A Prototype System for Design Automation via the Browsing Paradigm

Sandeep Kochhar
Harvard University

Abstract

Design activity is often characterized by a search in which the designer examines various alternatives at several stages during the design process. Current computer-aided design (CAD) systems, however, provide very little support for this exploratory aspect of design. Two major problems exist: rapid generation of design alternatives, and presentation of these alternatives to the user in an accessible manner.

This paper provides the foundations for a novel CAD technology intended to encourage a synergistic, cooperative relationship between the computer and the human designer: at any stage in the design process, the user can request that the system generate *automatically* design alternatives that satisfy certain constraints. These alternatives are structured by the system in a spatial framework using properties specified by the user as independent dimensions. The user can systematically *browse* through the design alternatives via graphical gestures such as scrolling and pointing, and select any of these designs for further development.

A prototype CAD system—FLATS—supporting this cooperative design paradigm has been constructed; it consists of three major modules: the *modeling system*, the *grammar-directed constrained generator*, and the *browsing system*. This paper describes the algorithms and data representations used by these modules, and demonstrates the use of the system to design small architectural floor plans.

A videotape demonstrating the prototype system accompanies this paper.

Keywords: Graphical user interfaces, spatial data management, direct manipulation, design automation, human factors, human-machine interaction, constraint-based design, grammar-directed design.

1 Introduction

At present, computer aided design (CAD) technology is based on two major paradigms:

- *Drafting Assistant:* Many widely available CAD systems are not really design assistants as much as drafting aids in that they rely on the user having in mind the goal or final product. While the better systems offer various orientation and bookkeeping aids—for example, design trees, histories and choice points—all decisions are made by the user.

- *Constrained Search:* In some systems being developed by researchers, the user typically creates a starting design, specifies a set of constraints on the goal design, and then lets the CAD system "go." The system employs its internal model describing designs to generate alternatives (often exhaustively) using the constraints to prune fruitless paths and to guide the search. At the end of this process, the user is presented with a set of design alternatives satisfying the constraints. Unfortunately, the set may be very large (often thousands of designs), and may be presented without any obvious structure to the output showing relations between the alternatives.

This paper explores the foundation for a very different CAD paradigm—that of cooperative design between the user and the computer. The user creates a partial design, lets the system generate alternatives subject to a "language" of designs and given constraints, explores the alternatives by structuring them in a spatial framework (using graphical gestures to move through the designs), picks some design, modifies it, lets the system suggest more alternatives, and so on. I believe that this paradigm offers great potential for increasing productivity by letting designers concentrate on making critical and creative choices, and letting the system take care of the details of elaborating designs.

Before presenting the details of this paradigm, I recapitulate briefly the motivation for this work. Humans rarely design in a linear fashion from start to finish. A person typically creates a partial object, examines a few elaborations of that object, selects a few alternatives, examines their elaborations, and so on, until he or she produces (one or more) designs that have certain desired characteristics. (Many of these, while satisfying the same rigid constraints, may differ only in some unquantifiable attributes.) Thus, exploring alternatives is an integral part of human design activity. As mentioned earlier, a major weakness of current CAD systems is the lack of support for this exploratory nature of the modeling task: these systems do not support a systematic search through all possible design alternatives. Thus, the user is limited to manually exploring a few choices—a slow and demanding task, potentially missing many desirable options.

Author's address: 112 Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138, USA; (617) 495-3998; kochhar@harvard.harvard.edu

Research Issues

This paper focuses on two major research questions:

- How do we generate automatically design alternatives that satisfy the user's requirements, and,

- How do we present design alternatives to the user so that he or she can easily perceive the relationships between those designs?

The paper also examines the major components of an existing, experimental CAD system supporting the technology described above: 1) a *modeling system*, which allows the user to create and manipulate designs; 2) a *constrained generator*, which allows the user to request that the system generate alternatives subject to certain constraints; and, 3) a *browsing system*, which lets the user explore the alternatives (produced by the generator) in a systematic way.

The rest of this paper is organized as follows. In Section 2, I discuss related work of other researchers. In Section 3, I describe essential features of the user interface of a CAD system that supports the cooperative design paradigm. The prototype system is described in Section 4, including the theoretical underpinnings of the paradigm, the representation scheme for designs and the algorithms needed for automating design. An example of using the system to design small architectural floor plans is also presented in Section 4. Finally, in Section 5, I conclude with the future direction of this research.

2 Related Work

In this section, I discuss some of the relevant work of researchers in the areas of human factors, spatial data management, automatic generation of building layouts, and graph grammars in solid modeling. Many of the principles expounded by these researchers have been used in the design of the user interface described in Section 3 and the algorithms described in Section 4.

2.1 Graphical User Interfaces

Schneiderman [Schn82] first introduced *direct manipulation* as a set of guidelines for the design of user interfaces. In this paradigm, the user manipulates a continuous representation of an object via operations using physical gestures or command buttons; the user is provided immediate feedback showing the effect of these operations on the object. Schneiderman also pointed out that such systems are easily used by both novices and experts, and that users are able to achieve their goal more rapidly because of the feedback. Thus, the users can "understand" the system better.

The successful application of these principles to the technique of spatial data management—organizing, accessing and manipulating information via its graphical representation—was demonstrated by researchers at MIT [Done78] and CCA [Hero80]. A spatial data management system (SDMS) organizes information in a spatial frameworks and allows users to access it in a natural manner by employing basic perceptual skills, such as vision, sound and touch. Donelson [Done78] argues that such systems are easy to use since people understand spatial relations naturally.

The prototype SDMS developed at CCA [Hero80] can be used as an interface to large, shared databases. The information is laid out in a *graphical data space (GDS)* consisting of a set of *data surfaces* used for displaying the data. The user sees portions of this space at different levels via graphics displays that act as windows into the data surface. For example, the *world view monitor* displays an entire data surface, with a highlighted "you are here" marker; the details of the highlighted region are shown on another monitor. Various navigational aids are provided to let the user move about in the GDS. For example, a joystick can be used to move along any direction on the data surface (moving perpendicular to the surface corresponds to zooming); the user can also rapidly move to a different part of the data surface by just touching the appropriate position in the world view.

The success of an SDMS depends on the graphical representations employed and on the controls provided to make the movement in the system comprehensive and natural to the users. The prototype SDMS described above was extended in the VIEW system by Friedell *et al.* [Frie82] to tailor the graphical presentations of information to the user's task, identity and database query. The VIEW system stores a "conceptual model" of the data contained in the database. This model includes information about how to present the information and how to display it graphically. Information is represented through icons arranged in two-dimensional *information spaces* or *I-spaces*, which are generated dynamically. The VIEW system also attempts to display the I-spaces in a structured manner by using a description of the database query (and its responses) to formulate a layout plan that suits the type and amount of data. For example, to display information in response to a query against a database of ships, the icons might be arranged according to nationality.

Reiner *et al.* [Rein84] developed the Database Design and Evaluation Workbench (DDEW) as a graphics-oriented system for database designers. In DDEW, the user interface was integrated with the overall design process. To simplify the design process, multiple windows and a *design tree* were provided to display levels of design and to keep a record of design alternatives. Navigational aids (similar to the prototype SDMS described above) were provided for visualizing large designs and moving through its different levels.

2.2 Exhaustive Generation of Floor Plans

Many researchers in architectural CAD have focussed their attention on providing schemes whereby users can provide the system with constraints on the final building layout; the system then attempts to produce an exhaustive enumeration of all floor plans meeting the user's criteria (see, for example, [Mitc76, Gall81, Flem86, Flem89]).

Flemming [Flem86, Flem89] describes LOOS as a "generative expert system" that complements a designers ability to create floor plans by "systematically searching for alternative solutions with promising trade-offs." The system is able to incorporate a wide range of design considerations. LOOS differs from earlier approaches to exhaustive floor plan enumeration in having a grammar used to describe the design process (see Section 2.3), and in drawing a distinction between quantitative or continuous properties of objects (for example, physical dimensions) and their qualitative or discrete properties (for example, geometric and spatial relations). LOOS does not, however, attempt to either structure the enumerated floor plans or permit the user to control or explore intermediate designs.

2.3 Grammar-based design systems

The underlying representation of floor plans used in LOOS described in Section 2.2 are Flemming's *orthogonal structures*. These orthogonal structures are essentially graphs that describe the spatial relations between rooms and associated geometric information. The design process is described in terms of a grammar for manipulating these orthogonal structures; the language described by this grammar consists of valid floor plans that can be created.

Friedell [Frie90] presents *Landscape Grammars* as a generative mechanism powerful enough to describe a language of landscapes; the emphasis there is on having the grammar guide directly the manipulation of geometry. The rewriting rules in that system essentially create scenes by subdividing objects into more detailed features.

3 An Experiment in Design Automation via Browsing

FLATS[1] is a prototype system for design automation via browsing that was constructed to demonstrate the paradigm of cooperation between the user and the computer in CAD applied to the design of small architectural floor plans. The most important contribution of this system is the support it provides for the exploratory aspect of design. The overall design scenario that is supported is as follows:

1. The user starts up the system and begins designing an object model using conventional modeling aids. The partial design produced is referred to as the *nascent design*.

2. At some point, the user is not sure of which step to take next (or would like to try out the effects of different steps), but has some idea of what constraints the final design(s) should satisfy. He or she then requests "browsing" assistance from the system.

3. The user informs the system of the desired criteria in the final designs via a *goal dialog*.

4. The system generates the set of designs satisfying the given constraints.

5. The system then acquires from the user the attributes of designs with which to graphically *structure* the design space; thus the designs may be considered points in this multi-dimensional browsing space, each dimension of which corresponds to a specified attribute. At any time, a 2-dimensional projection of the browsing space is shown to the user on the graphical *data surface*; the user specifies *navigation* or *layout* attributes that determine the X- and Y- positions of designs on the data surface.

6. The user is presented with the set of design alternatives on a graphical data surface. Since the number of designs may be large, the data surface may not fit in a single window. A *world-view map* always presents the entire data surface in miniaturized form.

7. The user can move about the designs on the data surface via scrolling. The user can "zoom" into portions of the data surface and examine particular designs in detail. The user can also transfer any of the designs from the data surface to the modeling system; this design can then be further developed using the various modeling operators.

8. The user can also specify restrictions on attributes of the designs that should constitute the data surface. This causes the system to display a new data surface. (Steps 7 and 8 can then be repeated.)

[1]For those interested in this detail, the name is derived from Floor plan LAyouT System.

3.1 Structuring and Browsing

Often there will be tens or even a few hundred designs satisfying the goal criteria. These designs constitute the *browsing set*; they can be considered points in an n-dimensional *browsing space*, each dimension of which corresponds to a specified attribute. *Structuring* refers to graphically positioning the designs in browsing space. Even in cases where the number of designs in the browsing space is small, structuring can be useful to bring out logical relationships between the designs. Moreover, the user can explore the spatially structured browsing space using techniques similar to the well established direct manipulation and spatial data management paradigms (Section 2.1).

The attributes of designs used to structure the browsing space are specified by the user from a predefined set. For example, the user might wish to structure the browsing space in a mechanical CAD system using "number of linkages," "maximum torque" and "average pressure" as dimensions. Certain domains may have default structuring dimensions that are always useful. For example, "number of rooms" is almost always a useful dimension in architectural design. The *characteristic vector* of a design contains the values of its structuring attributes.

To further restrict the set of design alternatives, the user can specify a *selection vector* to select designs through which he or she would like to browse. This operation is essentially the same as a selection operation as defined in relational database terminology. For example, the user of an architectural CAD system might decide to examine only houses that had 5 or fewer rooms; later he or she might decide to browse through those that had areas less than 3000 square feet.

The set of designs that need to be presented to the user at any time constitutes the *data surface*. Finally, the attributes that determine the X- and Y-positions of designs on the data surface will be referred to as the *navigation* or *layout dimensions*. For example, the user might request that designs be positioned on the data surface using the "number of rooms" as the X-dimension, and "area" as the Y-dimension.

3.2 User's Conceptual View of the System

The user sees the system as an automated design assistant. It is used to explore alternatives when he or she does not have a fixed final model in mind, but has some idea of the properties of the ultimate design. If the user had a firm idea of what the final design would be and knew the sequence of operations that would lead to that goal, he or she would use the system's conventional components to create the design. Often, however, the user can only provide a "backbone" or starting design (the nascent design) and some indication of the direction he or she is headed in (the goal criteria). In that case, the user expects the system to help him or her explore elaborations of the nascent design that are in the "right direction."

The structuring mechanism helps the user explore design alternatives in a systematic way—by varying those properties of the designs that are of primary interest. By controlling the layout of the data surface using navigation dimensions, the user can explore designs that differ only along that dimension. Spatial proximity of designs having similar values for the dimension being visualized provides the user with a sense of continuity in exploring the designs. Consider, for example, the process of designing a house. If "house area" is one of the layout dimensions, the user would only have to look in the proximity of a given house on the data surface to find a house with an area slightly larger than its area.

I believe that this conceptual model of the system supports well "exploratory design" performed by human designers (Section 1). In that scenario, the goal is roughly known (along with certain constraints), but other attributes need to be "tuned" to provide a design satisfying "unquantifiable requirements."

4 System Architecture

Figure 1 presents an overview of the architecture of FLATS. The user interacts with the system via three modules: the *modeling system*, the *goal dialog module*, and the *browsing system*. The goal dialog module is essentially an interface to the *constrained generator*. Between the modeling system and the generator is an additional component—the *recognizer*.

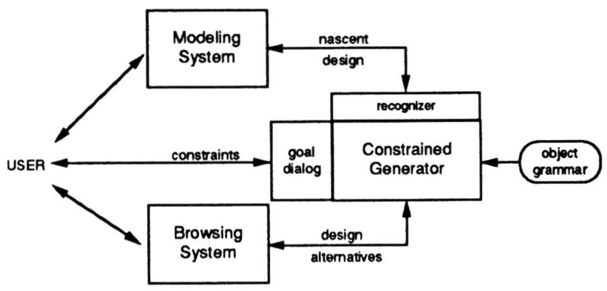


Figure 1: System Architecture

The major functions of these modules are:

- The *modeling system* allows the user to create and manipulate a *nascent* design as is possible with a conventional CAD system.

- The *recognizer* converts a design from the representation used by the modeling system to a valid design or "word" in the language of designs described by the object grammar. This word also forms the starting point from which the generator produces design alternatives.

- Via the *goal dialog*, the user can specify any constraints that he or she wishes the generated design alternatives to satisfy.

- The *generator* develops the nascent design to produce design alternatives that are consistent with both the user-specified constraints and the rewriting rules specified in the language of designs (*object grammar*).

- The *browsing system* structures the design alternatives produced by the generator in a spatial framework specified by the user, who then can *browse* through these designs via graphical gestures such as scrolling and pointing.

4.1 Modeling System

The main function of the modeling system is to allow the user to create a nascent design. This component of the system allows the user to create and manipulate floor plans, essentially at the level of individual rooms, and objects within rooms. The major operations currently supported include:

- create rooms as arbitrary shaped polygons
- create corridors etc. as a set of connected lines
- delete any object
- rotate any object
- move any object
- modify an object's properties (such as orientation and type)
- subdivide a room
- save to and read from files
- constrain drawing to grid
- generate alternatives (i.e. pass nascent design to the Generator)

Figure 2 shows a snapshot[2] of a floor plan being created, and the currently available menu options. Along the left edge are the drawing menu items, represented by icons. Clicking the mouse on the command buttons towards the top, for example, "File", "Edit", etc., creates popup menus that allow the user operations such as "select a room," "delete a room," "save to file" and "read from file." Many of these would be familiar to users of other CAD systems; the remainder are particular to the domain of floor plans and to the grammar I use to describe the process of developing floor plans.

Design Representation

The description of a design in the modeling system consists of parts that have a direct graphical interpretation and can be manipulated by the user. For example, a floor plan is described as a list of polygons corresponding to the room boundaries, each polygon being described in terms of its vertices. The representation scheme used by the modeling system may be different from the linguistic (language-theoretic) description required by the generator and discussed in the next section. Thus, we need to convert the representation of a design from the scheme used in the modeling system to that used in the generator.

4.2 Constrained Generator

The generator essentially takes a starting "word" in the language of designs and applies rewriting rules from the grammar (describing this language) to produce alternative designs satisfying the constraints specified by the user.

Figure 3 depicts the architecture of the generation subsystem. As shown in the figure, the generator contains two modules—the constraint tester and the recognizer—to ensure that a valid "word" is used as a starting point, and that only valid words are produced. The *constraint tester* has a

[2]For clarity, I use screen dumps in the figures while describing the modules. Though these are in black and white, the actual system uses color.

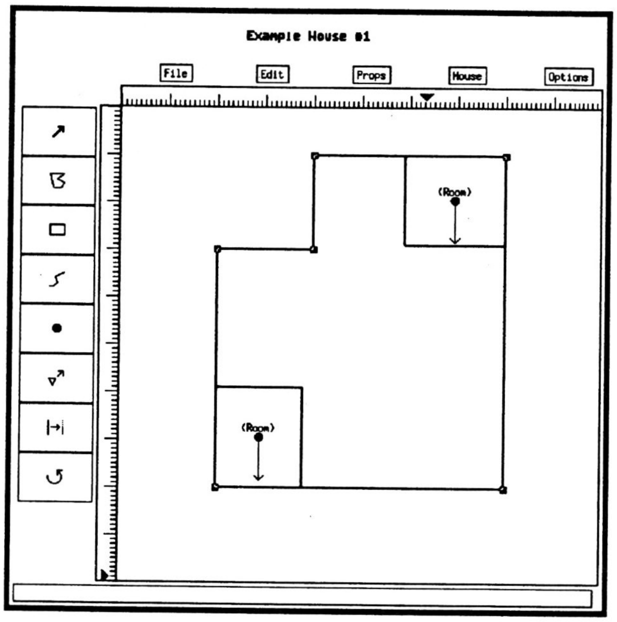


Figure 2: Modeling System—snapshot

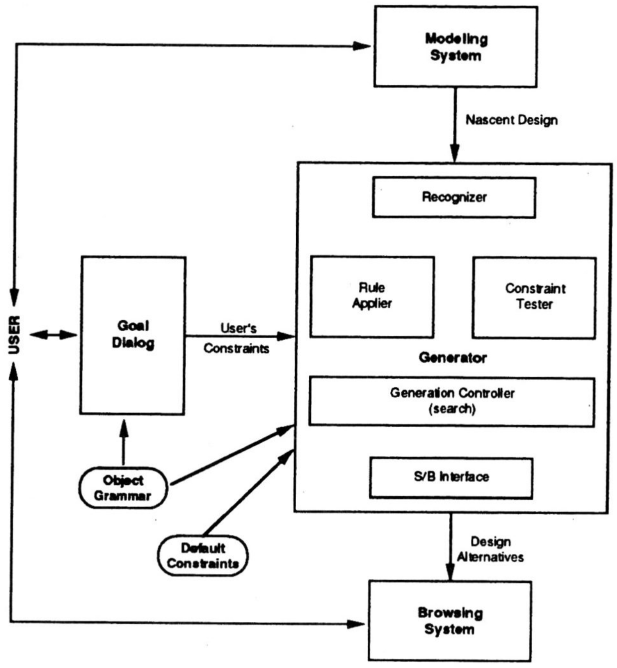


Figure 3: Constrained Generator Architecture

database of constraints, which are of two types: those derived from domain characteristics, such as "the kitchen and dining must always touch," and those specified by the user, such as "the house should have 2 bedrooms"; these are used by the generator to prune fruitless paths when exploring different alternatives.

The *recognizer* consists of a set of rules that operate on the nascent design represented by the modeling system and transform this design into a form (valid "word") that the generator can develop further.

The generator has rules to deal with *hierarchic, topological* and *geometric* considerations. For example, one of the rules (named *Tri-room*, in Figure 4) adds a triangular room to the house. The *rule applier* module in Figure 3 incorporates these rules and the details of how to apply them.

The core of the generation algorithm is constraint-based search, implemented by the *Generation Controller* module. It uses standard search techniques—all the rewriting rules applicable at each step to the set of current alternatives (which is initialized to the nascent design) are applied until no more are applicable. Constraints are used to prune branches of the search graph; paths containing duplicate designs are also deleted.

Finally, the *structuring/browsing (S/B) interface* module essentially passes the generated alternatives to the browsing system.

The *Goal Dialog* module, also shown in Figure 3, allows the user to specify constraints to the system. These can consist of restrictions on derived attributes of the designs (for example, "number of rooms", "area", etc.). The user can also focus the generation process by specifying the set of rules to be used by the generator (for example, "use only rewriting rules that add rectangular rooms"). Thus, the object grammar is also shown as an input to the Goal Dialog in the figure.

Design Representation

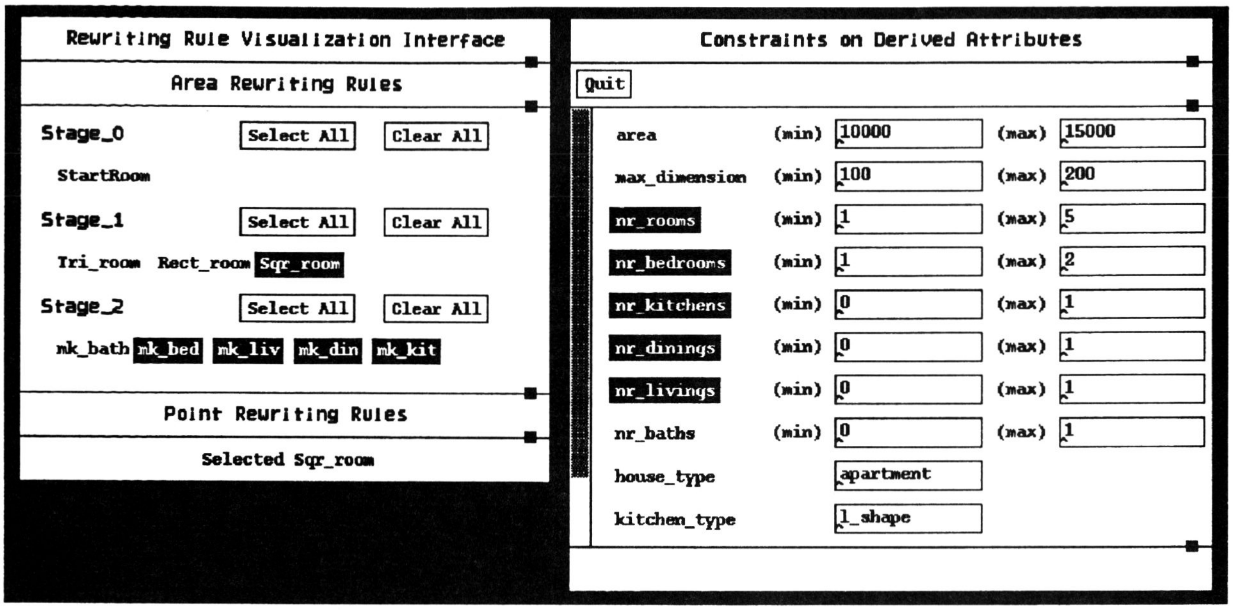
To reduce the complexity of the design problem, I chose to focus on 2-dimensional objects. The representation of designs was developed based on the following considerations:

- *topology and geometry*: we need to be manipulate both topological features—for example, the kitchen and dining room in a house touch—and geometrical features—for example, the shape of the bedroom is a rectangle;

- *hierarchy*: we need to be able to develop designs hierarchically and visualize partially developed designs. This requirement is motivated by the cooperative CAD paradigm—the user might wish to develop a given design only partially and then examine the alternatives produced;

- *arbitrary shapes*: we need to be able to represent arbitrary shaped features in designs. This is a useful requirement for most graphical domains, even for the domain of floor plans (see, for example, [Krie83]).

Another important consideration was the two different styles of design that are widely used—*outside-in* and *inside-out*. For example, an architect designing a house whose outer boundary is fixed (because of constraints placed upon it by its location, other neighboring houses, etc.) might start by roughly dividing the house into the day-area, night-area, food-preparation area, etc. This is the *outside-in* approach. On the other hand, an *inside-out* approach might be used, for example, when we are designing a house on a large plot of land (thus, we have few restrictions on the outer shape), but are concerned with issues such as sharing the plumbing between rooms; in this case, we might start with one room, add another touching it, and so on.



Rewriting Rule Visualization Interface

Area Rewriting Rules

Stage_0 Select All Clear All

StartRoom

Stage_1 Select All Clear All

Tri_room Rect_room Sqr_room

Stage_2 Select All Clear All

mk_bath mk_bed mk_liv mk_din mk_kit

Point Rewriting Rules

Selected Sqr_room

Constraints on Derived Attributes

Quit

area	(min)	10000	(max)	15000
max_dimension	(min)	100	(max)	200
nr_rooms	(min)	1	(max)	5
nr_bedrooms	(min)	1	(max)	2
nr_kitchens	(min)	0	(max)	1
nr_dinings	(min)	0	(max)	1
nr_livings	(min)	0	(max)	1
nr_baths	(min)	0	(max)	1
house_type		apartment		
kitchen_type		l_shape		

Figure 4: Generator and Goal Dialog—snapshot

Flemming's LOOS system [Flem89] is an example of the inside-out style—the core of his system is the addition of one rectangle at a time to an existing group of rectangles subject to certain constraints. Friedell's *Landscape Grammars* [Frie90] demonstrate the outside-in style of design—designs are developed by subdivision.

I chose to represent designs as area, line and point features, similar to those used by [Frie90]; these features specify the shape, location, orientation and other properties of objects in the designs. *Area features* (AFs) correspond to polygonal features, such as rooms; *line features* (LFs) correspond to linear phenomena, such as corridors; *point features* (PFs) describe point phenomena, such as furniture in a room (where the exact shape is unimportant). An example of a floor plan is shown in Figure 5. The rooms are described by their polygonal boundaries. The thick line dividing the figure represents a corridor. The small circle in the bathroom represents a sink; the small circle in the living room represents a table.

The design process is described by subdivision grammars that consist of rules to rewrite area features—these rules incorporate both inside-out and outside-in styles of developing designs, and basically define the "language" of valid designs. An example of a rule that divides a house into the night- and day-areas is shown graphically in Figure 6. The details of the types and application of rules are described in [Koch90].

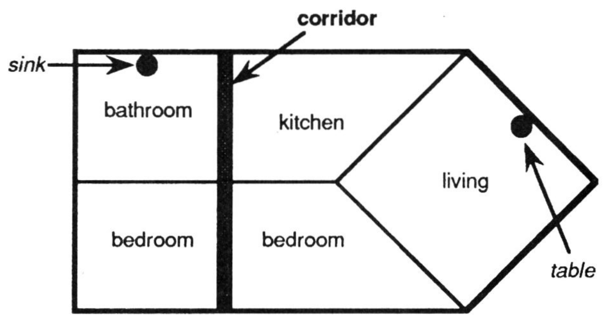


Figure 5: A Simple Floor Plan

The Recognizer

As mentioned in Section 4.1, the representation scheme used by the modeling system may be different from the linguistic description required by the generator. Even though the linguistic description of a design and its description in the modeling system both correspond to the same design, the attributes and relations brought out and emphasized by each are often different. For example, in the modeling system, the kitchen in a house may be described by the coordinates of its vertices $(X_0, Y_0, X_1, Y_1, ...)$. In the linguistic description, its size and type become important. Moreover, it has a relation—*touching*—with the dining room that is not described directly by the modeling system (and the user cannot manipulate it). Finally, concepts such as the "night-area" which have no graphical representation may not be defined in the modeling system; similarly absolute X- and Y-locations that are needed by the modeling system are not stored in the linguistic description.

Both of the representations discussed above are abstractions of the same object but for different purposes; also, often they will not be in a one-to-one correspondence. For example, two floor plans differing only in the floor color might have different descriptions in the modeling system, but the same linguistic description. Note that in a syntax-directed modeling system, the representation used would correspond more closely to the language-theoretic linguistic-based one. However, syntax-based modeling is currently an open research is-

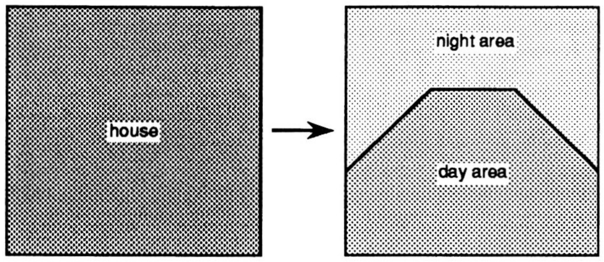


Figure 6: Rule for Subdividing a House into Night and Day Areas

sue by itself. In my prototype system (and in CAD systems that are not fully syntax-directed), we need to convert between the two representations, since the generator can only operate on valid linguistic descriptions. The *recognizer* in my system performs precisely the task of transforming a design from its description in the modeling system to a valid linguistic description ("recognizing its validity", or parsing).

In Figure 4 the user has specified that the system develop the nascent design of Figure 2 by adding square rooms (the highlighted rule named *Sqr_room* in the window titled "Rewriting Rule Visualization Interface") and developing rooms into a kitchen, living, dining and bedroom (the rules named *mk_kit, mk_liv, mk_din, mk_bed*), subject to the constraints (as shown in the window titled "Constraints on Derived Attributes") that the total number of rooms is 5 or less, and that there is no more than one kitchen, living and dining each, and 1–2 bedrooms. At this point, the system generates various alternatives, which are then passed to the browser.

4.3 Browsing System

Figure 7 depicts the architecture of the Browsing system, including control and data flow. The ovals correspond to the domain-specific components of the system. The *Top Level* controller essentially initializes all modules and then stays in the following loop (until requested to exit the module):

```
loop
        get layout dimensions and selection vector from
        the user via the Dialog Manager;
        produce the current data surface via the Data Sur-
        face Creator;
        display the data surface to the user via the Data
        Surface Display;
until requested to quit
```

The *Dialog Manager* consists of two modules:

- *Selection Setup*, which allows the user to specify value restrictions on some attributes of the designs to be visualized; and,
- *Navigation Setup*, which allows the user to specify X- and Y-dimensions to be used in the spatial layout of the data surface window.

Information about the selection vector and the navigation dimensions is then passed to the Data Surface Creator and the Data Surface Display modules.

Using the set of alternatives (stored as linguistic descriptions) produced by the generator the current data surface is produced by the *Data Surface Creator*. This operation is essentially the same as a *selection* operation on a relational database. The selected designs are then passed to the Data Surface Display module to be shown to the user.

The *Data Surface Display* module shows the current data surface to the user, who can also move about and explore various designs in detail, via the following modules: 1)the *Structurer (Layout Manager),* which uses the currently selected X- and Y-attributes to produce information about where to place each design on the two-dimensional data surface window, where to draw axes, etc.; 2) the *Visualizer,* which produces a graphical description of a given design at a given scale from the linguistic description of the design; 3) *Traverse and Zoom,* which allows the user to move about the data surface and to visualize designs in greater detail; and, 4) the *Display Control,* which interfaces with the windowing system to create windows, send drawing commands, and process windowing events (such as mouse clicking, and scrolling). Since the data surface may not fit in a single window, a *world-view map* presents the entire data surface in miniaturized form. "You are here" highlights in the world-view map indicate the regions of the data surface that are presented at normal scale in scrollable windows.

Figure 8 shows a snapshot of the user browsing through the set of design alternatives that was produced by the generator. As shown on the right side of the figure (the window titled "Selection Vector"), the user has decided to browse through all designs (that is, no restrictions); also, the X-axis for structuring the data surface has been chosen as the "number of bedrooms". As can be seen, the data surface is larger than the window size; hence, scrollbars are provided, allowing the user to look at the different designs via scrolling. The vertical dashed line separates the floor plans with 1 bedroom (left) from those with 2 (right). Finally, the user has chosen to zoom into one of the floor plans (outlined by the dashed rectangle) as shown in the window towards the bottom right.

At this point, the user can select any of these alternatives and transfer it to the modeling system (by clicking on the "Txfer..." button near the top of the zoom windows); that alternative then becomes the new nascent design, and the steps described so far in this example can be repeated to develop the design further.

4.4 Implementation

FLATS currently runs on both a Sun SPARCstation and a DECstation 3100, utilizing the X window system (Version 11). The software is written in C, and is built on top of the X library, X toolkit and the MIT/Athena widget set.

Figures 9 and 10 show FLATS being used. In Figure 9, the nascent design—a floor plan with two rooms—is shown on the manual modeling system (the window towards the top left of the figure). The window titled "Rewriting Rule Visualization Interface" shows that the user has restricted the generator to use the highlighted rules: add a square room (*Sqr_room*), and convert a room to a bathroom (*mk_bath*), a bedroom (*mk_bed*), a living (*mk_liv*), a dining (*mk_din*), or a kitchen (*mk_kit*). The window titled "Constraints on Derived Attributes" lists the constraints that the user specified: 1–5 rooms, at most 2 bedrooms, and at most 1 kitchen, dining, living, and bathroom each. The window in the bottom right of the figure, titled

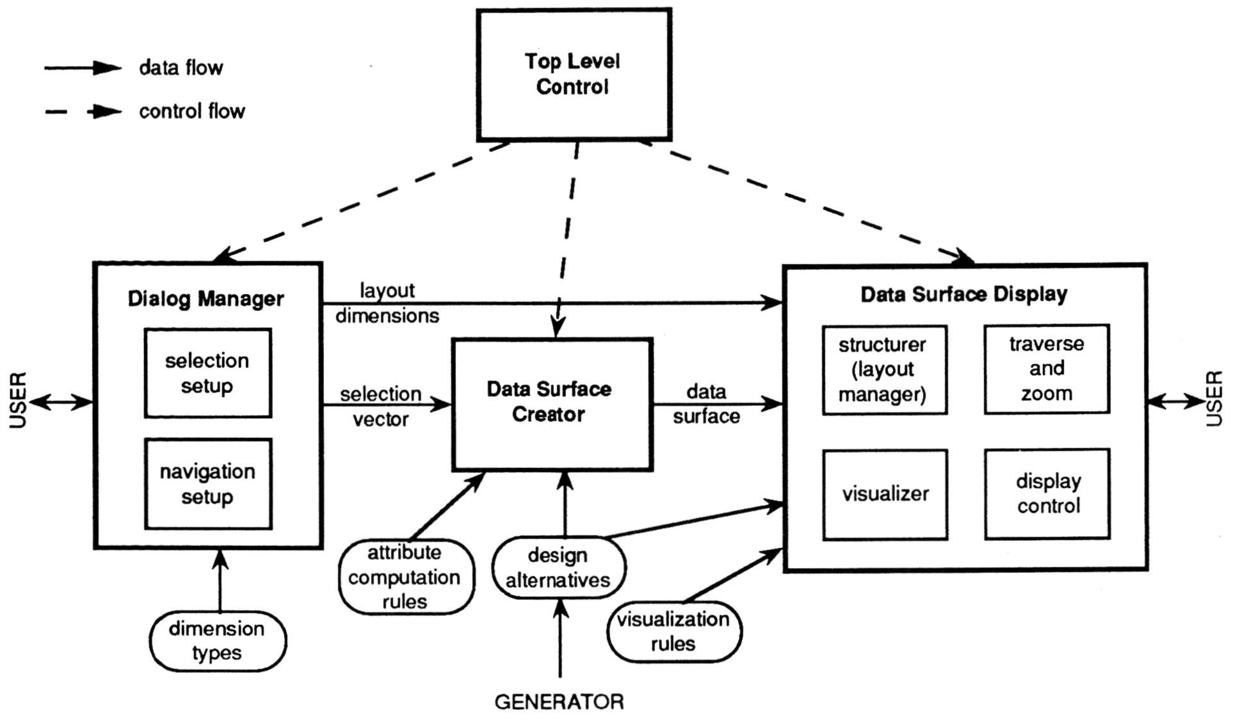


Figure 7: Browsing System Architecture

"Selection Vector", shows that that user wishes to view the design alternatives laid out along the X-axis using the number of bedrooms, and along the Y-axis using the number of dining rooms. The window titled "World View" towards the right shows the entire data surface in miniaturized form, along with highlights showing the current data surface. Finally, the window in the top right, titled "Current Data Surface" shows the design alternatives (numbering 18) that satisfy the user specified criteria. The scrollbars can be used to examine different portions of the data surface. Also, the floor plans on the left of the vertical dashed line have 1 bedroom; those on the right have 2.

Figure 10 shows another example (the figure has the same layout of the various windows as the previous example). In this example, the user starts by dividing the house into "night" and "day" areas (as shown in the top left of the figure). The rest of the figure can be interpreted in a manner similar to the previous example.

5 Conclusions and Future Direction

I believe that the paradigm for "Design Automation via Browsing", based on the techniques described in this paper, offers a qualitatively significant improvement over currently available CAD technology. Preliminary experimentation with the prototype system in the domain of small architectural floor plans has shown that the system allows the user to systematically control the generation of design alternatives and explore through them, which also leads to a better understanding of the relations between the alternatives.

At present, my focus is on developing the grammar further and exploring the limits of this system. Specifically, I am looking at the following issues:

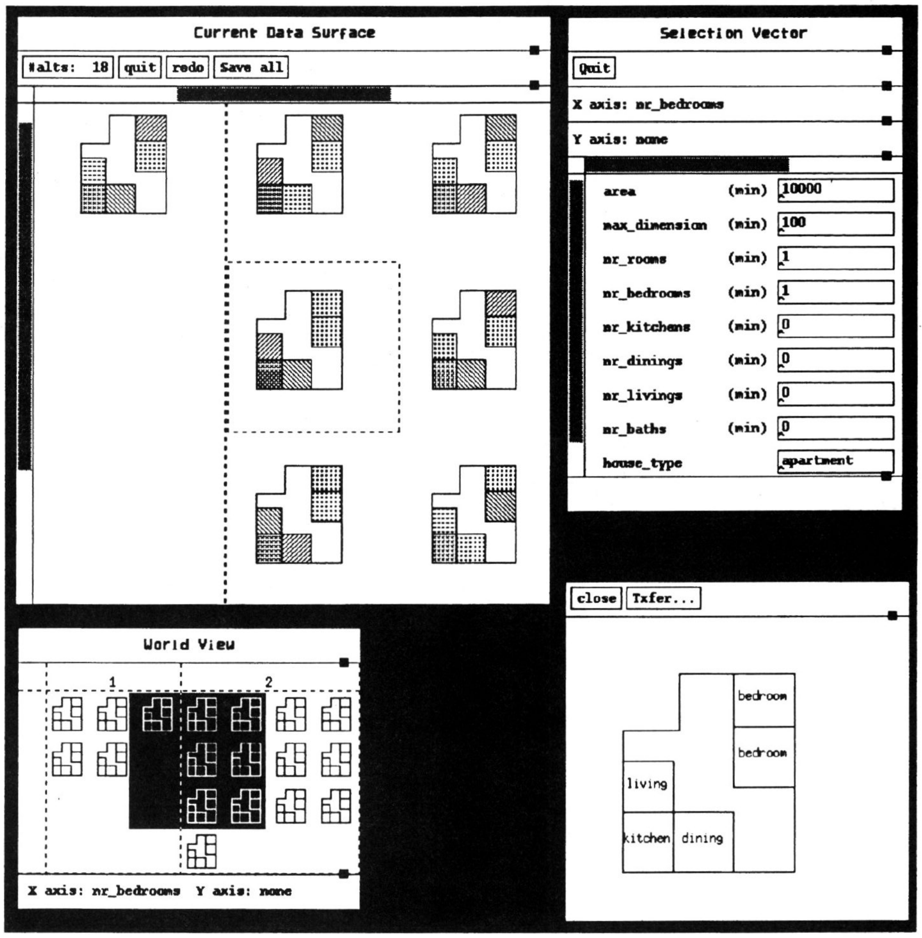
- how do we make sure that the user does not get "lost" while browsing through a data surface containing hundreds of designs?
- can we speed up the generation process by using the selection vector specified by the user (while browsing) as additional constraints that can further prune the search graph?
- what issues involved when applying this technology to another application domain?

This research addresses one of the major limitations of current CAD technology—the lack of user-computer cooperation in the design process—and provides a common conceptual framework in which CAD systems similar to the prototype system described here can be constructed. With such systems, I believe that the productivity of designers can be significantly improved.

References

[Done78] Donelson, W. 1978. "Spatial management of information," *Proceedings of ACM Siggraph '78*, Atlanta, Georgia.

[Flem86] Flemming, U. 1986. "On the representation and generation of loosely packed arrangements of rectangles," *Environment and Planning B* **13**: 189-205.



X axis: nr_bedrooms
Y axis: none

```
area            (min) 10000
max_dimension   (min) 100
nr_rooms        (min) 1
nr_bedrooms     (min) 1
nr_kitchens     (min) 0
nr_dinings      (min) 0
nr_livings      (min) 0
nr_baths        (min) 0
house_type            apartment
```

X axis: nr_bedrooms Y axis: none

Figure 8: Browsing through Structured Designs—snapshot

[Flem89] Flemming, U. 1989. "A generative expert system for the design of building layouts," Final report, Carnegie-Mellon University.

[Frie82] Friedell, M., Barnett, J., and Kramlich, D. 1982. "Context-sensitive graphic presentation of information," *Computer Graphics* **16 (3)**: 181-188.

[Frie90] Friedell, M. 1990. "Constrained, grammar-girected generation of landscapes," *Graphics Interface '90*, Halifax, Canada.

[Gall81] Galle, P. 1981. "An algorithm for the exhaustive generation of building floor plans," *Communications of the ACM* **24** : 813-825.

[Hero80] Herot, C., Carling, R., Friedell, M., and Kramlich, D. 1980. "A prototype spatial data management system," *Proceedings of ACM Siggraph '80*, Seattle, Washington.

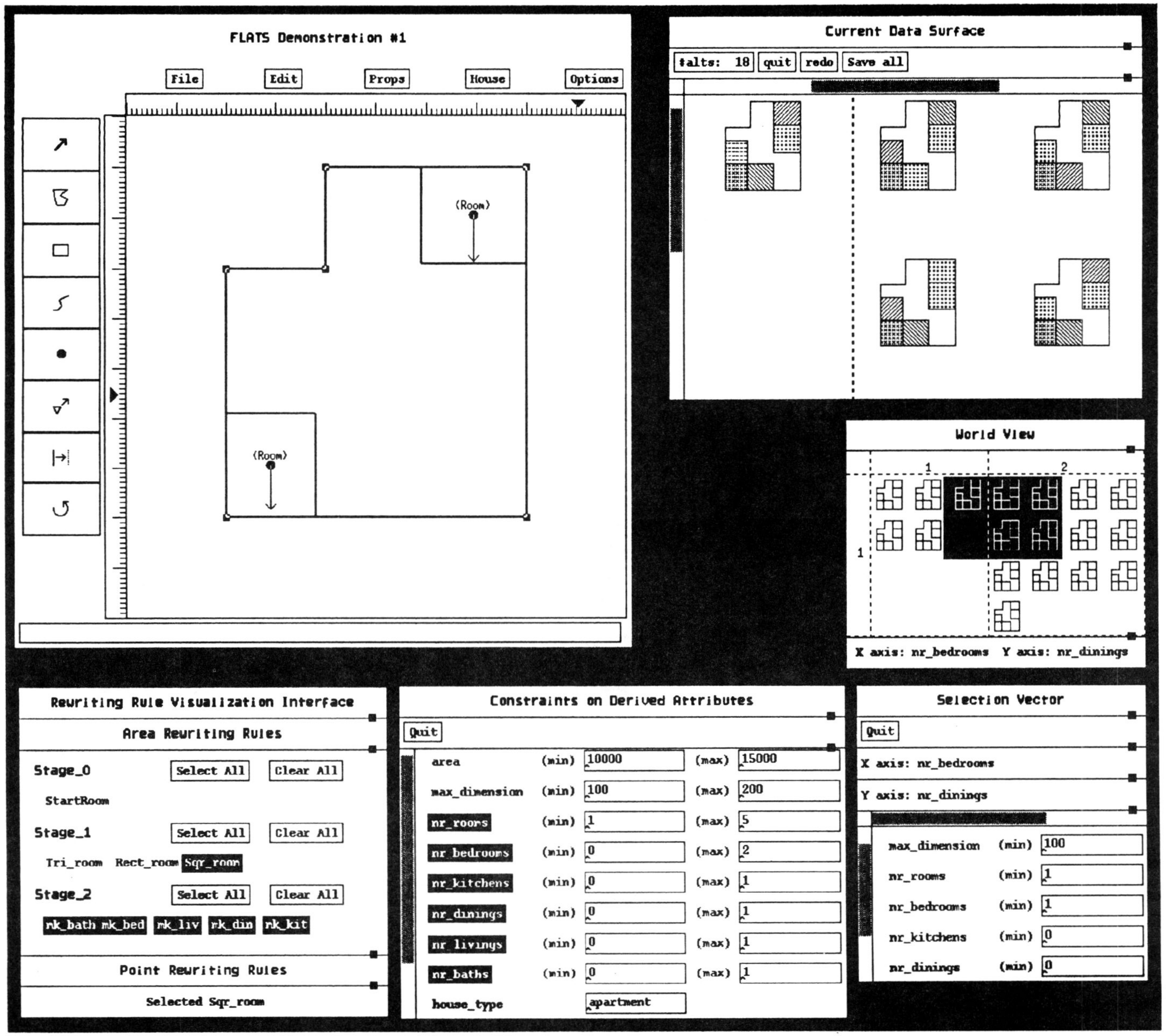
[Koch90] Kochhar, S. 1990. "Synergistic design automation via the constrained generation and browsing paradigm," Ph.D. Dissertation, Harvard University, *in preparation*.

[Krie83] Krier, R. 1983. *Elements of Architecture*, London: Architectural Design.

[Mitc76] Mitchell, W.J., Steadman, J.P., and Liggett, R.S. 1976. "Synthesis and optimization of small rectangular floor plans," *Environment and Planning B* **3** : 37-70.

[Rein84] Reiner, D, Brodie, M., Brown, G., Friedell, M., Kramich, D., Lehman, J., and Rosenthal, A. 1984. "The database design and evaluation workbench (DDEW) project at CCA," IEEE *Database Engineering* **7 (4)**.

[Schn82] Schneiderman, B. 1982. "The future of interactive systems and the emergence of direct manipulation," in *Human Factors and Interactive Systems*, Yannis Vassiliou (ed.), *Proceedings of the NYU symposium on user interfaces*, New York, May 26-28.



Rewriting Rule Visualization Interface

Area Rewriting Rules

Stage_0 Select All Clear All

StartRoom

Stage_1 Select All Clear All

Tri_room Rect_room Sqr_room

Stage_2 Select All Clear All

mk_bath mk_bed mk_liv mk_din mk_kit

Point Rewriting Rules

Selected Sqr_room

Constraints on Derived Attributes

Quit

area	(min)	10000	(max)	15000
max_dimension	(min)	100	(max)	200
nr_rooms	(min)	1	(max)	5
nr_bedrooms	(min)	0	(max)	2
nr_kitchens	(min)	0	(max)	1
nr_dinings	(min)	0	(max)	1
nr_livings	(min)	0	(max)	1
nr_baths	(min)	0	(max)	1
house_type		apartment		

Selection Vector

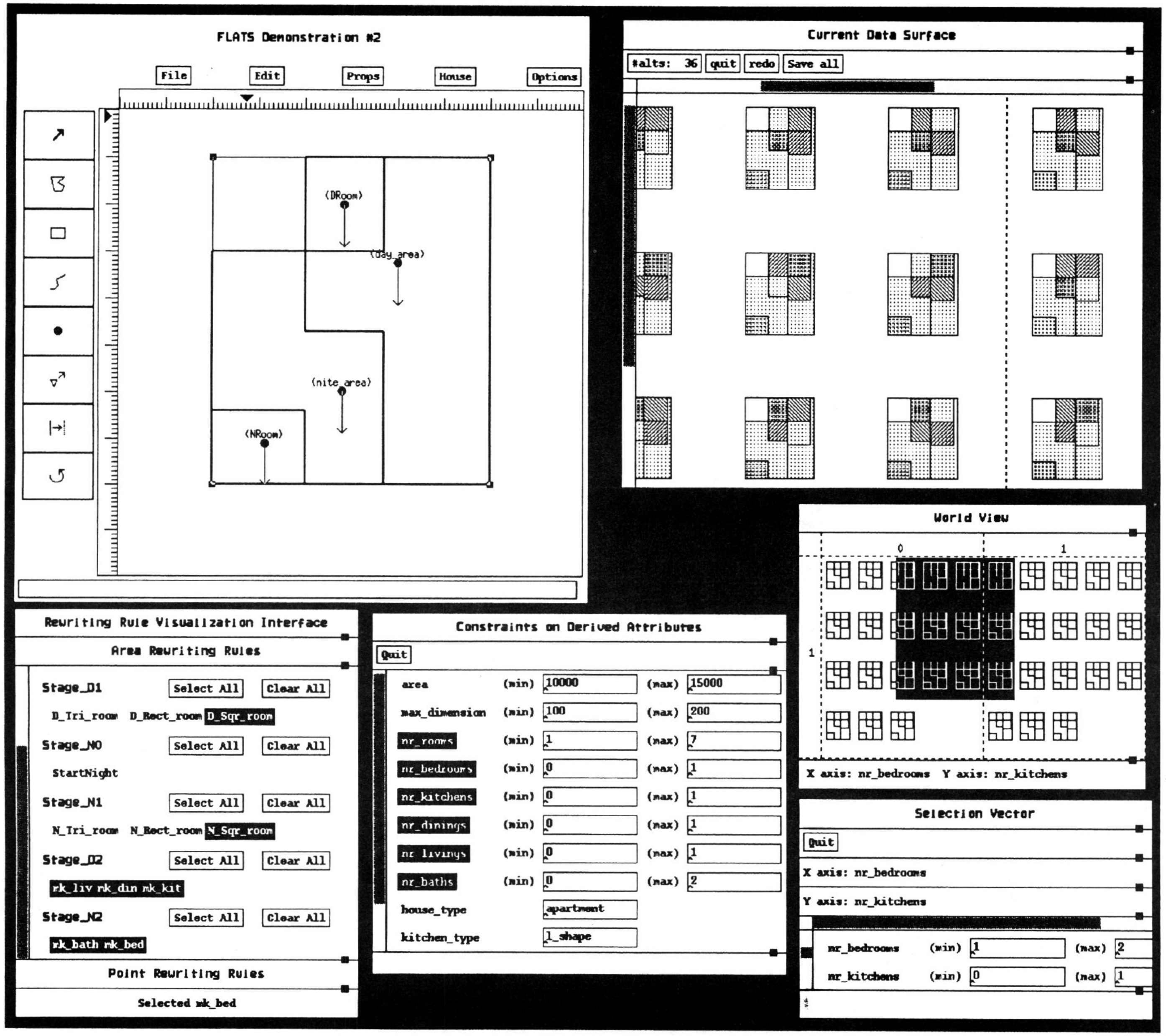
Quit

X axis: nr_bedrooms

Y axis: nr_dinings

max_dimension	(min)	100
nr_rooms	(min)	1
nr_bedrooms	(min)	1
nr_kitchens	(min)	0
nr_dinings	(min)	0

Figure 9: FLATS in use — Example 1



X axis: nr_bedrooms Y axis: nr_kitchens

Stage_D1 Select All Clear All
D_Tri_room D_Rect_room D_Sqr_room
Stage_N0 Select All Clear All
StartNight
Stage_N1 Select All Clear All
N_Tri_room N_Rect_room N_Sqr_room
Stage_D2 Select All Clear All
rk_liv rk_din rk_kit
Stage_N2 Select All Clear All
rk_bath rk_bed

Selected rk_bed

area	(min)	10000	(max)	15000
max_dimension	(min)	100	(max)	200
nr_rooms	(min)	1	(max)	7
nr_bedrooms	(min)	0	(max)	1
nr_kitchens	(min)	0	(max)	1
nr_dinings	(min)	0	(max)	1
nr_livings	(min)	0	(max)	1
nr_baths	(min)	0	(max)	2
house_type		apartment		
kitchen_type		l_shape		

X axis: nr_bedrooms

Y axis: nr_kitchens

nr_bedrooms	(min)	1	(max)	2
nr_kitchens	(min)	0	(max)	1

Figure 10: FLATS in use — Example 2