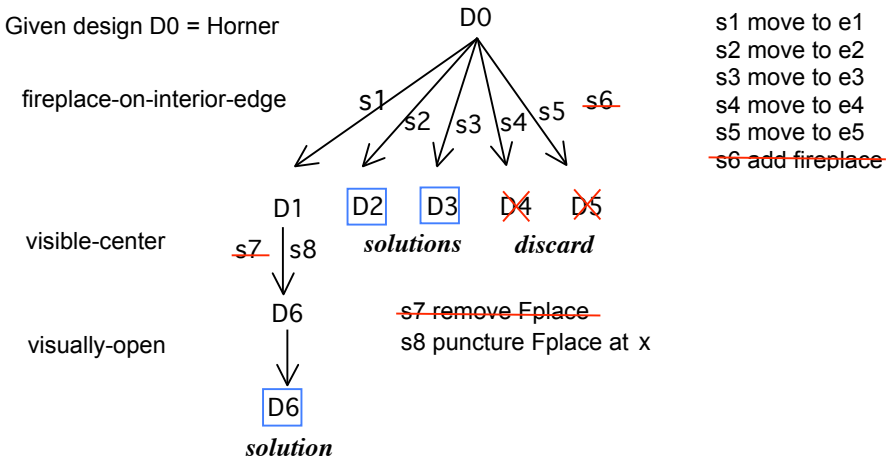

Figure 9. Control structure for Horner design example

The goals in this example exhibited several different kinds of interaction. Obvious synergy was exhibited by the two fireplace count goals: having one fireplace in the living territory also satisfied having one fireplace in the entire design. Predictable conflict occurred between the goal of having one fireplace and a suggestion to remove the fireplace. Note that the goals themselves in this case were not in conflict, but rather one goal was in conflict with a particular repair suggestion proposed for another goal.

[2] "Puncturing" a fireplace is a technique Wright used in the Robie house.

Unpredictable synergy occurred between the visible-center goal and the visually-open goal: puncturing the fireplace to make the living territory center visible also caused the living territory to be visually open.

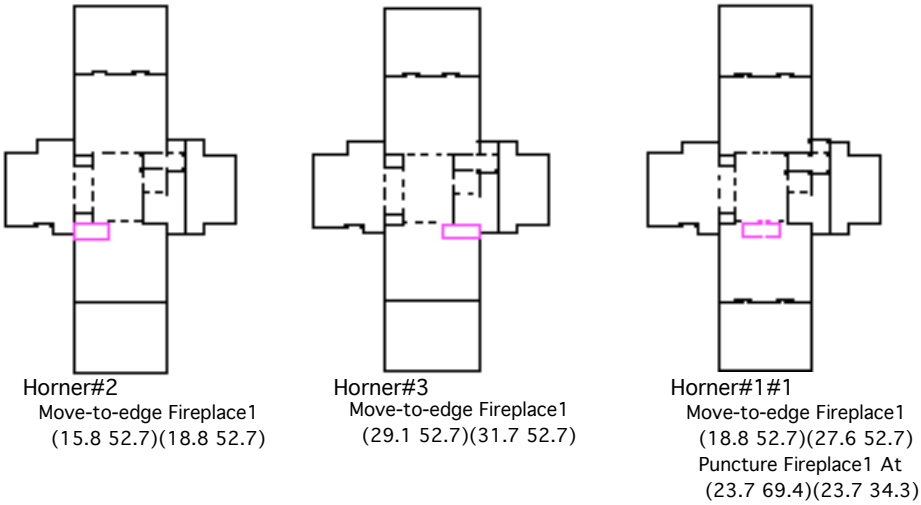

Figure 10. Solutions for Horner design problem

4.3. EFFECTIVENESS

Removing, puncturing, and adding design elements are examples of TAC's design modification operators. TAC contains 23 such operators, 13 of which form a basic set applicable to a wide range of design problems. Ten others are more specialized, e.g., adding built exterior paths. None of TAC's current design modification operators change the footprint of a design; more design modification operators could be added that do.

Preliminary experiments showed that TAC's dependency-directed redesign strategy proved effective: Its two techniques for performing informed search—using an explanation to guide search in repair suggestion space, and pruning and consolidating repair suggestions using a lookahead method that identifies conflict and synergy—significantly reduced search in a large design space. Without using explanation or lookahead, TAC would have generated approximately $4x10^8$ designs for the Horner design problem described in this section: 23 operators, 10 producing at least 4 new designs each, yields 53 new designs for each of 5 goals, or 53^5, approximately $4x10^8$.

The tables below summarize control structure experiments for the Horner design problem. Five goals were specified in each of two orders, optimal and nonoptimal.[3] Table 1 gives results using explanation to guide search; Table 2 gives results using both explanation and lookahead.

[3] An optimal goal order is one in which goals with synergistic operators, i.e., that will satisfy more than one goal, precede goals with which they interact. See (Koile 2001) for details.

TABLE 1. Five goals, Horner design problem, explanation used, no lookahead

Goal Order	# solutions	# designs	# repair cycles
optimal	4	16	5
nonoptimal	37	339	48

TABLE 2. Five goals, Horner design problem, explanation and lookahead used

Goal Order	# solutions	# designs	# repair cycles
optimal	4	8	3
nonoptimal	11	47	5

Using an explanation to guide search reduced the number of designs generated to 339 for a nonoptimal goal order, and to 16 for an optimal goal order, which is considerably better than $4x10^8$. Adding the lookahead mechanism further reduced the number of designs generated for nonoptimal goal order to 47, and for optimal goal order to 8.

The optimal goal order, both with and without lookahead, resulted in the same four solutions. The nonoptimal goal order, however, resulted in additional solutions. Most of these solutions were very similar to those found with optimal order. (They might have punctured the fireplace in a slightly different location, for example.) Several of the solutions found without lookahead, however, were significantly different, because designs were created that violated goals that were not the current focus—a situation not uncommon in search problems. TAC then repaired those designs, creating additional solutions. For this reason, the best control structure for generating solutions when goals interact would include an option for relaxing lookahead when desired.

5. A Real-World Design Problem

A system such as TAC can be used by architects as both a design tool and an analysis tool. This section illustrates TAC's utility as a design tool in an experiment using the Chatham house discussed in the opening example.[4]

The Chatham house was being redesigned at the same time that TAC was under development. The architects and TAC thus were able to work in tandem on the same design problems. TAC was given a model of the house and a set of design goals defined by the owners and their architects, and in response proposed new designs. Several of the designs are presented here to show that TAC finds plausible solutions to a real architectural design problem, and that is does so with breadth and generality.

The Chatham house, floor plan, and approach paths are shown below.

[4] See (Koile 2001) for discussion of TAC's utility in analysing designs and definitions of architectural type.

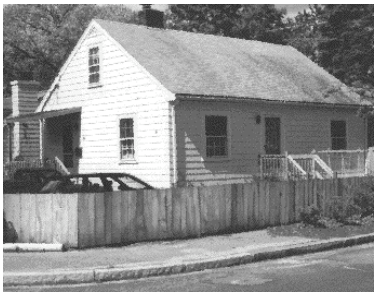

Figure 11. The Chatham house

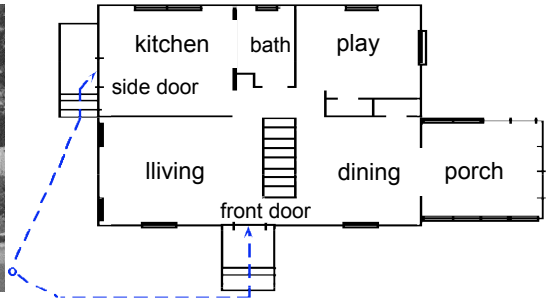

Figure 12. Chatham house first floor and approach paths to exterior doors
The usual approach point is marked by o.

Four problems with the house were identified:

- site: visitors approaching the house are not sure which door to use
- entry: living room is not private with respect to the front door
- territories: main living spaces feel isolated from one another
- use: kitchen activity is too far from the dining activity

We phrased goals for TAC in terms of physical access and visual openness:

- site: one perceived main entry
- entry: living room visually semi-open and physically semi-accessible (i.e., reached via somewhat crooked path) from the perceived main entry
- territories: main living spaces visually open from one another
- use: kitchen activity next to the dining activity

Figure 13 shows one of the designs produced by the architects, along with a similar design proposed by TAC. In both designs, TAC and the architects solved the problem of having more than one perceived main entry by removing the front door and making the side door the new front door. Moving the front door also increased the change in direction, and thus the crookedness of the path, between the entrance and the living territory, and decreased the visibility of the living territory from the entrance. The visibility was decreased too much, so both TAC and the architects removed a section of wall between the front door and the living territory, a modification that also makes the living territory more easily accessible from the entrance. TAC and the architects turned the stair to increase visual openness between the dining and living territories. They exchanged the playroom and kitchen activities so that the kitchen activity would be adjacent to the dining activity.

The designs also show differences, some of which result from the architects' working with a larger goal set than TAC. Some of these goals were not given to TAC because they would not have illustrated new TAC behavior, e.g., making the kitchen territory more visually open from the dining territory. Other goals were outside the scope of TAC's current operators, which do not change a design's footprint, e.g., enlarging the entry porch. Other differences between TAC's designs and the architects' are due

to both unspecified goals and lack of information in TAC's knowledge base. When the dining territory became smaller as a result of turning the stair, for example, TAC did not enlarge the territory at the expense of the porch, as the architects did: TAC did not know of an implicit assumption that the dining territory would not be smaller, nor that a territory can be enlarged by borrowing area from a neighboring territory.

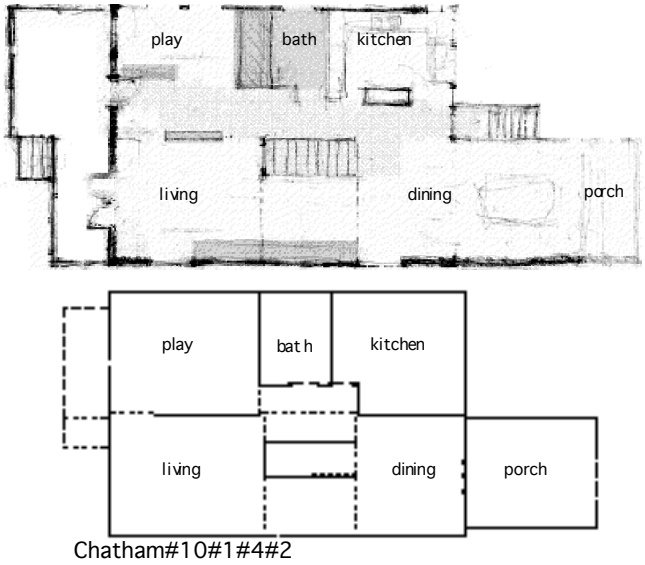

Figure 13. Architects' design (top) and TAC's design; labels are activities

An alternate design produced by the architects and a similar design proposed by TAC are shown in Figure 14.

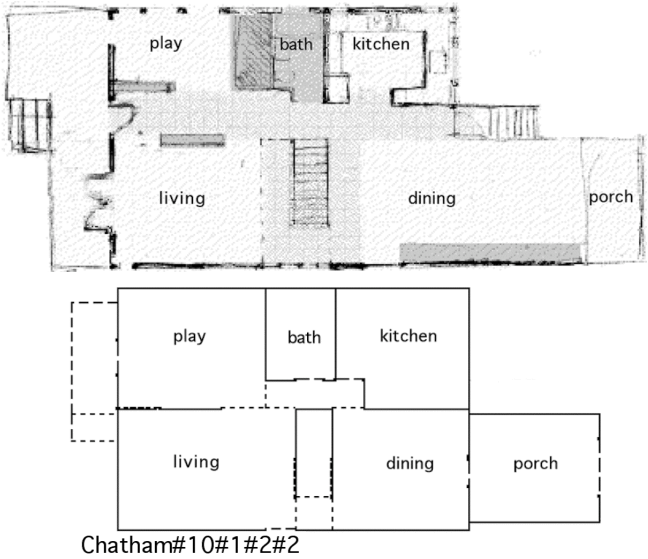

Figure 14. Alternate design by the architects (top) and TAC's similar design

In the designs shown in figure 14, TAC and the architects again have made the side door the new front door, removed a section of wall between the entrance and the living territory, and exchanged playroom and kitchen activities. Instead of turning the stair, however, they have replaced the solid wall of the stair with a screen, e.g., as shown in Figure 3.

TAC came up with designs that differed significantly from the architects' designs. TAC's design in Figure 15 uses a screen the full width of the living territory as a means of increasing privacy by decreasing visibility from the front door. Adding the screen satisfied the goal, but violated an implicit goal of creating only useful-sized territories.

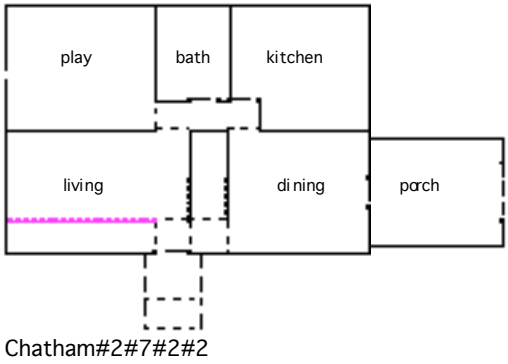

Figure 15. TAC's design with full width screen

TAC also was not told that the owners preferred the stair in a central location, so it suggested moving the stair to exterior walls, creating plausible designs but not what the owners had in mind, Figure 16. TAC was not told that the architects and owners desired that the house be connected to a neighborhood, nor given information about the neighborhood. As a result, TAC did not know that the side door makes a better main entry because the street on that side of the house is less busy and the houses closer together. It thus produced designs with the front door as the main entry and the side door removed, Figure 17.

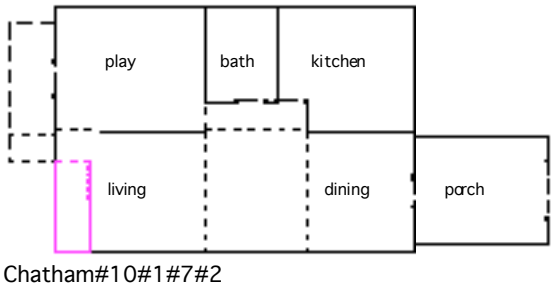

Figure 16. TAC design: stair on exterior edge

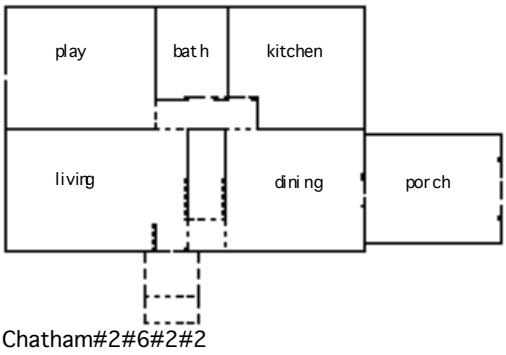

Figure 17. TAC design: screen at front door

Some of TAC's novel designs are quite plausible and result from its ability as a computational tool to carry out transformations easily and quickly: It produces many variations on a theme, a task an architect would find very tedious. In some cases, TAC's designs may be redundant or even bad, as in Figure 15, but they can be easily set aside by the designer as she focuses on the designs that meet specified and unspecified criteria. TAC has utility as a brainstorming tool and can help a designer and client elucidate goals by calling attention to desired or undesired features.

6. Related Work

There is a vast literature on computational tools for conceptual design. Tools that are most similar to the work reported in this paper either reason with similar experiential knowledge or employ similar reasoning techniques. An earlier paper, (Koile 1997), surveyed systems that evaluated designs with respect to experiential qualities. The discussion here is confined to work that shares features with TAC's dependency-directed redesign strategy, especially in the field of architecture.

Two methodologies that share features with TAC's redesign strategy are case adaptation in case-based reasoning, and performance-based refinement.

Case-based reasoning: Case adaptation methods employed in case-based reasoning systems are similar in spirit to TAC's repair mechanism. Given a design case, they modify it to meet specified design goals. Indeed, TAC's modification operators can be thought of as a "taxonomy for design adaptation" (Oxman 1996). Several case adaptation systems are mentioned here. (See Voss and Oxman (1996) for a survey.)

Constraint satisfaction techniques have been used to adapt architectural design cases. Some systems first adapt a case's topology using graph algorithms, then adapt geometry using constraint satisfaction techniques (e.g., Smith et. al. 1996, Hua et. al. 1996). Design knowledge may be represented implicitly in the systems' parameters and constraints (e.g., Smith

et. al. 1996), or explicitly using techniques such as hierarchies of object types (e.g., Giretti and Spalazzi 1997). Constraint satisfaction techniques are not appropriate for TAC's repair problem: Since particular design element arrangements for realizing abstract design characteristics are not known *a priori*, specific constraints between design elements cannot be specified.

Model-based reasoning techniques have been used to adapt cases, though typically for engineering fields in which qualitative models of device behavior can be built. Even though not in the domain of architecture, the systems described in Goel (1991) and Prabhakar and Goel (1998) are worthy of mention as examples of using explanation of failure (case mismatch) to guide iterative repair. The systems retrieve a mechanical design case, and evaluate the case using simulation via a causal model of the device's behavior. They then propose modifications by identifying the source of the device failure and selecting repair strategies. Model-based reasoning is not possible for TAC's task because the global effects of modification operators on abstract design characteristics cannot be predicted.

Performance-based refinement . The term "performance-based refinement" has been used in the computer-aided architectural design community to mean using desired values of performance variables (akin to TAC's design characteristics) to guide design refinement—just what TAC does. As Flemming and Mahdavi (1993) suggest, most performance-based refinement tools only evaluate performance variables; the designer must "guess" at likely design modifications for affecting desired values. The work of Mahdavi (1997, 1998) is an exception. GESTALT, described in Mahdavi (1997), employs an "intelligent" generate-and-test method to iteratively modify a design using knowledge of functional relationships between physical form and performance variables. It maps experiential qualities, such as light quality, to methods for changing them, as TAC does. Such qualities in GESTALT are quantitative (e.g., a five point scale of light quality) and can be mathematically modeled or formalized through regression analysis. Hence, optimization techniques can be used to select particular values for desired characteristics. TAC's power would be enhanced by employing this technique when possible, rather than always assuming monotonic relationships. Many design characteristics in architectural design are not quantitative, however, so TAC's qualitative reasoning cannot be replaced completely with optimization methods.

7. Future Work and Contributions

TAC's representations form a good foundation for the development of rich knowledge bases of architectural design knowledge. As with all systems that rely on knowledge bases, however, acquiring the knowledge is nontrivial. If a designer can assemble a set of designs that exhibit a

particular characteristic, machine learning techniques may able to help with the knowledge acquisition task. In addition, explanation-based learning techniques may be useful in adding knowledge of discovered synergies and conflicts to the knowledge base.

TAC's knowledge base could be extended to include knowledge of materials and light, both of which affect the experiential qualities of a space. The knowledge base also could be extended to include sociological influences on physical form (e.g., Wright 1954, Hillier and Hanson 1984). Changing attitudes about domestic life, for example, transformed the front and back parlors of Victorian times into the modern-day living room.

Focus to date has been on TAC's representation and reasoning capabilities, with little time spent addressing user interface issues. Thinking about what constitutes an appropriate interface for designers opens up a number of intriguing possibilities. TAC would benefit, for example, from integration with a sketching tool, e.g., (Gross 1996), so that a designer could move between sketching and TAC's evaluation and repair steps. TAC might also benefit from an interface that allowed a user to increase or decrease values of design characteristics and observe the resulting changes in physical form. A similar idea is proposed in (Flemming and Mahdavi 1993).

TAC's control structure could be extended to support goal specification and refinement. As design goals evolve along with a design solution, a designer might want to interrupt one of TAC's evaluation and repair cycles, redefine goals, then have TAC continue. This extension would be straightforward. TAC also could be extended to assist a designer in specifying goals by suggesting some goals automatically. If a design has a second floor, for example, TAC could suggest that the design needs a stair. The issue of how complete the goal set needs to be and whether goals could be inferred are open research questions.

TAC demonstrates that it is possible to construct a prototype intelligent assistant that supports conceptual design via iterative design refinement, representing and reasoning about how experiential qualities are manifested in physical form. Its hierarchy of design characteristics provides a means for operationalizing abstract qualities. Its dependency-directed redesign mechanism provides a means for exploring a design space using abstract qualities. Its use in finding plausible solutions to a real architectural design problem demonstrates the real-world potential of these ideas.

Acknowledgments

This research was funded by a National Science Foundation Graduate Fellowship. The author thanks Randall Davis, Howard Shrobe, Patrick Winston, Tomás Lozano-Peréz, and John Aspinall for assistance in AI; and Aaron Fleisher, Richard Krauss, Duncan Kincaid, and Mark Gross for assistance in architecture. The drawings in Figures 13 and 14 are courtesy of Duncan Kincaid and Daniel Gorini.

References

Alexander, C, Ishikawa, S, Silverstein, M, Jacobsen, M, Fiksdahl-King, I and Angel, S: 1977, *A Pattern Language*, Oxford University Press, New York.

Cui, Z and Randell, D: 1992, Qualitative simulation based on a logical formalism of space and time, *AAAI '92,* pp. 679-684.

Flemming, U and Mahdavi, A: 1993, Simultaneous form generation and performance evaluation: A `two-way' inference approach, *in* U Flemming and S Van Wyk (eds), *CAAD Futures '93,* North-Holland, 161-174.

Giretti, A and Spalazzi, L: 1997, ASA: A conceptual design-support system, *Engineering Applications of Artificial Intelligence* **10**(1): 99-111.

Goel, AK: 1991, A model-based approach to case adaptation, *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum, pp. 143-148.

Gross, MD: 1996, The electronic cocktail napkin--a computational environment for working with design diagrams, *Design Studies* **17**: 53-69.

Hertzberger, H: 1993, *Lessons for Students in Architecture*, Uitgeverij Publishers, Rotterdam.

Hillier, B and Hanson, J: 1984, *The Social Logic of Space*, Cambridge University Press.

Hildebrand, G: 1991, *The Wright Space: Pattern and Meaning in Frank Lloyd Wright's Houses*, University of Washington Press, Seattle.

Hua, K, Faltings, B and Smith, I: 1996, CADRE: Case-based geometric design, *Artificial Intelligence in Engineering* **10**: 171-183.

Kincaid, DS: 1997, *An Arithmetical Model of Spatial Definition*, Master of Architecture Thesis, Dept. of Department of Architecture, Massachusetts Institute of Technology.

Koile, K: 1997, Design conversations with your computer: evaluating experiential qualities of physical form, *in* R Junge (ed), *CAAD Futures '97*, 203-218.

Koile, K: 2001, *The Architect's Collaborator: Toward Intelligent Tools for Conceptual Design*, PhD Thesis, Dept. of EECS, MIT.

Mahdavi, A and Suter, G: 1997, On implementing a computational facade design support tool, *Environment and Planning B* **24**: 493-508.

Mahdavi, A and Suter, G: 1998, On the implications of design process views for the development of computational design support tools, **7**: 189-204.

Oxman, R: 1996, Design by re-representation: A model of visual reasoning in design, *Design Studies* **18**(4): 329-347.

Prabhakar, S and Goel, AK: 1998, Functional modeling for enabling adaptive design of devices for new environments, *Artificial Intelligence in Engineering* **12**: 417-444.

Saderdoti, E: 1974, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* **5**(2): 115-135.

Simmons, RG: 1992, The roles of associational and causal reasoning in problem solving, *Artificial Intelligence* **53**(2-3): 159-208.

Simoff, SJ and Maher, ML: 1998, Designing with the activity/space ontology, *in* JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, 23-43.

Smith, I, Stalker, R and Lottaz, C: 1996, Creating design objects from cases for interactive spatial composition, *in* JS Gero (ed), *Artificial Intelligence in Design '96*, Kluwer, 97-116.

Stallman, R and Sussman, G: 1977, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence* **9**: 135-196.

Sussman, GJ: 1975, *A Computer Model of Skill Acquisition*, American Elsevier, New York.

Voss, A and Oxman, R: 1996, A study of case adaptation systems, *in* JS Gero (ed), *Artificial Intelligence in Design '96*, Kluwer, 173-189.

Wright, FL: 1954, *The Natural House*, Horizon Press, New York.

Zeisel, J and Welch, P: 1981, *Housing Designed for Families: A Summary of Research,* Joint Center for Urban Studies of MIT and Harvard University, Cambridge, MA.