Intelligent Layout for Information Display: An Approach Using Constraints and Case-based Reasoning

by Grace Elizabeth Colby
B.S., Design
Illinois Institute of Technology
Chicago, Illinois
1985

Submitted to the Media Arts and Sciences Section,
School of Architecture and Planning,
in partial fulfillment of the requirements of
the degree of Master of Science at the
Massachusetts Institute of Technology,
June 1992

[illegible]

Signature of the author
Media Arts and Sciences Section
May 8, 1992

[illegible]

Certified by
Muriel Ruth Cooper, B.F.A.
Professor of Visual Studies, Thesis Supervisor

Accepted by
Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students

Intelligent Layout for Information Display: An Approach Using Constraint and Case-based Reasoning

by Grace Colby

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning, on May 8, 1992 in partial fulfillment of the requirements of the degree of Master of Science at the Massachusetts Institute of Technology.

Abstract

There is an increasing need for computer systems that contain graphic design intelligence. Current electronic information technology creates situations where the expertise of a graphic designer is sorely needed but the information is transferred and displayed so rapidly that the involvement of a graphic designers is precluded. This thesis presents an approach to representing graphic design intelligence in a system that automatically generates layouts for text and image information. The approach uses two artificial intelligence techniques: constraints and case-based reasoning. The approach also includes use of a general design knowledge base that contains knowledge of layout elements. grids and rules of legibility. The system makes design decisions based on layout content structure. The prototype system can generate new layouts as well as modify layouts in a dynamic display environment. The contribution of this research lies in culling layout design knowledge that can be represented using existing artificial intelligence technology and demonstrating how this knowledge can be used to solve layout problems automatically.

Thesis Supervisor: Muriel Cooper, BFA
Title: Professor of Visual Studies

This work was supported in part by Bitstream, Digital Equipment Corporation, Hewlett-Packard, Hitachi, NYNEX, Sony and the Rome Air Development Center (RADC) of the Air Force System Command the Defense Advance Research Projects Agency (DARPA) under contract No. F30602-89-0022. The views and conclusions in this document are those of the author and should not be interpreted as necessarily representing the official policies, express or implied, of the Rome Air Development Center (RADC) of the Air Force Systems Command, the Defense Advanced Research Projects Agency (DARPA), or the United States Government.

Intelligent Layout for Information Display: An Approach Using Constraints and Case-Based Reasoning

by Grace Colby

The following people have served as readers for this thesis.

[illegible]

Henry Lieberman, H.D.R
Research Scientist, MIT Media Laboratory

[illegible]

Jock Mackinlay, Ph.D.
Member of the Research Staff, Xerox PARC

Acknowledgments

Many people have contributed to this work. I would especially like to like to thank Muriel Cooper for providing an environment where I could grow both as a designer and as an encoder of design.

I would also like to thank:

Ron MacNeil, Henry Lieberman, Louie Weitzman, and David Small for their advice and assistance throughout this project.

Those outside of the Media Lab who have taken time and interest in this project: Jock Mackinlay and Joe Marks.

My fellow graduate students at the Visible Language Workshop – Didier Bardon, Michelle Fineblum, Steve Librande, Laura Scholl, Bob Sabiston, David Young.

Members of CGWII for their effort in encoding layout cases – Karen Donoghue, Craig Kanarick, B.C. Krishna, Alice Lei, and Alan Turransky.

Michael Johnson for the development of "BWI."

Amy Freeman for those last minute Fed-exes.

Linda Peterson for all her help.

Paula White for her editorial assistance.

Elizabeth Glenewinkel for her moral support.

My family: Coco, Gene, Caspar, Ogg, Judy, Mig, Skeeter, Daniel-Guy, and Cincy for their tremendous support throughout this endeavor.

Contents

Chapter 1
Introduction 6

Chapter 2
System Capabilities 12

Chapter 3
Approach 20

Chapter 4
Related Research 32

Chapter 5
How It Works 36

Chapter 6
Evaluation and Conclusion 66

Bibliography 75

Chapter 1 **Introduction**

There is an increasing need for computer systems that contain graphic design intelligence. This thesis presents an approach to representing graphic design knowledge in a computer system that can automatically generate layouts of text and image information. This document describes the approach and a prototype system that was developed to demonstrate it's utility.

Problem and Motivation

Current electronic information technology provides the ability to generate, transfer, customize, and display visual information rapidly. News information can be customized for the individual and delivered electronically [BACK 83]. Hypermedia documents can be edited on the fly to accommodate a user's interests and cognitive skills [ROBN 91] [FINB 91], and users can manipulate real-time data. These new capabilities present a plethora of visual communication problems including illegible text and poorly organized information. Traditional graphic design is rich with methods for solving these visual communication problems. Unfortunately, the many variables implicit in this new technology preclude the involvement of a graphic designer to formulate the necessary individual solutions [COOP 89]. This predicament points to the need for computer systems that contain graphic design knowledge and can automatically generate well designed displays of information.

A number of researchers have investigated the problem of encoding graphic design knowledge for the automatic generation of visual information displays. Most of this work [MACK 86] [MARK 90] has focused on chart and diagram design. The research presented in this thesis focuses on encoding knowledge about the layout of text and image information to be displayed on a computer screen.

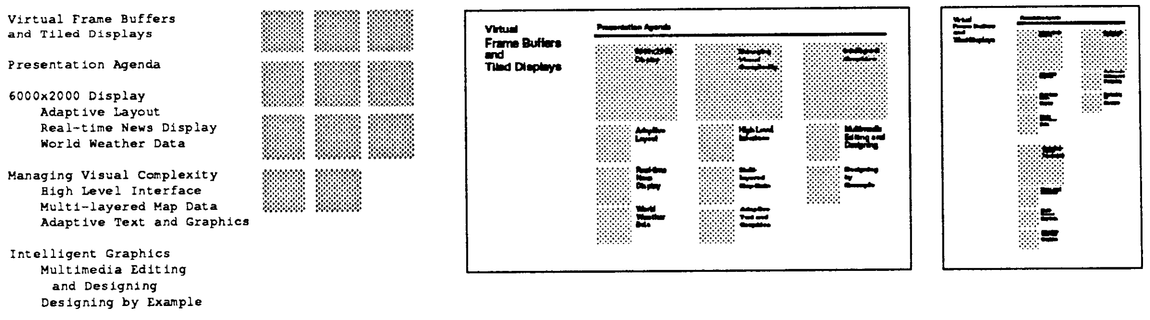
The layout of text and image information is traditionally referred to as "page layout." Good page layout design is critical to the communication effectiveness of visual information. Although the research presented in this thesis is applied to information that will be displayed on a computer screen, the graphic design knowledge that is used has its roots in page layout.

In traditional print media, graphic designers take the given content of a book or magazine and lay out each page to make the information understandable and to communicate the appropriate overall message. The designer carefully chooses the color, size, and position of individual elements such as photographs and text to: visually structure the information so that it reflects the content structure, ensure legibility, and create an overall look that conveys the nature of the information. These characteristics are certainly desirable in the presentation of information on computer displays.

There are many aspects of generating layouts for text and image information. For instance, the information itself must be developed appropriately for the given audience and intended message and an appropriate interaction strategy must be developed if the information is to be interactive. In addition, the visual appearance of the information must be specified. This thesis concentrates on this last aspect of generating layouts for text and image information displays: designing the *visual layout* of the information.

Approach

This thesis presents an approach to representing graphic design knowledge about text and image layout. The prototype system, LIGA (Layout Intelligence for Graphics Automation), was developed by the author to demonstrate this approach. The system contains graphic design knowledge and can use this knowledge to generate layouts as well as adapt layouts in the event the display environment changes. Figure 1.1a-c shows an example of what the system can produce. Figure 1.1a shows text and image content that needs to be laid out. Figure 1.1b shows the layout that LIGA generated for the content. Figure 1.1c shows the layout that LIGA generated when the display area was reduced.



(a) Content to be laid out (b) Layout generated by LIGA (c) Modified layout

Figure 1.1 a-c Shows an example of what the LIGA system can produce.

The approach to representing design knowledge for generating layouts is four-fold. The approach uses case-based reasoning, constraints, a general design knowledge base, and knowledge of layout content structure. Figure 1.2 shows how these approaches are related in the prototype system.

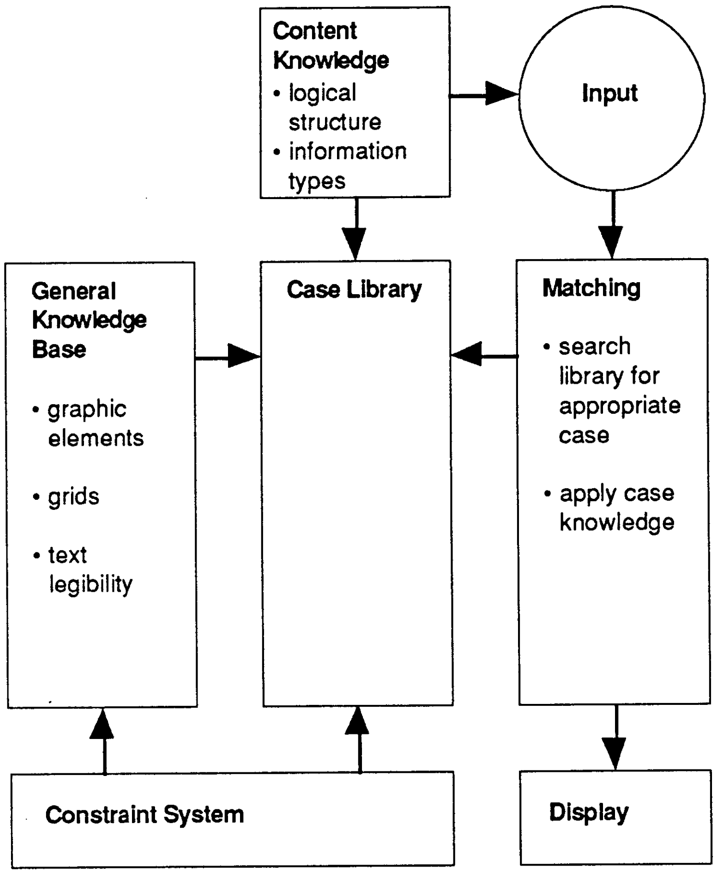


Figure 1.2 The components of the LIGA prototype correspond to the approach. The approach uses a case library, constraints, a general knowledge base, and layout content knowledge.

Case-based reasoning

Case-based reasoning has been included in this approach because it is a good match to the natural layout design process where knowledge is best expressed as specific examples rather that as general rules. Case-based reasoning is a problem solving method that makes decisions based on prior examples or "cases." In LIGA, example layouts are used as cases.

Constraints

The second approach of this thesis is to use constraints to represent the arrangement of layout elements. Constraints are an artificial intelligence technique that specifies values as relationships rather than hard-coded numbers. This technique has been used successfully by other researchers to encode graphic design knowledge. [MACN 90], [WEIT 88]. Figure 1.3 shows some relationships in a layout that can be expressed using constraints.

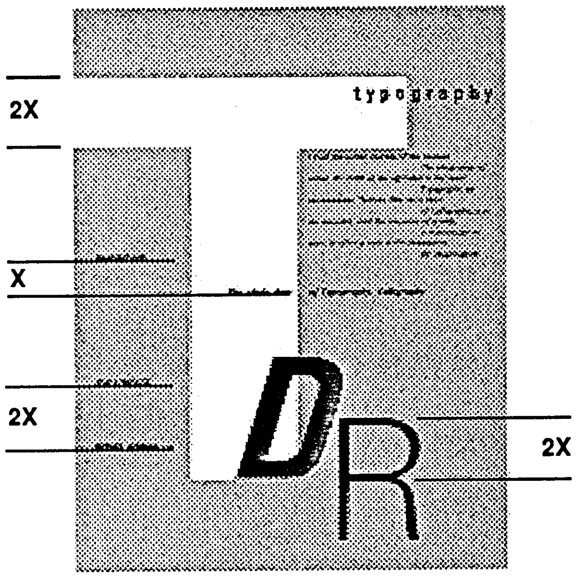


Figure 1.3 The layout elements have proportional relationships that can be represented in the system using constraints. Reproduced from [CART 85].

General Design Knowledge

The third aspect of this thesis approach is to represent general knowledge about layout design. Because it is difficult to express design knowledge as general rules, this knowledge is limited. This limited set of knowledge includes properties of graphic elements and layout grids, and rules of text legibility. A taxonomy of layout elements has been developed as part of this research. The taxonomy contains descriptions of the type of elements that win a layout and the properties of these elements. Figure 1.4 shows the taxonomy and graphic element properties. This design knowledge is based on the author's experience, anecdotal evidence, and cited references.

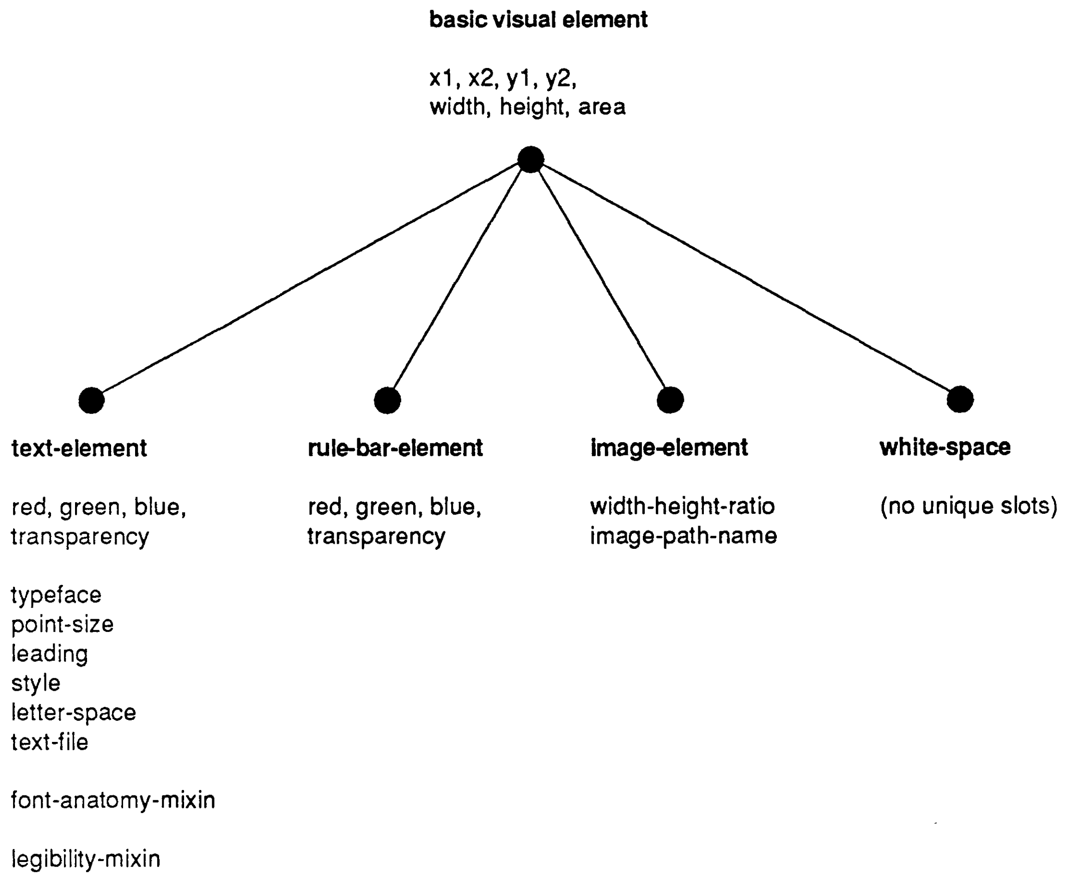


Figure 1.4 The taxonomy of layout elements that is represented in LIGA.

This information is stored in LIGA's general knowledge base. This general design knowledge proves to be very powerful in the system. It allows the representation of expert graphic design knowledge. In particular, the representation of layout grid properties and methods for constructing grids are included in the system. These layout grids will have knowledge of the information that is placed on them.

The general knowledge base also includes detailed descriptions of typographic letterform. This information is often used by expert designers to make design decisions. Figure 1.5 shows the properties of letterforms that are represented in the prototype system.

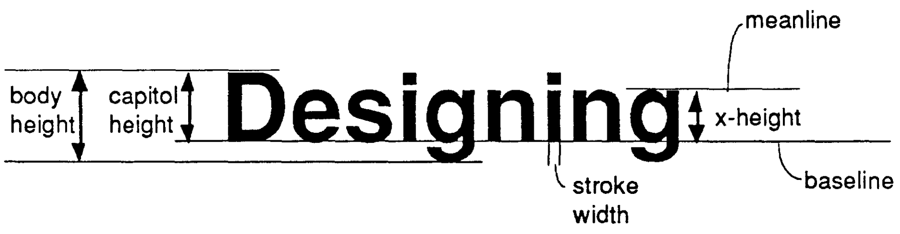


Figure 1.5 The properties of typographic letterform that are represented in LIGA.

Knowledge of content

The fourth part of this approach is to make layout decisions based on the logical structure an information types of the layout content. This approach, too, has been used successfully by other researchers in this field [MACK 86], [FEIN 88]. This content information is used as criteria for selecting a layout case that will be used to solve a new layout problem.

Document structure

This thesis includes the following. Chapter 2 presents an example of the prototype system capabilities. Chapter 3 gives a rationale for the approach. Relevant research is reviewed in Chapter 4. A technical explanation of how LIGA works is given in Chapter 5. Finally, the approach is evaluated in Chapter 6.

Chapter 2 **Overview of Project**

System Capabilities

The prototype system, LIGA, demonstrates the use of case-based reasoning and constraints for representing layout design knowledge. The goal of the system is to produce layouts that reflect the logical structure of the content and are legible for the given viewing context. The system can generate effective layouts of information as well as adapt layouts on the fly to accommodate changes in the presentation environment. Such changes might occur when a display window is resized by a user. An additional goal of the system is to ensure that an adapted layout is consistent in style with the original.

LIGA generates new layouts by modifying example layouts from its case library. Example layouts, or cases, are represented in the system using constraints. The constraints describe the visual relationship between the elements within the layout. The cases are not mere templates but robust representations of the underlying relationships between design elements. The cases also contain information about the logical structure and information types of the layout content. The system uses this content information to find an appropriate case for a new layout problem. The LIGA system adapts layouts by retracting dimensions that have changed in the layout, establishing new values for those dimensions, and recalculating values for the layout components based on the case constraints. The system has a general knowledge base that contains rules of text legibility and grid design. This knowledge ensures that text is readable and that the information will fit within the allotted area. The case constraints maintain an effective visual structure as well as the consistent style of a layout.

A specific example of interaction and results

The LIGA system can take text and image information as input and generate a layout for that information. Figure 2.1 shows some text and images that need to be laid out. This information describes the agenda for a Visible Language Workshop presentation attended by research sponsors. Figure 2.2 shows the layout that LIGA generated for this content. LIGA goes through two basic steps in solving a layout problem. These steps are: finding an appropriate example layout in the case library and applying the knowledge of that case to the content of the layout problem. The remainder of this section gives a description of the process that the LIGA system went through to generate this layout. An example is also given of how LIGA adapted the layout to accommodate a reduction in available display area.

```
Virtual Frame Buffers
and Tiled Displays

Presentation Agenda

6000x2000 Display
   Adaptive Layout
   Real-time News Display
   World Weather Data

Managing Visual Complexity
   High Level Interface
   Multi-layered Map Data
   Adaptive Text and Graphics

Intelligent Graphics
   Multimedia Editing
      and Designing
   Designing by Example
```

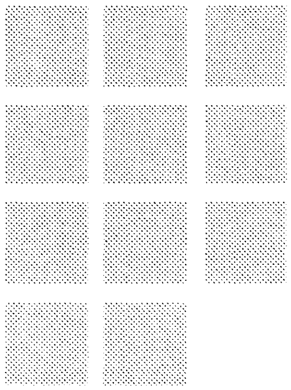


Figure 2.1 The text and image content for the Presentation Agenda layout problem.

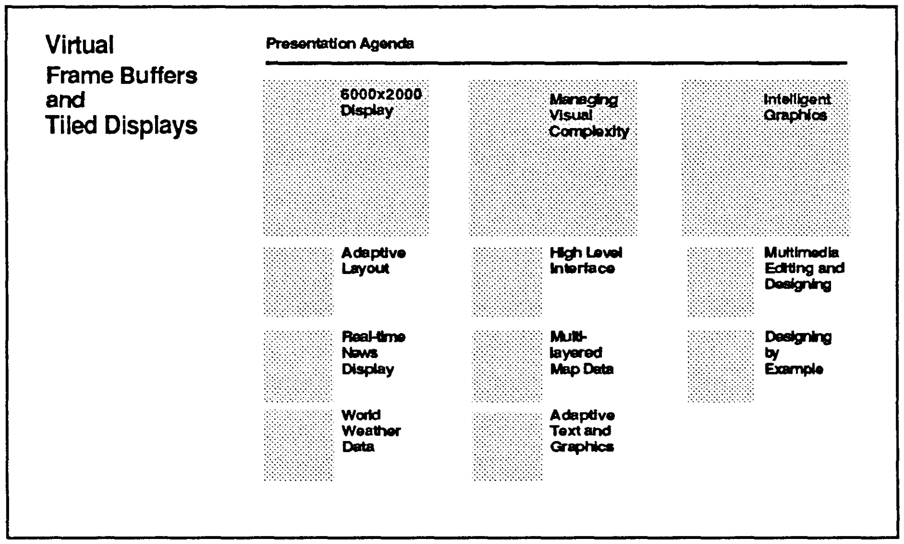


Figure 2.2 The layout that LIGA generated for the Presentation Agenda.

Input data

Before LIGA can solve the layout problem, it must be provided with the following data: the logical structure and information types of the content, the size of the available layout area, and the context in which the information will be viewed. The logical structure of the content describes the hierarchical and relevancy relationships between content elements. Information type specifies whether a piece of content is an image or text. The content of the Presentation Agenda consists of a title, a subtitle, and three categories of text with accompanying images. The logical structure of the content is a four-level hierarchy with relevancy relationships between specific text and image elements at the third and fourth levels. Figure 2.3 shows the logical representation of the Presentation Agenda content structure. The available layout area for the Presentation Agenda is 3500 x 2048 pixels. The layout will be part of a presentation – viewers will stand about six feet away from the display. Viewing context information is used by the rules of text legibility to determine readable text sizes.

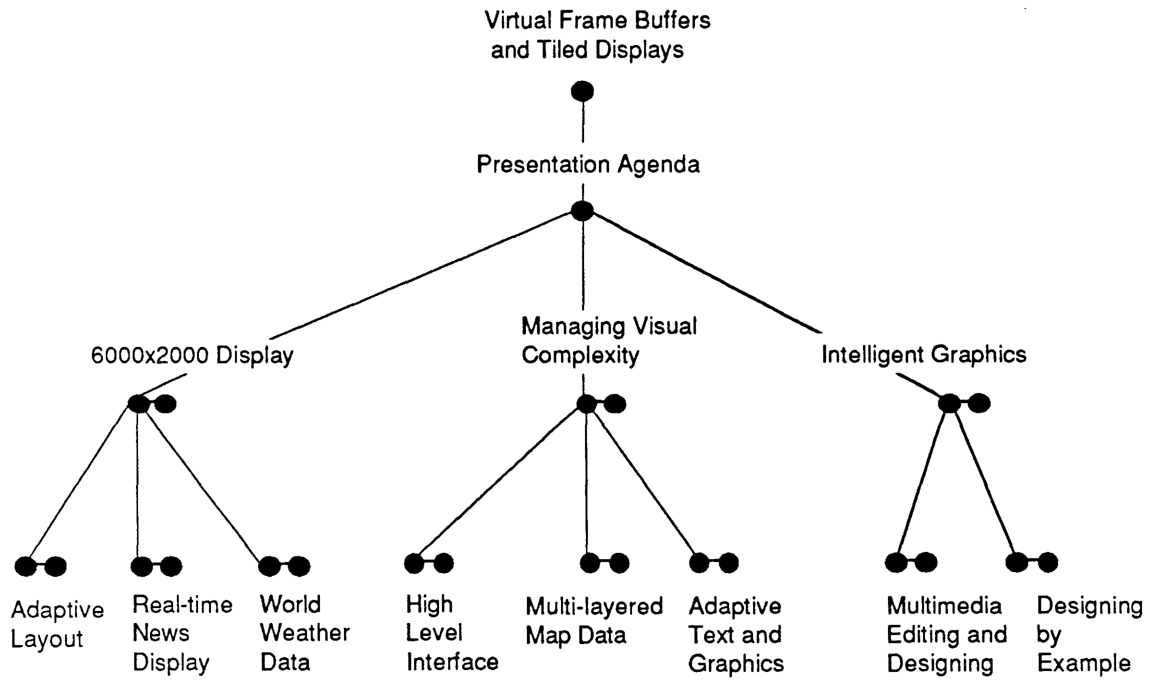


Figure 2.3 The logical structure of the Presentation Agenda layout content is a four-level hierarchy with relevant text and image pairs at the third and fourth levels.

Selecting a case

The first step in solving the layout problem is to search through the case library for a case that can be modified to the new content. LIGA searches for a layout with a logical structure and information types similar to the layout problem. The case library for this example contains five different layouts. Figure 2.4 shows the example layouts in LIGA's case library.

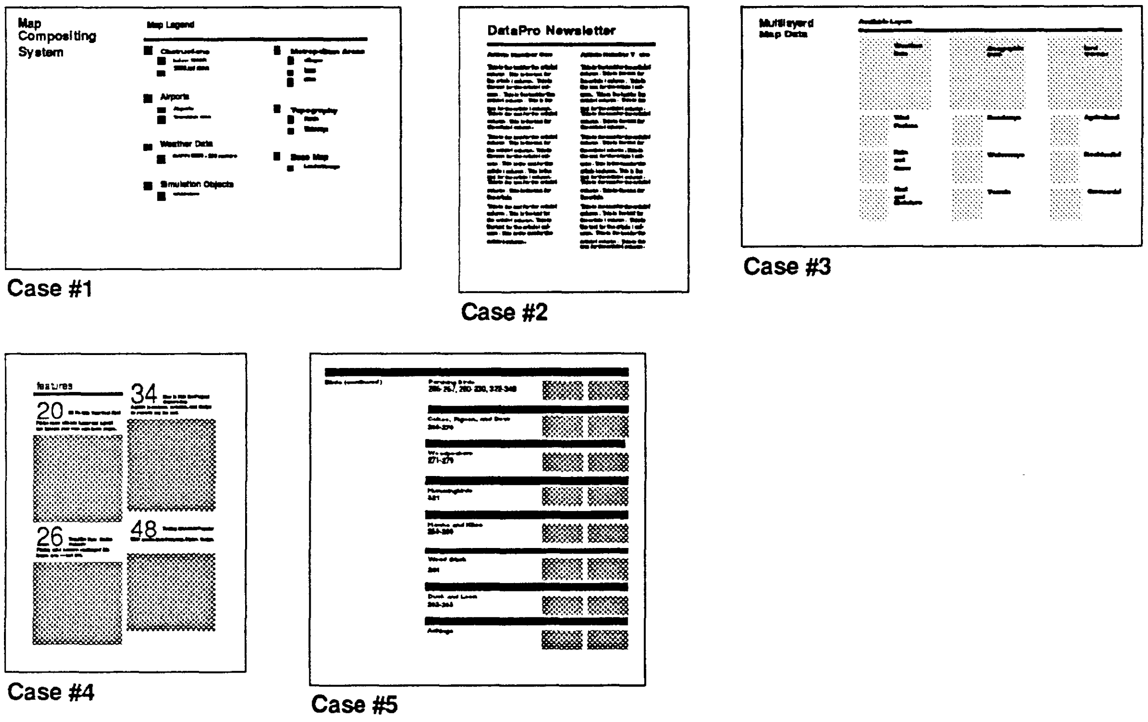


Figure 2.4 The layout cases in LIGA's case library.

As LIGA searches through the library, the first and second cases are rejected; they do not have enough content attributes that are similar to the problem layout. Although the first case layout has a similar logical structure, it does not use images. The second case has a different logical structure than the layout problem. LIGA selects the third case as appropriate to solve the problem. This case also has a three level hierarchy and uses text with accompanying images.

Applying case knowledge

Once a solution case has been found, the knowledge from the case is applied to the unformed content. This knowledge consists of functions that generate a layout grid and graphic objects, and apply constraints to the attributes of those objects. The constraints, along with the system's general knowledge of text legibility, calculate the size, color, and position for each element in the layout.

Following are two examples of the constraints that are part of the knowledge for case #3. The LIGA system uses constraints to encode typical geometric relationships as well as relationships that are unique to graphic design. The relationships are maintained when a case knowledge is applied to a new problem and when a layout is adapted for a new display area. For example, figure 2.5 shows a typical geometric relationship in which image size is a function of the grid column width. Figure 2.6 shows a visual relationship that is unique to graphic design: the case layout specifies that the height of the graphic rule equals the width of the stroke of the title text. The LIGA system contains detailed representations of typographic letterform enabling refined design knowledge to be encoded.

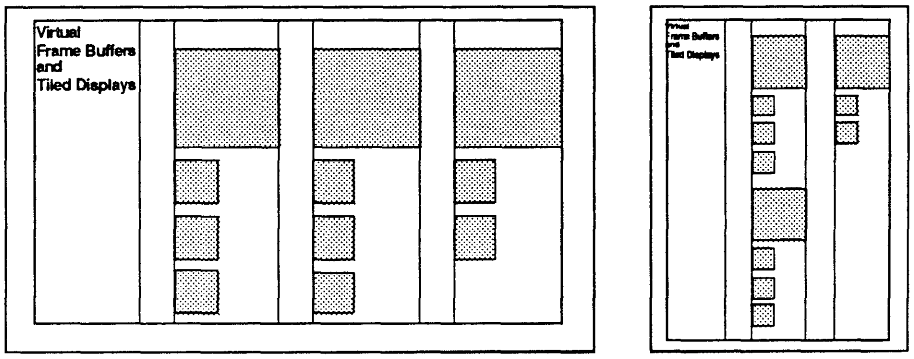


Figure 2.5 The underlying grid of the original (left) and adapted (right) layouts. The sizes of the images in the layout are constrained to the grid column width.

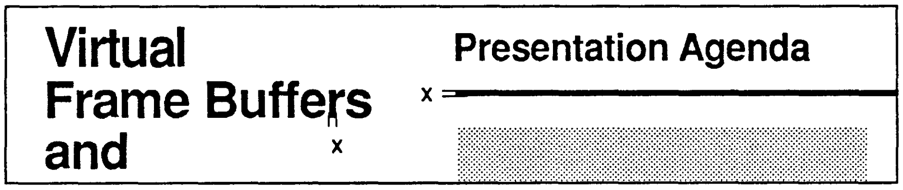


Figure 2.6 An example of a refined design constraint. The height of the graphic rule equals the width of the stroke of the title text.

Displaying a layout

When graphic objects have been generated for all pieces of the content, the layout is displayed. Figure 2.2 shows the layout that was generated by applying the knowledge from case #3 to the content of the Presentation Agenda.

Adapting a layout

The LIGA system can also adapt a layout if the display environment for that layout changes. In this example, that layout of the Presentation Agenda is displayed next to an enlarged image form the agenda. Figure 2.7 show this display. During the presentation to research sponsors, an additional image from the project list is enlarged and displayed on the screen. Figure 2.8 shows that this image occupies one half of the area formerly used for the project list. The layout area has been reduced and its proportions have changed; it is now tall and narrow rather than long and wide.

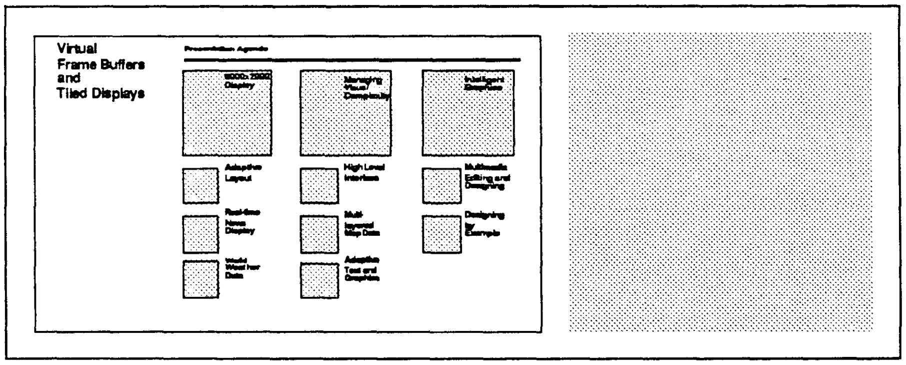


Figure 2.7 The Presentation Agenda is displayed alongside an enlarged image from the agenda.

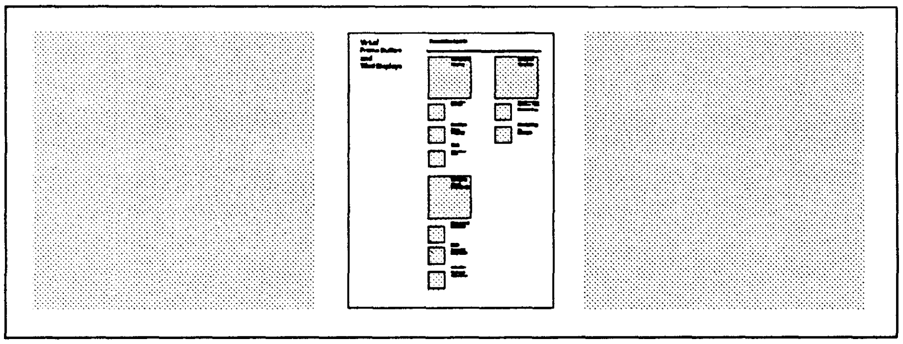


Figure 2.8 When a new image is enlarged and added to the display, the available area for the Presentation Agenda layout is reduced.

The LIGA system adapts the layout by retracting the values from width and height of the layout area, and replacing them with the new dimensions. This causes other dimensions in the layout to change. Initially the rules of text legibility reduce the text size so that information will fit in the smaller area. The system is careful not to reduce the size below the minimum threshold of legibility for the given viewing context. The rules of legibility are also responsible for reducing the number of columns in the grid from four to three. This change was made to ensure a wide enough column for legible text. The size and position of all other layout elements are then adjusted according to the case constraints. The constraints ensure that important design relationships are maintained, for example, the red rule is still the width of columns that have information underneath it, and the relative position of images with text labels is the same. The case knowledge along with the rules of legibility can redesign the layout to accommodate the reduction in display area. Figure 2.9 shows the resulting layout.

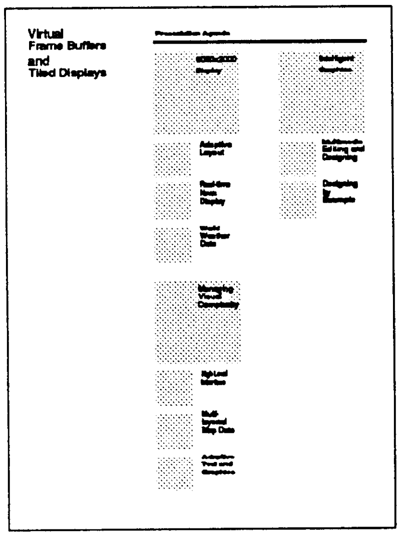


Figure 2.9 LIGA generated this layout to accommodate the reduction in display area.

Although the adapted layout is different from the original, the system has adhered to the goals of maintaining legibility, visual structure, and design style. The text size and positions of images have changed, but the visual structure still shows that there are three major categories of information. The graphic style of the layout has been maintained – the use of typeface, graphic rule, and white space is consistent with the original layout.

The case library in this example contains five layout cases. This number of cases is sufficient to demonstrate the use of content logical structure and information types for finding a case solution. The cases show a variety of logical structures and use of text and images. But the variations are not so disparate that the matching strategy is not challenged. Full implementation of such an automatic layout system would require the development of a more extensive case library. Chapter 6, "Evaluation and Conclusion," discusses how the matching strategy might be further developed to sort through a very large case library.

A detailed description of how LIGA generated the layouts that are presented in this chapter can be found in Chapter 5, "How It Works."

Chapter 3 **Approach**

A number of researchers have investigated the problem of encoding graphic design knowledge for the automatic generation of visual information displays. Most of this work has focused on chart and diagram design. The research presented in this thesis focuses on encoding knowledge about the layout of text and image information.

Encoding graphic design knowledge is difficult

The need for computer systems that contain design knowledge was identified as soon as the technology was developed to generate, transmit, and display information rapidly. Researchers have found it difficult to develop expert systems that contain visual design knowledge. Expert systems are developed by obtaining knowledge from a human expert about a particular domain and encoding that knowledge into a form that a computer can use to solve problems [LUGE 89]. This development process "breaks down" for domains, such as graphic design, that involve visual knowledge [COOP 89] [LIEB 88]. A predominant type of expert system is a rule-based system where knowledge is represented at a high level in the form of if-then rules. For example, a car repair expert system contains the rule "if the engine is getting gas, and the engine will turn over, then the problem is the spark plugs" [LUGE 89]. Although expert graphic designers can easily generate and critique visual examples, it is difficult for them to formulate their knowledge into if-then rules like the one above. Many design rules are context dependent. The rule "if the text is more important, then make it bigger" would not alone solve a layout problem. Other factors such as color, position, weight, and amount of text would also affect decisions about how to specify more important text.

There have been attempts by designers to formalize design knowledge. These attempts often come in the form of books which, just like the experts that are interviewed in person, present interesting visual examples but do not offer a cohesive theory or methodology. The authors of these books will offer design principles but this knowledge is not prescriptive advice that can be encoded into an expert system [CASN 91].

The difficulty for designers in formulating higher level rules stems from both the visual nature and the maturation of the discipline. Graphic designers use visual knowledge that is complex and not well understood even by visual perception researchers. In addition, graphic design as a profession is relatively young. The entire history of the field can be spanned by living memory. Many of the founders of graphic design, some of who are still practicing, did not acquire their knowledge through formal graphic design training but by a combination of training in fine arts, talent, and intuition [HURL 77]. It is just within the last decade that the graphic design profession has started to move toward formalizing its theories and methodologies. One impetus for this move toward formalization is the advent of information and artificial intelligence technologies [COOP 89].

The Approach

This thesis presents an approach to representing graphic design knowledge about text and image layout. The primary computational techniques used in this approach are case-based reasoning and constraints. Case-based reasoning is a problem solving method that makes decisions based on prior examples or "cases." This method is particularly appropriate for domains such as graphic design where it is difficult to represent knowledge as discrete rules. In the approach presented here, example layouts are used as the cases. The visual knowledge for each layout case is represented by using constraints. Constraints are an artificial intelligence method for representing relationships between variables. This technique is appropriate for representing the modular visual relationships in a layout. A case-based reasoner requires that features of the input problem and cases be characterized. These features are used to match problems with appropriate case solutions. The approach is to represent the logical structure and information types of the input problem content. The approach also represents general knowledge about layout design. This general knowledge includes rules of text legibility, knowledge of layout grids and knowledge of the types of graphic elements that will appear in a layout. The general knowledge is used in conjunction with the specific layout examples to solve new layout problems.

Case-based reasoning

Graphic design is not the only discipline that uses knowledge that is difficult to express at a higher level. Many domains involve knowledge that is either too difficult to acquire from an expert or is too complex to be formalized as rules.

An artificial intelligence technique, case-based reasoning, has been developed by Schank, Kolodner, and Reiesbeck [RIES 89] to address these difficult knowledge acquisition and representation problems. Case-based reasoning is a problem solving method that reasons, or makes decisions, based on prior examples. A case describes a problem and an appropriate solution. New problems are solved by identifying relevant cases from a case library and adapting them to the new situation. MacNeil [MACN 90] proposed the use of case-based reasoning to represent design knowledge and asserts that this method is a good match to the natural processes used in design.

For this research, the cases are example layouts of information. In order for a case-based reasoner to find the appropriate case, the system must know what features of input problems and cases to compare when searching for an appropriate match. When an appropriate case is found, the knowledge from that case is applied to solve the new problem. The approach of this research is twofold: to represent the case-specific design knowledge using constraints, and to match input problems to cases based on the logical structure and information types of the layout content.

Representing visual relationships via constraints

The visual relationships between elements within a case layout are represented using constraints. Although it may be difficult for a designer to explain why she made a certain image a particular size and placed it in a particular position, the visual decisions are not arbitrary. The size and placement of graphic elements are often based on some sort of modular system. A module is a unit of measure that is combined and repeated to create a whole. Modular relationships can be found in molecular, crystal, and cellular structures. Designers try to create a similar order in layout design [GERS 73] [MEGG 89]. When designing a layout, a designer will try to create an orderly combination of related parts. The order may be based on an underlying grid or on a relationship among the graphic elements themselves. Figure 3.1 shows the underlying modular relationship between the elements in a layout. The basic module is "x." The size and placement of other elements in the layout are a function of this module. The height of the stroke of the letter "T" is two times the module "x," the distances between lines of text are "x" and "2x," and the upper part of the letter "R" is "2x."

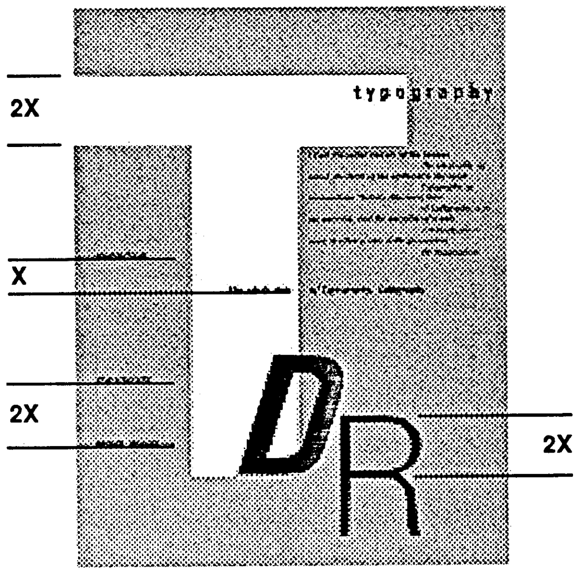


Figure 3.1 The elements in this layout have underlying modular relationships. Reproduced from [CART 85].

An existing artificial intelligence technique that can be used to represent these kinds of modular visual relationships is constraints. Constraint systems allow one to represent relationships between variables and the relationships can be viewed in several ways. For example, the relationship "a + b = c" is also viewed by the system as "a = c - b" and "b = c - a." When any two of the three values are available to the system, the third will be computed.

The following shows how constraints can be used to represent the design of layout elements. Figure 3.2 shows a text element and image element from a layout case. The text lies on top of the image and is placed in the upper left hand corner of it. The layout of these two elements can be described by the following relationships:

- the left edge of the text = the left edge of the image + 1/2 the width of the image.
- the top of the text element = the top of the image element plus the text size of the text element.
- the right edge of the text element = the right edge of the image element.
- the bottom of the text element = the bottom of the image element.
- image width = image height.

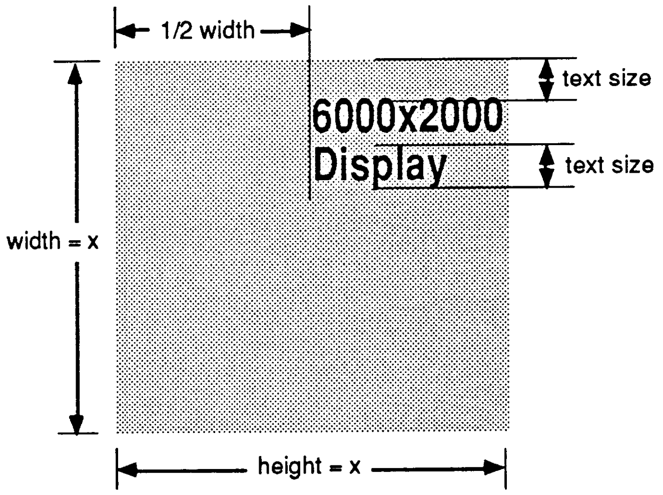


Figure 3.2 A text and image from a layout case. The visual relationship between the two elements can be described by constraints.

If these relationships were encoded into a system as constraints, the system would be able to maintain the design relationships when the size of the text or the size of the image changed. This research proposes the use of constraints to represent the visual relationships between graphic elements.

Logical structure and information types of layout content

In order for a case-based reasoner to find a solution case, the input problem must be characterized. The relevant features of the problem must be made explicit. The system must also know how to compare the features of the input problem to the cases in order to find an appropriate solution. The approach of this research is to represent the logical structure and the information types of the layout content and use these characteristics to match input layout problems to case solutions. Logical structure describes the hierarchy and relevancy relationships between content components. Hierarchy describes superior and subordinate relationships. Relevancy describes which pieces of content are relevant to one another. Images will often have relevancy relationships to the pieces of text that describe them. Figure 3.3 shows two layout elements that have a relevancy relationship. The image is neither superior nor subordinate to its label, but is relevant to it. Information type is also used to describe input content. Information type specifies whether a piece of content is text or an image.

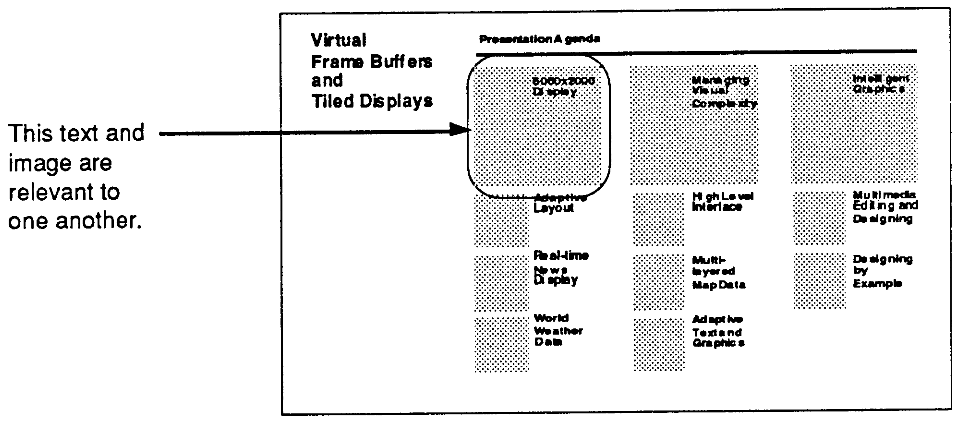


Figure 3.3. This text and image have a relevant relationship, not a hierarchical relationship.

A designer uses information about content logical structure and information types to make layout decisions. One goal of layout design is to make the visual structure of the layout reflect the logical structure of the information. The designer also handles text and images differently in a layout so it is important to represent this information.

There are other attributes of content such as subject matter and appearance of images, that designers use to make layout decisions. These aspects of content will not be addressed in this thesis.

General design knowledge

Although most graphic design knowledge is context dependent, there is some higher level knowledge about information layout design. This knowledge includes knowledge of layout grids, rules of text legibility, and knowledge of the types of graphic elements that will appear in a layout. Although this knowledge cannot be expressed as if-then rules, it is applicable to many layout situations and can be represented independently from the layout cases.

Grids

Graphic designers use grids as an ordering system to organize the presentation of visual information. A grid is a common method for creating modular relationships in a layout. A grid divides a two-dimensional space into equal columns and equal rows. The rows and columns are separated by an intermediate space, or gutter, so that columns of text do not touch each other and legibility is preserved. The use of the grid makes it easier for a designer to give the layout a rational organization. This orderliness helps the viewer perceive the information as being credible. "Information presented [clearly and logically] will not only be read more quickly and easily but the information will also be better understood and retained in memory." [MULL 81].

A grid used for layout design has unique properties so it is necessary and useful to represent this knowledge. The general definition of a grid is "a pattern of horizontal and vertical lines forming squares of uniform size..." [WEBS 85]. A layout grid has more specific attributes than those described in the dictionary definition. Figure 3.4 shows a typical layout grid. In addition to a pattern of horizontal and vertical lines, a layout grid has margins, gutters, columns and/or rows. The grid units are not required to be square, and there are not necessarily both horizontal and vertical grid divisions. Although layout grids vary in proportion and in the number of horizontal and vertical divisions, the grid components and method of construction remain constant for all layout grids.

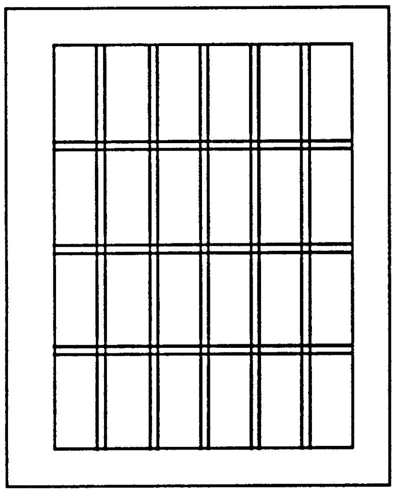


Figure 3.4 A typical layout grid provides margins and divides the layout area into units of equal measure.

Text legibility

The legibility of text is affected by many factors. Some of these factors are letter size, line length, color, line spacing, letter form, justification, and audience. The term legibility, as used here, describes not only whether letters and words can be discerned, but how comfortable the text is to read. Poorly laid out text can cause the reader fatigue or interfere with comprehension of the material. [CRAG 71]. This research focuses on two factors that affect legibility: letter size and line length. These two aspects are chosen because the rules of legibility for these factors are well established and can be represented with existing artificial intelligence technology.

Text size

The readable size of text depends on the viewing context of the layout. For example, the text size used for magazine and newspaper articles is usually between 9 to 12 point. The text sizes that would be readable for a poster would be larger. The approach of this research is to represent the ranges of acceptable text sizes for different viewing contexts. Two viewing contexts have been defined for this research. These are "demo" and "application." The "demo" context refers to a situation where the information is presented to a group of viewers. This viewing context assumes a viewing distance of six to eight feet. The "application" context refers to a situation where the viewer is at a distance of about 12 inches, like that of someone using an application on a desktop computer. In general, the allowable text sizes for the "demo" context are larger than that of the "application" context, therefore input must specify the viewing context of the layout problem. This data about acceptable letter sizes is part of the general design knowledge base.

Line length

Line length affects how comfortable it is to read text. Lines of text must not be too short or too long. Lines that are too long require the reader to actually move his head to read the entire line. This causes fatigue and creates interruptions of thought while moving from line to line. Lines that are too short often break up words or phrases that are generally read as a unit.

There are no measurements for the minimum and maximum lengths of a line. The rules of legibility for line length concern the number of characters per line, not the actual measurement of the line. For print material, the maximum number of characters per line for readable text is 65. This figure is based on perceptual studies [CRAG 71]. A rule of thumb that designers use for minimum line length is to set a line at no less than x picas wide for type size x. For example, the minimum line length for 12 point type is 12 picas. This rule can be translated as a minimum of 27 characters per line. The minimum line length for a heading or title would be less than 27 characters per line.

Although these rules are based on knowledge of text legibility for print material, they are still applicable to text displayed on a computer screen. Further studies may reveal that the exact thresholds for legibility may be different for light display rather than a reflective display, but letter size and characters per line will still affect legibility [RUBE 88]. The specific thresholds can be adjusted based on conclusive research for a light display legibility, but the method of implementing these rules will be the same.

Knowledge of layout elements

Included in the general knowledge about text and image layout is knowledge of what types of visual elements appear in a layout. The elements that will appear in a layout will be one of three types: text, image, or abstract mark. The general knowledge includes the properties of each of these kinds of elements.

Although this knowledge seems like common sense, it needs to be made explicit in an intelligent computer system. The system needs to know that a color can be assigned to a piece of text or a graphic bar but not to a photograph. Likewise, an image will have a proportion that needs to be maintained regardless of desired modular relationships, but a graphic bar does not. Text has a typeface, but other layout elements do not have this property. In fact, text has many more properties than the other types of layout elements.

Text will appear in a layout as typographic letterforms. The general knowledge includes detailed knowledge of typographic letterforms. The purpose of having this information available in the system is to allow the expression of modular relationships based on these letterform properties. Designers often make size and placement decisions of layout elements based on the measurements of the typographic letterforms being used in the layout. For example, in the layout shown in figure 3.5, the designer has specified that the height of the graphic bar is equal to the width of the stroke of the title text. Figure 3.6 shows a layout in which the image is aligned with the top of the lower-case letters of the accompanying text. The designer chose to align by this measures because the stronger horizontal axis of the text is created by the top of the lower-case letters, not the upper-case letters. The lower-case letters appear more frequently than the upper-case letters. These types of decisions are made to create visual unity within a layout and are consistent with the concept of visual modularity that was discussed earlier in this chapter.

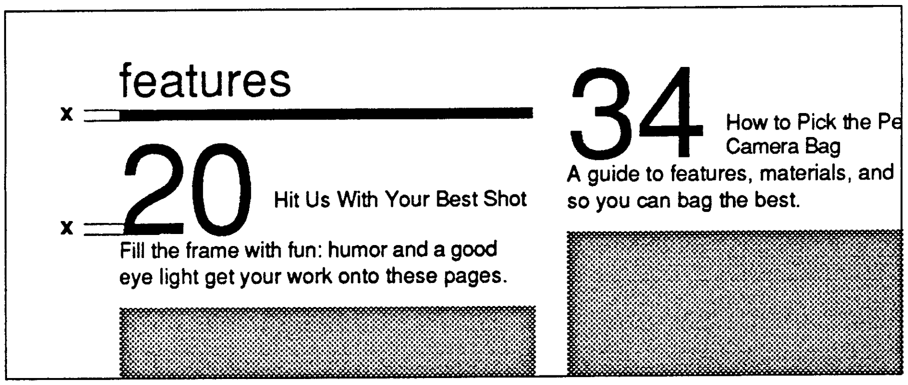

features
x
x
20
Hit Us With Your Best Shot
Fill the frame with fun: humor and a good eye light get your work onto these pages.
34
How to Pick the Pe
Camera Bag
A guide to features, materials, and so you can bag the best.

Figure 3.5. In this layout, the designer has specified that the height of the rule is equal to the width of the stroke of the large numbers.

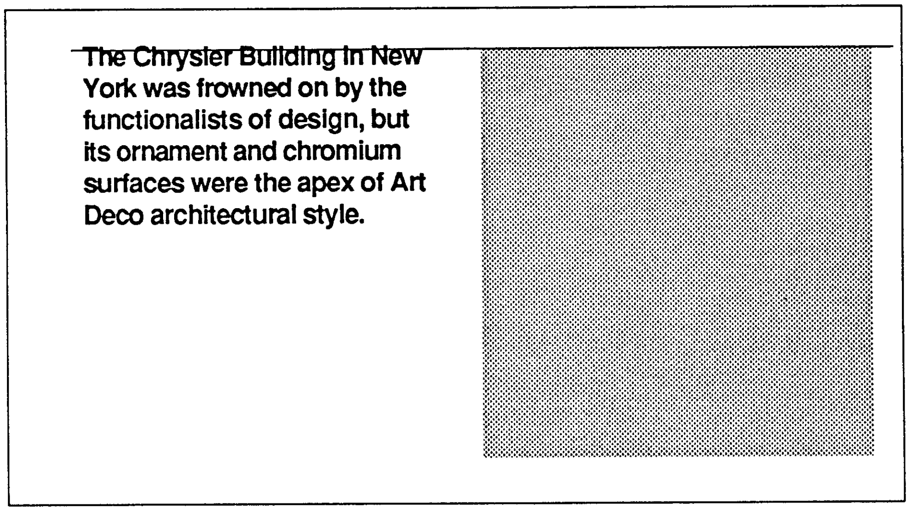

The Chrysler Building in New York was frowned on by the functionalists of design, but its ornament and chromium surfaces were the apex of Art Deco architectural style.

Figure 3.6 In this layout, the designer has specified that the top of the lower-case letters aligns with the top of the accompanying image.

Figure 3.7 shows the parts of the letterform that are represented in system. Below are definitions of each of these parts of the letterform.

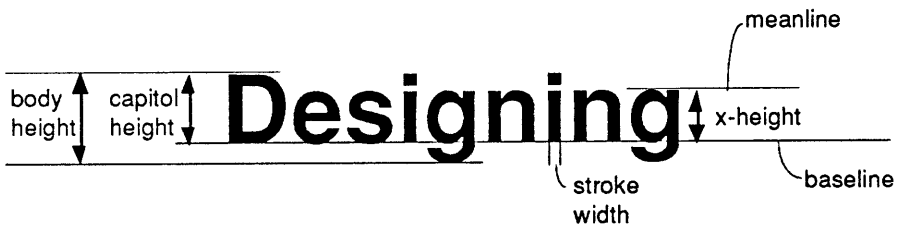


Figure 3.7 The parts of the typographic letterform.

Capital height – the height of the capital letter of a typeface.

X-height – refers to lower-case letters only. It is the height of the main element of the lower-case letterform and is equivalent to the lower-case "x." X-height is significant because it creates the strongest horizontal axis in a line of text that is set in upper and lower case.

Meanline – the top of the lower-case letters.

Stroke-width – the thickness of the stroke of the letterform. This measure is used primarily for sans serif faces where the stroke is more nearly uniform in thickness. This is equivalent to the width of a lower-case "i."

Baseline – the line where the base of the letter sits.

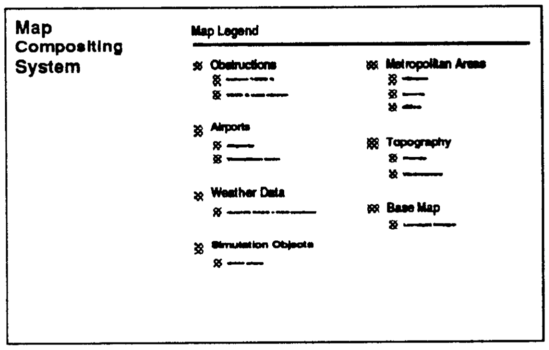
Body-height – the distance between the tallest ascender and the longest descender. No individual letter fills the entire body of type. Traditionally, the point size of type is equal to the body height. Current digitized and outline typefaces do not always follow this convention, so it is necessary to represent this value independently of type size.

Role of the case library

The approach described here uses case-based reasoning to represent layout knowledge. A case library will contain several example layouts. It is important to consider what types of layouts are represented in the case library. The choice of example layouts will depend upon the use of the intelligent layout system. If the system was being used to generate personalized newspapers, the style of all the layouts that the system produced should be the same. If the system was being used by a designer to get ideas of how to solve a layout problem, then the case library should have a variety of styles among the case layouts.

A layout style is the consistent use of grid structure, typography, colors, and position of key elements. A layout style will be developed by a designer to unify a series of layouts and help identify them as being related. A commonly known use of a visual style is corporate graphic program. The example shown in Chapter 2 uses a case library where the case layouts all share the same style.

Although graphic style is not represented explicitly in this research, the organization of case libraries with layouts of the same styles approaches this issue. In fact, when layout cases share the same style, they can sometimes share the same specific knowledge. Figure 3.8 shows two example layouts that are used as cases. The layouts display different content but both have a title with a subtitle. The same graphic conventions are used in both layouts to show the title and the subtitle. This layout knowledge about the title and subtitle can be represented once in the system and shared by more than one case. This is a way of representing the design knowledge that the designer intended the title/subtitle format to be used in several situations.



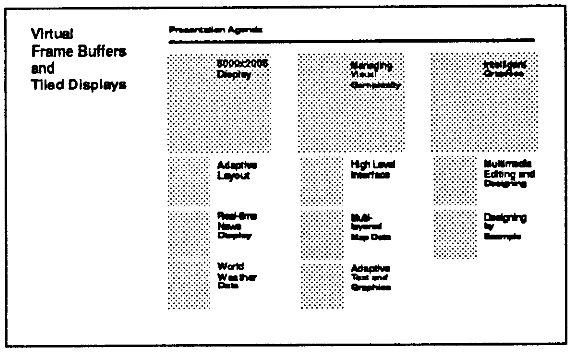


Figure 3.8 Both of these layouts use the same format to show the title and subtitle of the content.

Although the approach presented in this thesis does not attempt to represent all types of layout design knowledge, the knowledge that is represented can be used to generate good layouts. Chapter 5, "How It Works," discusses how the layout knowledge that was described in this chapter is encoded into the prototype system and how the system solves layout problems.

Chapter 4 **Related Research**

Other researchers have investigated the problem of encoding graphic design knowledge for the automatic generation of effective visual presentations. Most of this work has focused on chart and diagram design. Exemplary works on the encoding of knowledge about chart and diagram include: [KOSA 91] [MARK 90] [MACK 86] and [KAMA 91]. These researchers have taken many different approaches to the problem of encoding graphic design knowledge. The approaches range from the application of language theory to the development of genetic algorithms that generate diagram layouts.

Although this research shows successful methods for representing graphic design knowledge for generating network diagrams and bar charts, it is difficult to apply these strategies directly to page layout design. The design conventions used in page layout are not as well matched to mathematical methods, such as the Least Squares Method, as are the design conventions for network diagrams, flow charts, and bar charts. In addition, these domains have content relationships and graphic conventions that are more readily defined using existing artificial intelligence technology. The purpose of most charts is to show quantitative relationships which are a natural for computational representation and manipulation. Text and image layout rarely convey quantitative relationships.

There are three pieces of research that are more closely related to the problems of page layout design: [BEAC 83] [MACN 90] and [FEIN 88]. The following section describes some of these different approaches.

Marks and Reiter: "Avoiding Unwanted Conversational Implicatures in Text and Graphics"

Marks and Reiter [MARK 90] describe their Automated Network-Diagram Designer (ANDD). The system takes the content of a diagram as input and assigns appropriate visual attributes to the components. A theory of language conversation is used as a model for evaluating and correcting the design. This model's goal is to avoid "unwanted Gricean implicatures." These unwanted implicatures, as applied to graphics, are redundant visual attributes that imply meaning but have none. This implied but unintended meaning can confuse the interpreter of a diagram. This model served as a successful method for encoding the proper use of graphics to generate effective network diagrams.

Kosak, et al: "New Approaches to Automating Network-Diagram Layout"
Kosak, Marks, and Shieber [KOSA 91] present a parallel genetic algorithm which generates alternative layouts for network diagrams. Design alternatives are generated and mated until a near optimal layout is found. The mating is directed by a worth ranking. Alternate layouts are ranked on syntactic, organizational, and aesthetic criteria. This ranking is used to determine which layouts should be mated in the hopes of successively producing improved layouts. Although this approach does not guarantee that an optimal layout will be generated, the worth ranking proved to be a "robust mechanism" for directing the search.

Weitzman: "Designer: A Knowledge-Based Graphic Design Assistant"
Weitzman's "Designer" [WEIT 88] is a system that critiques the design of a two-dimensional interface and suggests design alternatives. The alternatives are meant to improve the design by making it more consistent and visually more effective. The system uses knowledge of visual communication principles to link visual relationships to content relationships. For example, "Designer's" principle of significant difference requires that if two elements are significantly different in meaning, the visual representation of the elements should also be significantly different. The principles are applied by using constraints. The system uses both continuous and discrete constraints. Discrete constraints restrict a variable to be one value. Continuous constraints allow a variable to span a range of values.

Kamada and Kawai: "A General Framework for Visualizing Abstract Objects and Relations"
Kamada and Kawai [KAMA 91] describe their visualization framework which determines a layout of graphical objects under geometric constraints. The importance of this work resides in the approach to resolving constraint conflicts. This research introduces the concepts of rigidity of constraints. "Rigid" constraints must be satisfied exactly, "pliable" constraints may be satisfied approximately. Conflicting constraints are resolved using the Least Squares Method on the pliable constraints. Kamada and Kawai show how this method can be applied to constraint conflicts in tree and network diagrams. Although the Least Squares Method works for such diagrams, it is not an appropriate strategy for solving constraint conflicts in page layout problems.

Mackinlay: "Automating the Design of Graphical Presentations of Relational Information"

Mackinlay [MACK 86] describes his APT (A Presentation Tool) which generates effective charts and graphs. The approach draws on the graphic design principles of Bertin and on perceptual studies to define graphical languages. These languages describe the "effectiveness" and "expressiveness" of a graphical presentation by associating visual properties with a ranking of their most effective use. For example, size is more effective than color for expressing quantity. This higher level approach of encoding design knowledge is useful when applied to chart and graph design. It would be more difficult to apply these principles effectively to the design of text and image presentations, which do not have as clearly defined relationships between elements. Although this approach alone would not serve to solve page layout problems, it would be interesting to discover what complementary knowledge is needed to do so.

Beach and Stone: "Towards High Quality Illustration"

Beach and Stone [BEAC 83] describe their system for generating scientific illustrations. This research extends the concept of typographic style sheets to the format of illustrations. A hierarchical structure is used to describe the semantic components of an illustration, then varying graphic styles can be applied to it. This research addresses the issue of viewing context by providing a different style for different media output. For example, the x and y axes in a graph illustration will be drawn with a thicker line weight for a 35mm slide than for a journal article. The approach presented in this thesis also uses knowledge of viewing context to influence the design choices.

MacNeil: "Adaptive Perspectives: Case-Based Reasoning with TYRO, the Graphic Designer's Apprentice"

MacNeil [MACN 90] proposes the use of case-based reasoning to represent graphic design knowledge. His research focuses on how a designer might enter and refine case knowledge. MacNeil describes a system, TYRO, that allows a designer to present and revise a design, helping the system to properly generalize case knowledge. The example design domain used is the layout of a subway map. MacNeil uses a smaller granularity for case representation than that proposed in this current thesis. In TYRO, each case contains knowledge of how to do a specific kind of task for the map layout problem. For example, a case would have knowledge of adding a new stop to an existing subway map. The underlying representations of visual relationships in the cases are constraints. MacNeil's use of case-based reasoning served as an important inspiration for the development of the approach described in this thesis.

Feiner: "A Grid-based Approach to Automating Display Layout"
Feiner [FEIN 88] addresses the problem of encoding page layout expertise in a prototype that uses knowledge of a layout grid. The system knowledge resides in a grid prototype that handles text and image content of a certain structure and uses rules of legibility and viewing context to determine an appropriate column width and text size for the layout. This thesis shares many approaches with Feiner's work. The approach presented in this thesis also uses knowledge of layout grids, text legibility and viewing context, adding knowledge of typographic letterform, and the encoding of design styles through the layout cases. In addition, LIGA can alter the number of columns in a grid depending on text legibility criteria.

When evaluating his system, Feiner notes that his system cannot produce a layout in a situation where the minimum text size has been reached and the text blocks are still too large to fit within the display area. In its current implementation, LIGA would also fail in this situation. Future work in this area would include incorporating knowledge of design compromises.

Chapter 5 **How it Works**

Overview

The prototype system, LIGA demonstrates the use of case-based reasoning and constraints for representing knowledge of text and image layout. LIGA consists of six components: a component that represents content structure, a general knowledge base, a case library, a matching components, a constraint system, and a display component. The constraint system and display components were existing, all other components were developed by the author for this research. Figure 5.1 shows the components of the system and how they relate to one another.

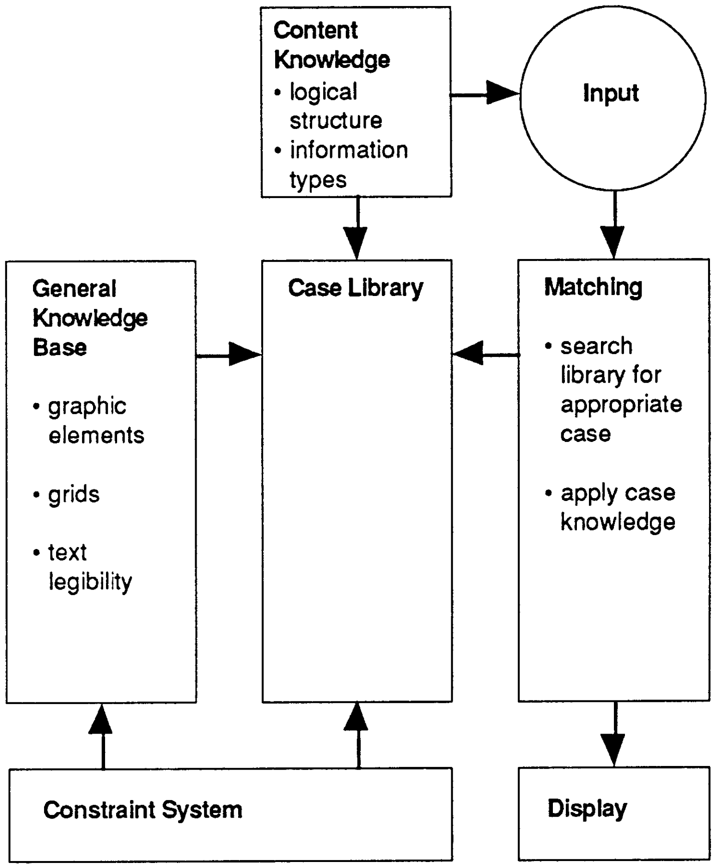


Figure 5.1 The system components of LIGA.

Implementation

LIGA is implemented in an object oriented programming language which provides a hierarchical structure for object class definitions. This allows the definition of classes and subclasses where subclasses inherit both attributes, or "slots," and behaviors, or "methods." The LIGA system is written in Common LISP and CLOS (Common LISP Object System), and runs on a UNIX workstation. Layouts are displayed using BadWindows, a C-based system developed at the Visible Language Workshop. Information is transferred from the LIGA system to the display system through a LISP-C interpreter.

How it works

Input to the system is the content and context of an unsolved layout problem. The matching rules and retrieval functions search the case library for a layout case that has content attributes that are similar to the layout problem. The general knowledge base contains knowledge of graphic elements, layout grids, and text legibility. This knowledge, along with the constraint system, is used to represent the specific design knowledge of the layout cases. When an appropriate layout case is retrieved, the knowledge from that case is applied to the layout problem, and specifications for a new layout are generated. The new layout is then displayed.

A specific example

The remainder of this chapter gives a detailed explanation of the steps that LIGA goes through to solve a layout problem. The layout problem and solution that were described in Chapter 2 are used as the example for this section. Chapter 2 describes a layout problem in which some text and images need to be laid out for a presentation agenda. Figure 5.2 shows the original content for the Presentation Agenda, the layout that LIGA generates for it, and the layout that LIGA generates to accommodate a smaller display area.

```
Virtual Frame Buffers
and Tiled Displays

Presentation Agenda

6000x2000 Display
    Adaptive Layout
    Real-time News Display
    World Weather Data

Managing Visual Complexity
    High Level Interface
    Multi-layered Map Data
    Adaptive Text and Graphics

Intelligent Graphics
    Multimedia Editing
      and Designing
    Designing by Example
```

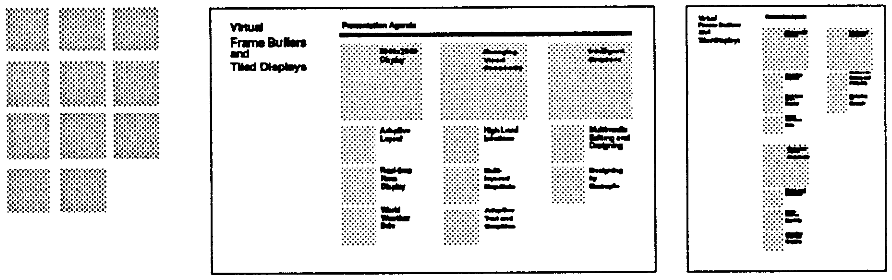


Figure 5.2 The content for the Presentation Agenda and the layout that LIGA generated for it.

The following aspects of how the system works are described in this chapter:

1. Input to the system
2. Finding an appropriate case
3. Case knowledge representation
4. Generating a new layout
5. Adapting an existing layout
6. Displaying a layout

1. Input to the system

LIGA requires the following information as input to solve a layout problem: the display area dimensions, the viewing context for the new layout, and the actual content for the layout. The content must include information about its logical structure and information types.

The available display area for the Presentation Agenda is 2048 x 3500 pixels. The viewing context is "demo" – the information is to be viewed by a group standing six feet away from the display. The logical structure of the content is a four-level hierarchy of text with relevant text at the third and fourth levels.

Viewing context

LIGA allows one of two viewing contexts to be specified. These are "demo" and "application." The "demo" context refers to a situation where the information is presented to a group of viewers. This viewing context assumes a viewing distance of six to eight feet. The "application" context refers to a situation where the viewer is at a distance of about 12 inches; like that of someone using an application on a desktop computer. A class is defined which stores maximum and minimum legible sizes for a particular viewing context. Instances of these objects are part of the general knowledge base and serve to represent some of the system's knowledge of text legibility. In general, the allowable text sizes for the "demo" context are larger than that of the "application" context. The slots and values of the "demo" viewing context object are shown below.

```
*demo* <viewing-context object>
          :context "Demo"
          :min-size 30
          :max-size 80
          :default-size 50
```

Layout content knowledge

The case-based reasoning method that is used in this system requires that input problems be characterized and that this information be used to find appropriate case solutions in the case library. The matching strategy for LIGA is to find a case that has content logical structure and information types similar to the input problem. This strategy requires that the content attributes, logical structure and information type, be represented in the system for both the unsolved layout problems and layout cases. A class, *information-unit*, is defined which describes the necessary properties of a discrete piece of content. Both the input content and layout case content are represented as a hierarchy of these units. The *information-unit* class has methods that make inferences about the slot attributes of the *information-unit.*

Information-unit

The *information-unit* class is used to describe both unsolved layout problems and layout cases. An *information-unit* represents a discrete part of the layout content. The complete content of a layout is made up of many *information-units.* For example, in the layout shown in Figure 5.3, each discrete piece of text and each image would be represented by an information unit. Note that the graphic bar under the words "Presentation Agenda" would not be represented by an *information-unit* as it is not part of the original content. These *information-units* form a hierarchy that describes the content logical structure.

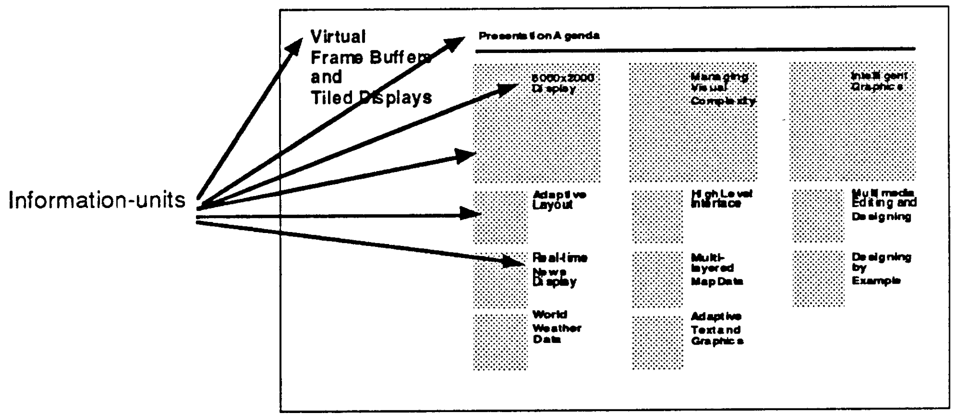


Figure 5.3 Each discrete piece of text and each image is represented by an *information-unit.*

The *information-unit* provides a complete description of a layout component. It contains the content itself as well as information about the content structure and graphic representation. Below are the slot attributes of an *information-unit.*

```
Information-unit:
content
information-type
supertopic
subtopic
relevant-to
layout-function
layout-object
```

The *content* slot contains the actual text string or image file of the piece of content. The value of the *information-type* slot will be one of the following: text, paragraph, or image. The *supertopic, subtopic,* and *relevant-to* slots are used to describe the logical structure of the content. The value of these slots will be a list of *information-units.* The *supertopic* and *subtopic* slots describe the hierarchical structure of the layout content.

The value of a *layout-function* slot will be a function that knows how to create a constrained layout element for its information unit. The resulting element is then stored in the *layout-object* slot. Layout cases have values for the *layout-function* and *layout-object* slots. A problem layout only has a value for these slots after the layout has been solved. Figure 5.4 shows the logical structure of the input layout content. Each node on the tree represents an *information-unit.*

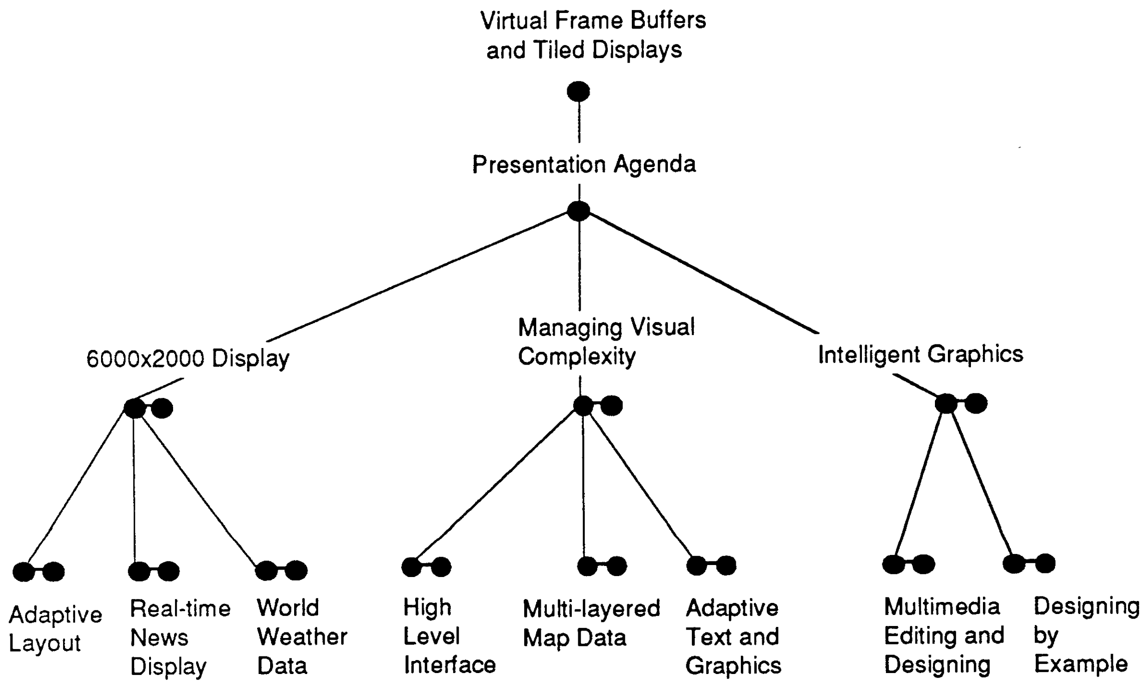


Figure 5.4 The logical structure of the Presentation Agenda content.

Relationship Inferences

The system makes inferences about relationships between *information-units.* For example, if *information-unit* A is the value of the *supertopic* slot of *information-unit* B, then the inference is made that A is a *subtopic* of B. *Information-unit* A is then added to the *subtopic* list of *information-unit* B. There is also an inverse relationship for the *relevant-to* slot. This inferencing allows one to specify only one direction of the hierarchy when initially instantiating the layout content. The inferencing on content structure relationships is necessary because matching is based on all content relationships for each *information-unit.*

2. Finding an appropriate case

LIGA's first step in solving a layout problem is to search through the case library for a case that can be modified for the new content. LIGA's case library contains five example layouts. For the Presentation Agenda layout problem, LIGA rejected the first two cases in the library and selected the third case to solve the layout problem.

The case library

The layout cases are organized in the library as a simple list. Figure 5.5 shows the organization of the cases and the case library. Because the *information-units* for an individual case are structured hierarchically, the entire case can be accessed through the top-level *information-unit* of the case. This list organization is sufficient for the current implementation of the system. As the number of cases increases, a more sophisticated organization of the case library will be required. Cases might be organized by information type, for example, so that it would be easier to find cases that contain images. Cases might also be organized by similar logical structure.

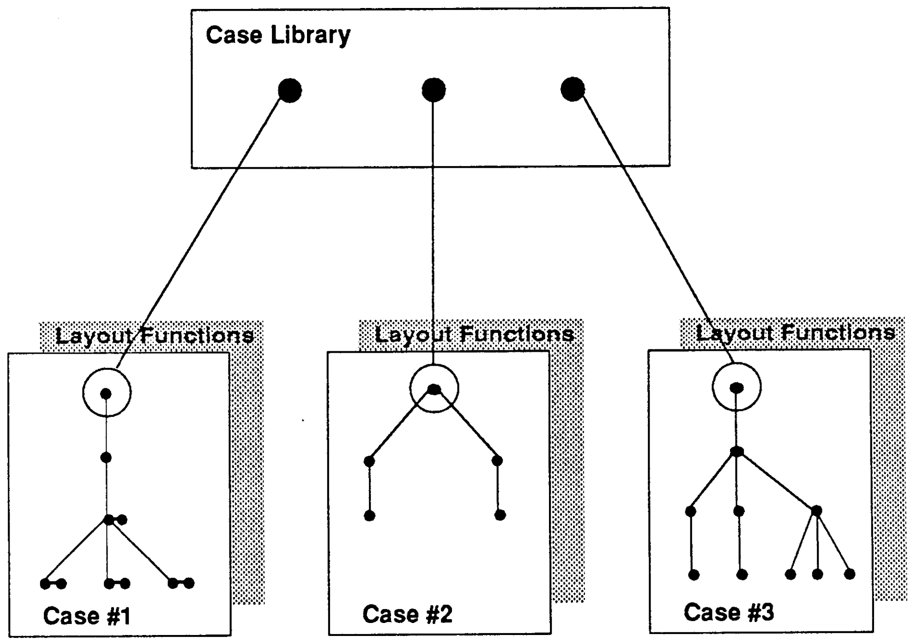


Figure 5.5 The case library is organized as a list of the top-level *information-units* of each case. The cases contain a hierarchy of *information-units* and a set of layout functions.

Searching through the case library

Because the library is organized as a list, the library search function simply goes through the list linearly. Searching stops when the first matching solution is found. A case matches a layout problem if the contents of both have an equal number of hierarchical levels and the *information-units* on each level match on *information-type*, *supertopic*, *subtopic*, and *relevant-to* slots.

Comparing input problem and layout case

When comparing a problem and a case, the search starts at the top of the hierarchy of the layout and continues down one branch until a test fails or the bottom of the tree has been reached. In this implementation, the matching function compares only one branch of the problem to one branch of the case hierarchy. This requires that the branches within the layout content hierarchy be of equal depth. This is a rudimentary matching function which could certainly be further developed to be able to search multi-depth hierarchies.

Comparing information-units

At each level of the hierarchy the *information-units* are compared. If any pair of *information-units* does not match, the case is rejected. The *information-units* are compared on the *information-type, supertopic, subtopic,* and *relevant-to* slots. *Information-type* slots match if their values are equal. That is, the *information-units* must be of the same *information-type. Supertopic, subtopic,* and *relevant-to* slots match successfully if they are both "nil" or both have values. The length of the *subtopic* slot list, for example, in each *information-unit* is not considered in the matching process. This means that a three-level hierarchy with two nodes on the second level will match a three-level hierarchy with four nodes on the second level.

The specific example

The case library for this example contains five layout cases. Figure 5.6 shows the example layouts in the library. While searching for a solution to the Presentation Agenda problem, LIGA rejects case #1 and case #2. LIGA selects case #3 as the solution.

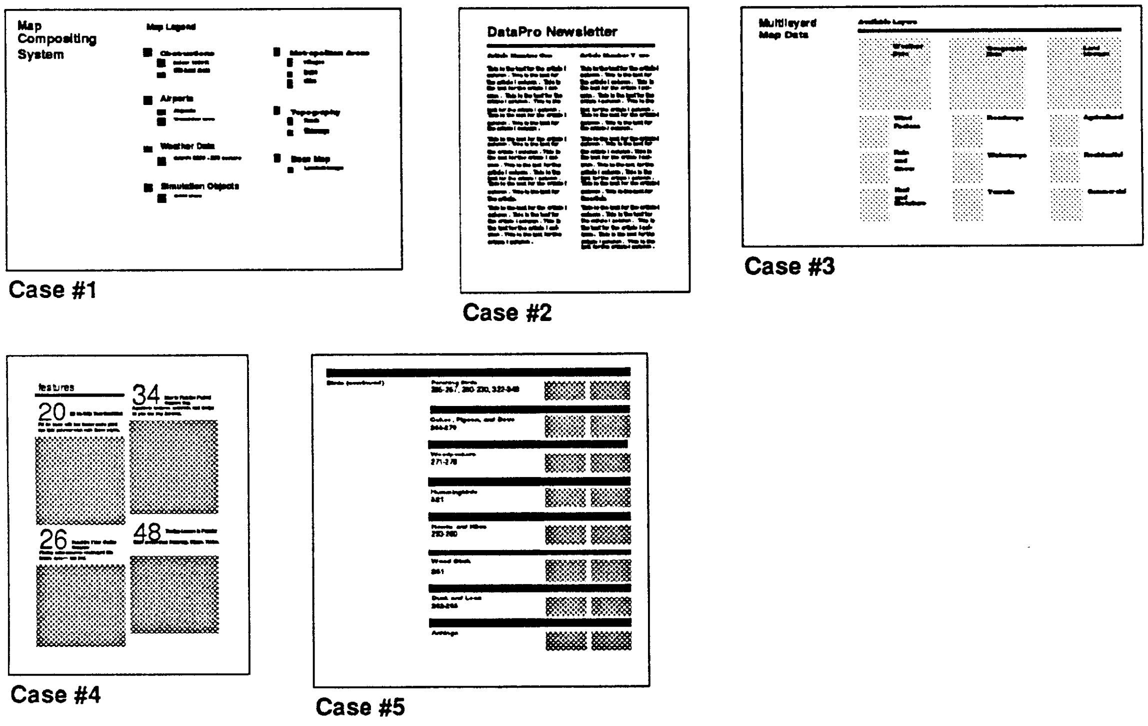


Figure 5.6 The example layouts in LIGA's case library.

Reject case #1

LIGA rejects case #1 because although both this case and the problem have four-level hierarchies of text, the first case does not have relevant images at the third and fourth levels. The search fails when the *information-units* at the third level of the hierarchy are compared. Figure 5.7 shows the search path for this case. The case *information-unit* does not have a value for the *relevant-to* slot while the problem *information-unit* does. Figure 5.8 shows the *information-units* that fail to match.

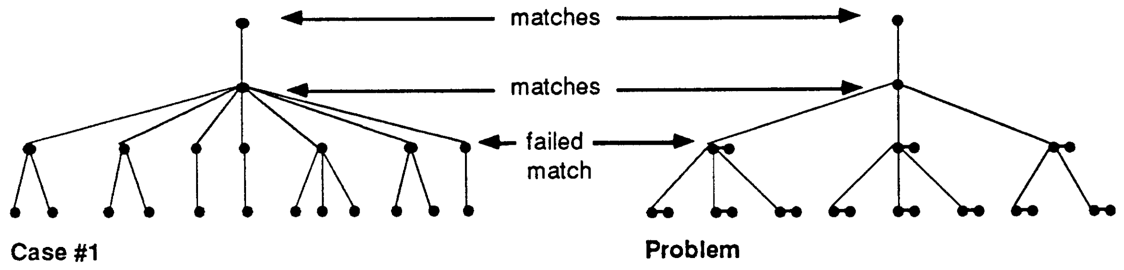


Figure 5.7 The search on case #1 failed at the third level when the *information-units* did not match.

```
CASE #1: Level 3 INFO-UNIT                     PROBLEM: Level 3 INFO-UNIT

CONTENT        "Obstructions"                  CONTENT        "6000x2000  Display"
INFO-TYPE      TEXT                            INFO-TYPE      TEXT
SUPERTOPIC     <INFO-UNIT>                     SUPERTOPIC     <INFO-UNIT>
SUBTOPIC       (<INFO-UNIT>  <INFO-UNIT>)      SUBTOPIC       (<INFO-UNIT> <INFO-UNIT>
                                                              <INFO-UNIT>)
RELEVANT-TO    NIL ◀—— failed match ——▶        RELEVANT-TO    <INFO-UNIT>
LAYOUT-FUNCTION                                LAYOUT-FUNCTION
    #<Standard-Generic-Function                    NIL
     LAYOUT-BULLETED-LIST (1)>                 LAYOUT-OBJECTNIL
LAYOUT-OBJECT NIL
```

Figure 5.8 For case #1, the *information-units* fail to match on the *relevant-to* slot.

Reject case #2

Because case #1 failed, the system goes onto the next case in the library list. Case #2 also fails to match; it uses a different information type than the problem content at the third level of the hierarchy. This search fails at the third level of the hierarchy because the case *information-type* slot is paragraph while the problem's *information-type* is text. Figure 5.9 shows the search path for this case. This case would have also failed to match because the logical structure is different that the problem. The case *information-unit* does not have a value for the *subtopic* slot while the problem *information-unit* does. Figure 5.10 shows the *information-unit* that fails to match for case #2.

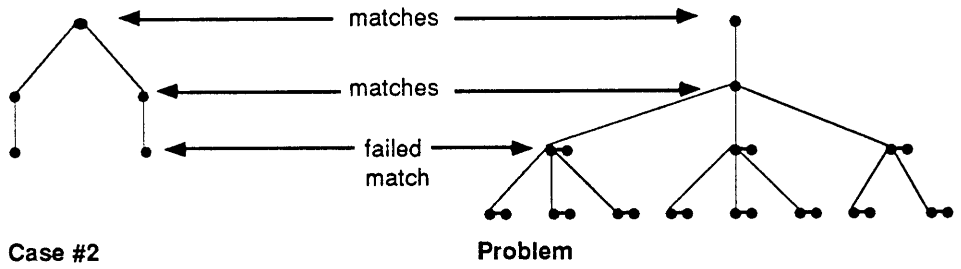


Figure 5.9 The search on case #2 failed at the third level when the *information-units* did not match.

```
CASE #2: Level 2 INFO-UNIT                    PROBLEM: Level 3 INFO-UNIT

CONTENT         "Current research ..."        CONTENT       "6000x2000  Display"
INFO-TYPE       PARAGRAPH<— failed match —>   INFO-TYPE     TEXT
SUPERTOPIC      <INFO-UNIT>                   SUPERTOPIC    <INFO-UNIT>
SUBTOPIC        NIL                           SUBTOPIC      (<INFO-UNIT> <INFO-UNIT>
                                                            <INFO-UNIT>)
RELEVANT-TO     NIL                           RELEVANT-TO   <INFO-UNIT>
LAYOUT-FUNCTION                               LAYOUT-FUNCTION
     #<Standard-Generic-Function                   NIL
      LAYOUT-PARAGRAPH-LIST (1)>              LAYOUT-OBJECTNIL
LAYOUT-OBJECT NIL
```

Figure 5.10 For case #2, the *information-units* fail to match on the *information-type* slot.

Select case #3

LIGA selects case #3 as appropriate to solve the layout problem. The *information-units* at each level match on logical structure and information type. Both the case and the problem are four-level hierarchies of text with relevant images at the third and fourth levels. Figure 5.11 shows the search path for this case. Note that the logical structures of the problem and case do not have to be identical. The problem has eight node pairs at the fourth level while the case has nine. Figure 5.12 shows the *information-units* that are compared at the last level of the hierarchy. When the bottom of the tree is reached without a failed match, case #3 is selected to solve the layout problem.

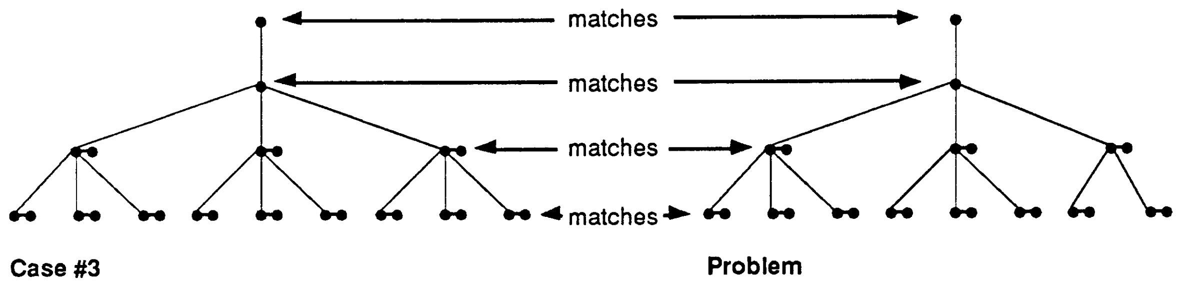


Figure 5.11 Case #3 matched the problem at all four levels of the hierarchy.

```
CASE #3: Level 2 INFO-UNIT                     PROBLEM: Level 3 INFO-UNIT

CONTENT        "Rain and Humidity"             CONTENT        "World Weather Data"
INFO-TYPE      TEXT ◄─────────────────────►    INFO-TYPE      TEXT
SUPERTOPIC     <INFO-UNIT>◄───────────────►    SUPERTOPIC     <INFO-UNIT>
SUBTOPIC       NIL ◄──────────────────────►    SUBTOPIC       NIL
RELEVANT-TO    <INFO-UNIT> ◄──────────────►    RELEVANT-TO    <INFO-UNIT>
LAYOUT-FUNCTION                                LAYOUT-FUNCTION
     #<Standard-Generic-Function                    NIL
      LAYOUT-PARAGRAPH-LIST (1)>               LAYOUT-OBJECTNIL
LAYOUT-OBJECT NIL
```

Figure 5.12 For case #3, the *information-units* at the bottom of the tree match on all necessary slots.

3. Case knowledge representation

LIGA selected case #3 to solve the layout problem. The following section describes the knowledge representation for this case. In LIGA, the case specific design knowledge is represented as a set of functions that create a grid and as well the visual elements that will appear in the new layout. The functions also place constraints on the slots of those elements. The constraints describe the relative size, position and color of the layout elements. The cases use the general knowledge base and case specific constraint definitions to represent the design knowledge for a particular layout case.

Layout functions

The most detailed design knowledge in the system resides in the layout case functions. A single layout case may have 10 layout functions. The layout functions are organized hierarchically and each layout function generates a component of the layout. The design knowledge for case #3 is contained in six such functions: *make-constrained-grid*, *make-title-with-subtopic, make-heading-with-rule-below, make-text-over-bitmap-list, make-supertopic-image-text,* and *make-subtopic-image-text.* The *make-constrained-grid* function generates a grid and has constraints that describe the relationship between the margins, column gutters, and the standard text size of the layout grid for that particular layout case. The general knowledge base contains a class definition for a layout grid.

The remaining functions generate the actual graphic elements that will appear in the layout. Figure 5.13 shows the components of the layout that these functions generate. In general, these layout functions generate layout elements for one level of the hierarchy and impose constraints on the elements of its immediate subtopic. For example, *make-title-with-subtopic* generates the piece of text for the title and has constraints between the title text and the subtitle's text and the graphic rule below it.

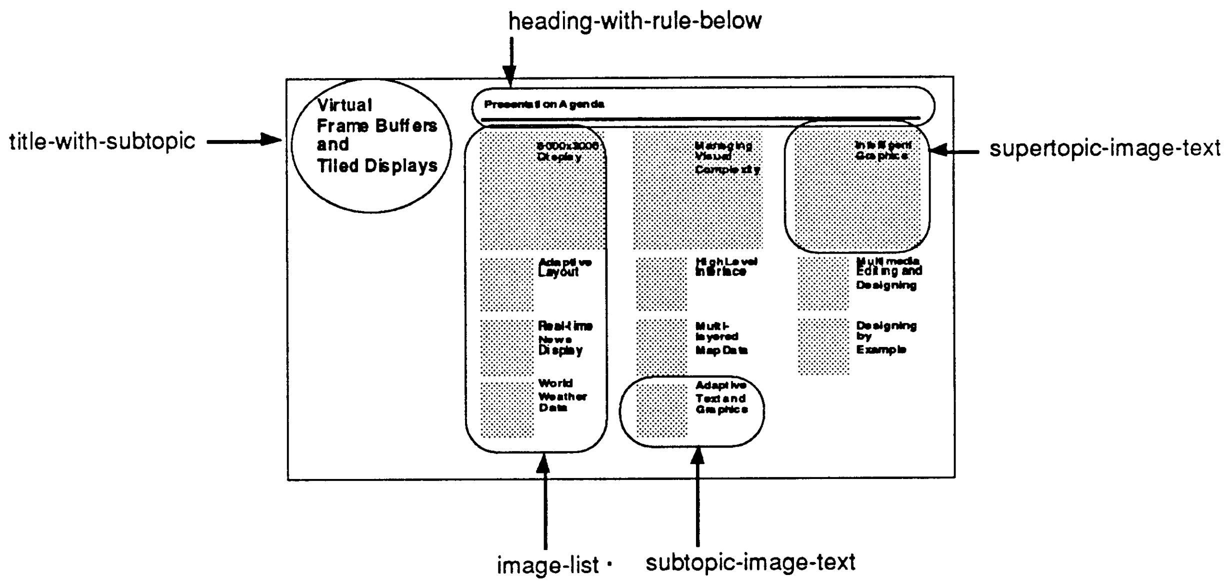


Figure 5.13 The case layout is broken down into five layout functions. Each function is responsible for generating a unique component of the layout.

The layout functions reside in the *information-units* that represents the case layout. The layout function *make-title-with-subtopic* resides in the *information-unit* that represents the content for the title of the layout. This *information-unit* is shown below. The value of its *layout-function* slot is *make-title-with-subtopic.*

```
< INFO-UNIT "Virtual Fra" 274349038>
    is an instance of the class INFO-UNIT:
 The following slots have allocation :INSTANCE:
 CONTENT         "Virtual Frame Buffers and Tiled Displays"
 INFO-TYPE       TEXT
 SUPERTOPIC      NIL
 SUBTOPIC        (< INFO-UNIT "Presentation Agenda" 274349142>)
 RELEVANT-TO     NIL
 LAYOUT-FUNCTION #<Standard-Generic-Function
                   MAKE-TITLE-WITH-SUBTOPIC (1)>
 LAYOUT-OBJECT   NIL
```

The layout functions may appear more than once in the hierarchy of case *information-units.* For example, the *make-text-over-bitmap-list,* function appears in each of the *information-units* at the third level of the hierarchy.

Each layout function has an associated class definition as well as constraint definitions. The class and function definitions for the *title-with-subtopic* element are shown below. The value of the subtopic slot for this class would be an object generated by *make-heading-with-rule-below.* The *title-with-subtopic* class has constraints with the grid, so it has a *layout-area* slot for the *layout-area* object.

```
(defclass title-with-subtopic
    ()
  ((layout-area :initform nil ;a layout-area object
          :initarg :layout-area
          :accessor layout-area)
   (text    :initform nil ;the title
           :initarg :text
           :accessor text)
   (subtopic :initform nil ;a heading-with-rule element
           :initarg :subtopic
           :accessor subtopic))
  (:documentation "This is an example of a Title with one
subtopic."))

(defmethod make-title-with-subtopic ((content info-unit))
  (let ((unit (make-instance 'title-with-subtopic
                     :text (make-text-element
                            :text-string (content content)
                            :typeface *swiss*
                            )
                     :subtopic (subtopic content)
                     :layout-area *case3-layout-area*)))
   (make-constraints unit)
       unit))
```

Note that *make-title-with-subtopic* calls the function *make-text-element.* This function is defined in LIGA's general knowledge base. The general knowledge base contains class definitions of basic layout elements and functions for generating instances of those classes. The *make-title-with-subtopic* function creates an instance of a *text-element* and places constraints on the slots of that element by calling its own method *make-constraints.* LIGA's constraint system and general knowledge base of layout elements are described below.

Constraint System

LIGA uses constraints to represent the visual relationships between layout elements. The constraint system underlies both the general knowledge base and the case library of the system. The constraints are primarily responsible for generating the specifications for a new layout.

The constraint system used in this research is based on the constraint system described by Abelson and Sussman [ABEL 85]. The system provides the capabilities of typical constraint systems: the expression of relationships between variables, local propagation, and nondirectionality of computation. In LIGA, the constraint system is implemented as an object oriented system.

The constraint system has the capability of defining arbitrary constraints between variables that are functionally related. For example, an "adder" can be defined that represents the relationship a + b = c. These definitions are referred to as constraint primitives. The plan of the constraint system is to combine the primitives to express more complex relationships. These combinations are referred to as "constraint networks:" the primitives compute values locally based on available data, and the direction of computation is determined dynamically [STEL 80].

The constraint system did not originally include relaxation capabilities, but the system was general enough that it was possible to extend it. A new method, *relax-value*, was added to the constraint system as part of this research. The relaxation method was developed in order to implement the rules of text legibility. This method allows special constraints to be written that test a value, rather than compute a value. If the value does not pass the test a *relax-value* method is called that adjusts the value until the constraint is satisfied.

Representation of grids and layout elements

The general knowledge base contains class definitions for layout grids and the visual elements that will typically appear in a layout. These elements are: *text-element*, *image-element*, *rule-bar-element* (traditionally called rules and/or bars in graphic design), and white-space. These class definitions are used as building blocks for the layout cases. The case layout functions create instances of these classes. The purpose of defining these object classes is to provide consistency of elements between cases, and to avoid redundant class definitions. It is not necessary to redefine element attributes for each case.

The general knowledge base has a class definition *basic-element* which describes the properties of all two-dimensional elements. The grid and layout element classes inherit form this class. The slots for this class are: *x1, x2, width, y1, y2, height,* and *area.* Figure 5.14 shows the dimensions to which these slots refer. Three methods are also defined for the *basic-element* class. These methods describe the geometric relationships that are common to any two-dimensional object. The relationships are implemented as constraints. The methods and the relationships they describe are listed below.

method-name	relationship
• basic-height-constraint	y1 + height = y2
• basic-width-constraint	x1 + width = x2
• basic-area-constraint	width * height = area

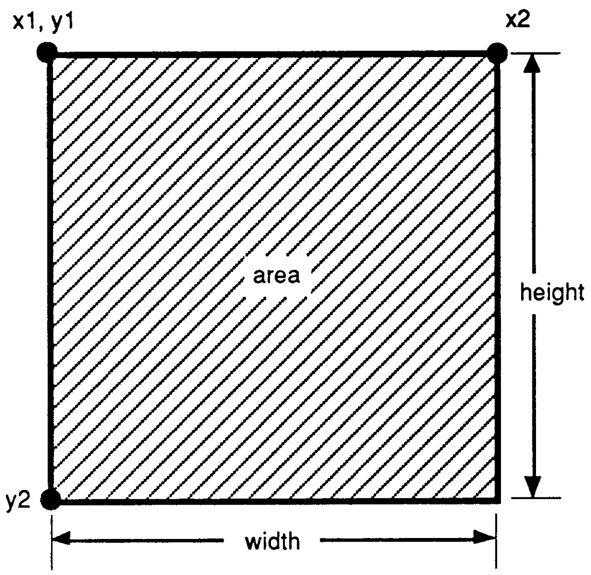


Figure 5.14 The *basic-element* class has slots for *x1, y1, x2, y2, width, height,* and *area.*

Layout grid representation

Designers use grids to help organize the information in a layout. Because a layout grid is unique to layout design, it is necessary and useful to represent this information in the system. Although layout grids vary in proportion and in number of horizontal and vertical divisions, the properties of a grid and the method of constructing grids remain constant for all layout grids. In LIGA, the grid object also has knowledge of the layout viewing context and of the standard text size for the layout. This information is used in the system to ensure that text placed on the grid is legible. To represent knowledge of the layout grid, the general knowledge base contains a set of class definitions, constraints on grid components, and functions for creating grid-component objects.

A class definition *layout-area* holds all information about the grid and has slots for margins, gutters, and row and column measures. Figure 5.15 shows how these slots correspond to the layout grid. Other class definitions associated with the grid include *active-area*, *column*, and *row*. The *active-area* represents the area within the margins. The *layout-area*, *active-area*, *column*, and *row* classes inherit slots and basic geometric constraints from the *basic-element* class. The *layout-area* class also has slots that describe the viewing context of the layout.

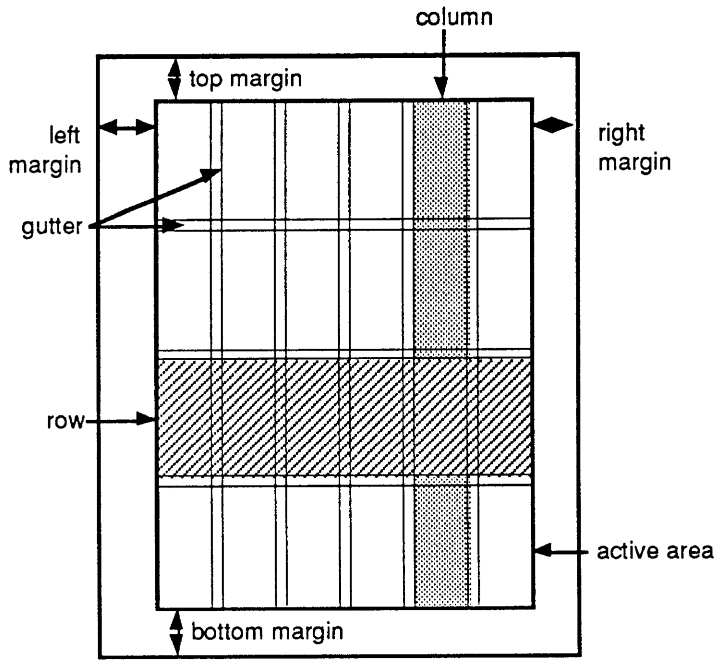


Figure 5.15 These properties of a layout grid are represented as slots in the *layout-area* class.

Grid constraints

The constraints specified for the grid in the general knowledge base are common to all layout grids. The constraints relate top and bottom margins to the *active-area* height, left and right margins to the *active-area* width, and number of columns and gutter width to column width. When the slots for number-of-columns, gutter-width, and column-width have values, a function is called that creates the appropriate number of *column* objects and propagates the x and y positions of the columns via constraints. The *column* object stores information about its own x and y coordinates, width, height, and column number. These *column* objects are placed in the *column-list* slot of the *layout-area*. There are analogous constraints and functions for generating the *row-list*.

Viewing-context and Basic-text-size

The slots of the *layout-area* class that are used to describe the viewing context of the layout are *basic-text-size* and *viewing-context*. The *basic-text-size* slot of the *layout-area* class represents the standard or common text size for that particular case layout. It represents what is often called the "body copy" of a layout. In a magazine page layout, the basic-text size would be the text size of the paragraphs of the articles. This information about *basic-text-size* is used by the rules of text legibility to ensure that the size of text will be large enough to read. The *basic-text-size* slot of the *layout-area* is constrained to be equal to the text size of a specific *text-element* in the layout case. This *text-element* is referred to as the *basic-text-element* and will be different for every case. In the layout shown in figure 5.16, the *basic-text-element* is the category title that lies on top of the larger images.

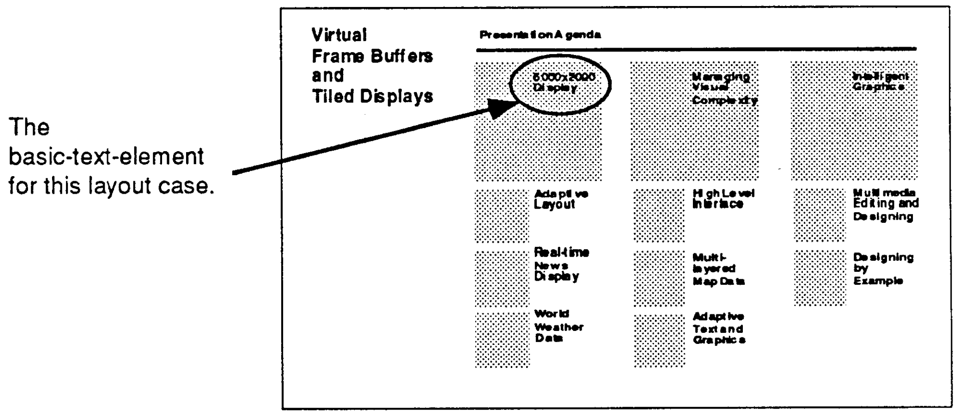


Figure 5.16 The *basic-text-element* for this case is the category element that lies on top of the larger images.

Knowledge of layout elements

The general knowledge base includes representations of the graphic elements that will appear in a layout. Four types of basic layout elements are defined. They are *text-element, image-element, rule-bar-element,* and *whit- space.*

Both class definitions of the basic layout elements and constraints that express inherent geometric relationships, reside in the general knowledge base. A database of typeface letterform ratios, and constraints for expressing the letterform relationships, are also included.

The four categories of layout elements inherit both the attributes and methods from the *basic-element* class. Figure 5.17 shows the taxonomy of layout elements as it is defined in LIGA.

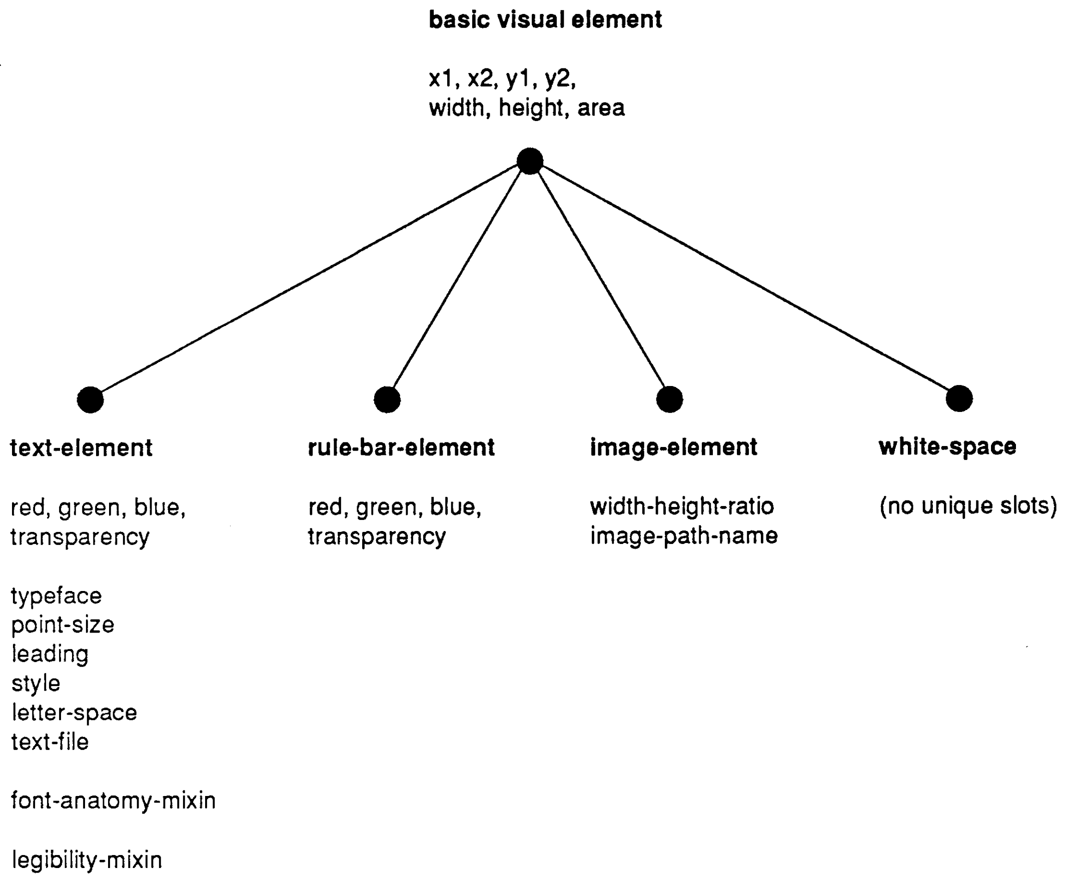


Figure 5.17 The taxonomy of basic layout elements. The layout elements inherit slots from the class basic-visual-element and have their own unique slots. The slots for each class are listed.

The four classes of basic layout elements inherit the attributes of the *basic-element* class and have their own attributes that further define each class. For example, the *text-element* and *rule-bar-element* have a color attribute while the *image-element* does not. It is assumed that an image might have many colors, but a color cannot be assigned to an image as it can to a piece of text or a graphic bar. The image element has a unique attribute: *width-height-ratio.* This ratio expresses the proportion of width to height of a given image.

The slots for each of the basic layout element types are listed in the taxonomy shown above in Figure 5.17. The *text-element* class has the greatest number of unique attributes. These additional slots are required to describe the specifications used in displaying text, such as typeface and line spacing. The *text-element* class also has two mixin classes, *font-anatomy-mixin* and *legibility-mixin.* The *legibility-mixin* has slots that are used to store information about the minimum and maximum size and characters per line of a *text-element.*

Font-anatomy-mixin

The *font-anatomy-mixin* provide slots for the detailed knowledge of typographic letterform. The purpose of having this information available in the system is to allow the expression of constraint relationships based on these attributes. Designers often make size and placement decisions of layout elements based on the measurements of the typographic letterforms being used in the layout. The slots of the *font-anatomy-mixin* are listed below. The terms used for the slot names were defined in Chapter 3, "Approach." Figure 5.18 shows the parts of the letterform that correspond to the slot names.

```
font-anatomy-mixin:
body-height
cap-height
x-height
meanline
baseline
stroke-width
```

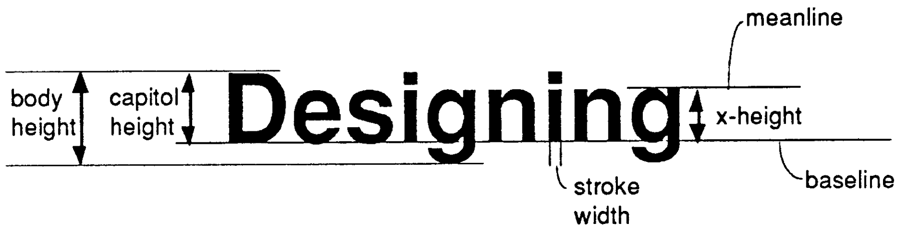


Figure 5.18. The parts of the letterform that are represented as slots in the font-anatomy-mixin.

A set of constraints, along with a database of letterform ratios, are used to calculate the values of the *cap-height, x-height, stroke-width, body-height,* and *meanline* slots for each instance of a *text-element.* The value of the letterform slots is dependent on the *text-size* and *typeface* slot values of the *text-element.* The letterform values are dependent on typeface because the proportions of the letterform attributes vary from typeface to typeface. The use of constraints to express these relationships allows the slot values to be calculated dynamically and changed dynamically when the type size or typeface changes.

The general knowledge base includes a database of letterform ratios for each typeface available in the system. A class is defined which has slots for the ratio of *text-size* to *cap-height, x-height,* etc., for each typeface. The object that contains the ratios for the typeface Swiss is listed below.

```
*SWISS*
#<Font-Anatomy-Ratios #X104B8706>
   is an instance of the class FONT-ANATOMY-RATIOS:
 The following slots have allocation :INSTANCE:
 TYPEFACE-NAME             "Swiss"
 CAP-HEIGHT-RATIO 73/100
 X-HEIGHT-RATIO 96/100
 BODY-HEIGHT-RATIO  11/100
 STROKE-WIDTH-RATIO 11/100
 CHARS-PER-PIXEL-RATIO 15/100
```

4. Generating a new layout

When a solution case is selected form the case library, the knowledge from the layout case is applied to the layout problem. Knowledge is applied by copying the values of the *layout-function* slots of the layout case's *information-units* into the *layout-function* slots of the problem layout *information-units.* A top down search is conducted, this time covering all nodes of the problem layout. As each node is passed, the layout function from the same level of the layout case *information-unit* is copied into the *layout-function* slot of the problem layout. Finally, to create the layout objects for the newly solved layout problem, the layout functions are called. The grid is instantiated and then the remaining layout functions are called from the bottom up.

To generate the layout for the Presentation Agenda, the grid object is made by calling *make-constrained-grid,* and the input values for display width and height, and viewing context, are set. Initially there are no values for *margins, gutter-width,* or *column-width* slots of the *layout-area.* All of these values are dependent on the *basic-text-size* and will not be set until the *basic-text-element* is identified.

After the grid object has been made, the layout functions are called from the bottom up as supertopic functions may have subtopic layout objects as arguments. The bottom-up generation of layout elements has also been used by Feiner [FEIN 88] and Kamada [KAMA 91]. The argument for the layout function is its own *information-unit.* The functions for this example are called in the following order: *make-subtopic-image-with-text, make-text-over-bitmap-list, make-heading-with-rule,* and *make-title-with-subtopic.*

Direction of propagation

The cases contain enough knowledge to generate values for the size, position, and color of all layout elements when given the width and height of the display area and the viewing context of the layout. For case #3, propagation of these values will not begin until the *basic-text-element* has been identified. This does not occur until the *make-text-over-bitmap-list* function is called. The first few function calls do not result in any constraint propagation. The objects are simply created and constraints are placed on their slots. None of the values for the image and text elements can be propagated until *basic-text-size* and *column-width* are known. Both of these values are dependent on the *text-size* of the *basic-text-element.* The remaining values will not be propagated until after the *basic-text-element* has been identified.

The *basic-text-element* is identified by calling the function *identify-basic-text-element.* This function call is part of the case specific knowledge. This function relates the *basic-text-element* to the *basic-text-size* slot of the *layout-area* and places the constraint *check-chars-per-line* on that *text-element. Check-chars-per-line* tests the current value of the *text-element's* text size and number of characters per line. If the values are not within the established limits, a relax method is invoked. One of the arguments to the relax method is a flag whose value is "over" or "under." The flag is used to determine which direction to adjust the value in order to adhere to the constraints.

The size of the *basic-text-element* is initially set to the default size for the viewing context of the layout. In this example, the viewing context is "demo" and the default text size is 50. The maximum and minimum size slots of the *basic-text-element* are set to the maximum and minimum sizes of the *viewing-context* object. Figure 5.19 shows the function definition for *identify-basic-text-element.*

```
(defmethod IDENTIFY-BASIC-TEXT-ELEMENT ((text-unit title-element)
    (layout-unit layout-area))
  (set-basic-text-size  text-unit layout-unit)
  (set-value! (point-size text-unit) ;sets text size to default
           (value (default-text-size
           (viewing-context layout-unit))) 'user)
  (CHECK-CHARS-PER-LINE text-unit layout-unit) ;legibility const.
```

Figure 5.19 The function definition *identify-basic-text-element.*

Now that the *basic-text-size* is known, all other values for the layout can be propagated. Figure 5.20 shows the values of the *layout-area* after the *basic-text-element* has been identified. The *margins, column-width,* and *gutter-width* slots have values. In addition, the *column* objects are generated and placed in the *column-list* slot. After the values for the *layout-area* are propagated, all other values for the layout can be propagated.

```
Describing #<Layout-Area #X1076D676>
RED   0
GREEN   0
BLUE   0
TRANSPARENCY   0
X1   0
Y1   0
X2   1000
Y2   600
WIDTH   3500
HEIGHT   2040
AREA   NIL
RELATIVE-X1   0
RELATIVE-Y1   0
TOP-MARGIN   100
BOTTOM-MARGIN   50
LEFT-MARGIN   100
RIGHT-MARGIN   100
ACTIVE-AREA   #<Active-Area #X1076D7DE>
NUMBER-OF-COLUMNS   4
NUMBER-OF-ROWS   NIL
GUTTER-WIDTH   100
COLUMN-WIDTH   600
ROW-HEIGHT   NIL
GUTTER-HEIGHT   NIL
COLUMN-LIST   (#<Column-Element #X10424AA6> #<Column-Element
#X10413E4E> #<Column-Element #X104031F6> #<Column-Element
#X103ECB5E>)
ROW-LIST   NIL
BASIC-TEXT-SIZE   50
VIEWING-CONTEXT   #<Legibility-Context-Sizes #X1076778E>
```

Figure 5.20 The values of the *layout-area* after the *basic-text-size* has been identified.

Rules of legibility check line length

The *check-chars-per-line* constraint does not become active until the *basic-text-element* has values for its *text-size* and*width* slots. In this example, the *basic-text-element* has an acceptable number of characters per line using the default text size for the "demo" viewing context. The rules of legibility will adjust the layout though, when it needs to be adapted to fit into a smaller area.

5. Adapting a layout

The LIGA system can adapt a layout in the event that there is a change in the available display area. In this example, during the presentation to research sponsors, an image from the project list is enlarged and displayed on the screen. The available area for the layout is reduced from 3500 x 2048 pixels to 1500 x 2048 pixels. Figure 5.21 shows the adapted layout that LIGA generates for this situation.

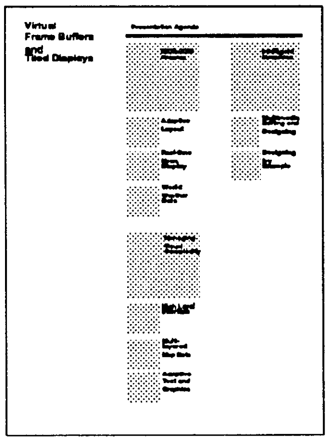


Figure 5.21 The adapted layout that LIGA generates for the reduced display area.

Change width and height

The only input values LIGA requires to adapt a layout are the width and height of the new display area. The layout case functions, along with the rules of legibility, contain enough knowledge to generate an adapted layout from this data alone. To adapt the layout, the original values for width and height are retracted and set with the new values. Retracting the width and height causes all dependent values to also be retracted. Essentially all widths, heights, x-positions, and y-positions of the layout elements are somehow dependent on the width and height of the layout area. All of these values are retracted. When the new values for the display width and height are set, the values for all widths, heights, x-positions, and y-positions are repropagated. The two values that are not immediately affected by the change in display dimensions are the number of columns in the grid and the size of the *basic-text-element.*

Check legibility of text

When all layout elements have new values based on the new display dimensions, the grid still has four columns. This results in a column width that is much smaller than in the larger layout. Figure 5.22 shows the resulting four column grid. The basic text size has not changed from the larger layout; it is still 50. At this point in the process the rules of legibility notice that there are too few characters per line. The *check-chars-per-line* constraint calls the *relax-value* method with the flag "under." This method decreases the text size by one and checks the characters per line again. The constraint continues to decrement the text size until the rule is satisfied or the minimum size for the viewing context has been reached.

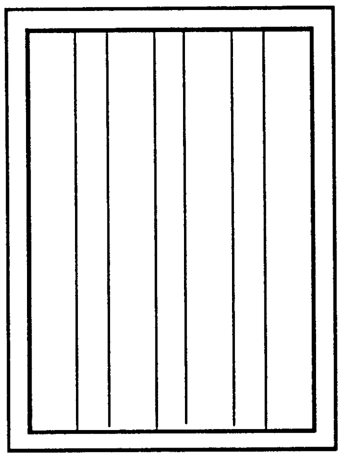


Figure 5.22 When the layout dimensions are reduced, the grid initially still has four columns.

In this example, the text size is decreased from 50 to 30, the minimum allowable size. Each time the text size is decreased, most values for the layout elements are retracted and recalculated. The grid margins and gutters for this layout are dependent upon the *basic-text-size* of the layout. Consequently, all text sizes, positions, widths, and heights, of all elements in the layout, are changed when the *basic-text-size* is changed.

When the minimum size for the viewing context is reached and the number of characters per line is still too few, the line length of the text must be increased to create a legible number of characters per line. This increase in line length is achieved by decreasing the number of columns in the grid which will in turn increase the width of the columns.

Because the text size is at the minimum for this example, the number of columns is reduced by one. This creates fewer columns of greater width, increasing the line length for text elements. The value for *number-of-columns* slot is retracted. This retracts the value for *column-width* and all values dependent on that. The values for the grid are repropagated using three columns instead of four. All positions and sizes of layout elements are subsequently repropagated. This change in the column width results in an acceptable line length for the text and the system displays the final layout.

Although the adapted layout is different from the original, the system has adhered to the goals of maintaining legibility, visual structure, and design style. The text size is reduced but still legible for the viewing context. The size and position of images are changed, but the visual structure still shows that there are three major categories of information. The graphic style of the layout is maintained – the use of typeface, graphic rule, and white space is consistent with the original layout.

After all of the values for the layout elements are generated and legibility constraints are satisfied, the layout is displayed.

6. Display

Layout descriptions generated by LIGA are displayed graphically using BadWindows [ALAV 91], a C-based system developed at the Visible Language Workshop, MIT Media Laboratory. LIGA represents graphic elements as LISP/CLOS objects. The specifications for these objects are transferred from LIGA to the graphic display system through a LISP-C interpreter.

This extra step of transferring data from LISP to C is done to take advantage of the high quality graphics available in BadWindows and the ability to display graphics on several platforms including a 2000 x 6000 display [MASU 92]. The representation of layout specifications in LIGA is independent of the display system.

The display component includes a library of LISP functions that correspond to the BadWindows functions. The functions in this library use one foreign function call to send the name of the BadWindows function and the appropriate arguments to the BadWindows Interpreter (BWI). This data is sent as a string data type. BWI parses the strings sent from LISP, casts data to the appropriate types, and calls the corresponding BadWindows function. The layout object then appears graphically on a screen. Any necessary return values from the BadWindows functions are received in LISP as integers. The interpreter takes advantage of "appcom," the applications communication library, which is a socket-based message passing library written as part of "Build-a-Dude." [JOHN 91]. This message passing system eliminates the need to send and receive complex foreign data types such as C-pointers to Window structures.

Chapter 6 **Evaluation and Conclusion**

This research identifies important knowledge about layout design, proposes a way of representing that knowledge, and shows how it can be implemented in a system that generates layouts. The approach of using case-based reasoning, constraints, a general knowledge base, and logical content structure works well. The system finds an appropriate case and applies case knowledge to produce a good layout. Some of the components of LIGA could be improved and extended to solve more complex layout problems and to represent more sophisticated design knowledge. The following section evaluates each aspect of the approach, proposes extensions to the system, and discusses areas of further research that are needed to more fully address the problem of representing layout design knowledge.

Case representation

In LIGA, case knowledge is generalized from the original example. The knowledge contained in the layout cases is sufficient to generate a layout for varying display areas, viewing contexts and amounts of content. An input layout problem does not have to be exactly like the original case in order to apply the knowledge and generate a new layout. Unfortunately, the case does not contain all of the knowledge that was used to design the original example layout.

The purpose of using case-based reasoning is to be able to represent knowledge that is too complex to be represented as discrete rules. This strategy works well to represent the combined use of color, size and position of a typographic hierarchy layout. Unfortunately, important design knowledge, such as reuse of layout components and visual balance, is not represented in the cases.

Reuse of layout components

Designers will often develop a set of layout formats that can be used in combination. In LIGA's representation of layout cases, the layout functions are organized modularly so that they can be used across cases. The system knows how to use formats in combination only because the specific example is given. Ideally, the system should have knowledge of which layout components can be combined. Further research is required to determine how to represent this knowledge of format combinations.

Visual balance

Knowledge of visual balance is difficult to capture and describe. Figure 7.1 shows the layout that LIGA adapts to accommodate a change in available display area. Although the adapted layout satisfies the goals of logical structure, legibility, and style consistency, it would not satisfy an expert designer's goal of visual balance. The layout is considered imbalanced because the third column is partially empty. The designer's original strategy for the case layout was that the three columns under the red rule-bar should be full, or nearly full, while the first column below the title should remain empty. This empty space in the first column draws attention to the title. Any other empty column voids the functionality of the empty space in the first column.

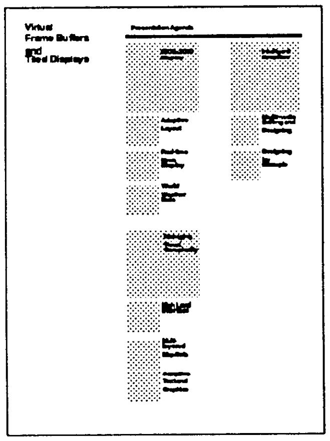


Figure 7.1 Although this layout satisfies the goals of logical structure, legibility, and style consistency, it would not satisfy an expert designer's goal of visual balance because the third column is partially empty.

LIGA does not contain knowledge of visual balance. The system can be extended to include some knowledge of visual balance, by having the case check its columns and decide which columns should be full. If the columns are not full, the solution will be rejected and LIGA will continue searching through the library for a case that will solve the layout problem completely. If another layout cannot be found, the system could combine layout components to produce a good solution.

The proposed method of representing visual balance described above does not address all issues relevant to representing visual balance. Visual balance depends not only upon occupied space but also upon the scale, shape, and tone of layout elements. This design knowledge is dependent on complex human visual processes and is difficult to describe and encode. Further research is required to address this representation problem.

Constraints

Constraints prove to be an effective way to represent many of the visual relationships that exist between layout elements. However, the use of constraints could be improved by using both discrete and continuous constraints. Discrete constraints restrict a property to be a specific value. Continuous constraints allow a value to range between a maximum and minimum [WEIT 88]. The original constraint system used in LIGA provides only discrete constraints. The incorporation of continuous constraints into the layout case representation would allow more sophisticated design knowledge to be represented. For example, a designer may specify a range of text sizes that could be used for a title. The specific size used might depend upon the number of words in the title. Continuous constraints could be used to represent this kind of design knowledge. The implementation of the rules of legibility in LIGA uses the concept of continuous constraints by having a maximum and minimum legible text size, and then a relax method that knows when to adjust text size within that range. This concept could be extended and used to represent the design knowledge for the layout cases.

Match on logical structure and information type

The matching strategy for identifying appropriate case layouts based on logical content structure and information types works well. The logical structure and information types are identifiable properties of the content. The matching strategy could be made more sophisticated by also checking the number of subtopics or relevant items that an information-unit has.

This system assumes that the branches in the content hierarchy are even because of the simplicity of the matching function. Many content structures would not have even hierarchies. The searching and matching methods could certainly be further developed to allow for content structures that did not have an even hierarchy.

Additional content properties

There are content properties in addition to logical structure and information type that designers consider when making design decisions. LIGA's representation of the input layout problem could be extended. For example, the communication function of the content elements could be described. Some pieces of content serve as descriptive information and others as directive. Designers often use visual attributes to differentiate between descriptive and directive information [ICHI 88].

There are other factors in addition to content characteristics that affect layout design decisions. Designers will use different visual arrangements depending upon the intended connotation of the layout. Two layouts can have similar content structure and information types but the arrangement of the visual elements will convey different moods. For example, one layout may be conservative while another is avant garde. The connotation that a layout design is intended to convey could be represented in the layout cases. This information could be used to further identify appropriate solution cases for input problems.

General knowledge base

LIGA's general knowledge base contains a limited set of general knowledge about information layout and is used in conjunction with the layout case information to solve layout problems. This approach reduces the amount of code required to represent the case layouts, and avoids redundant representations in the system. The representation of general knowledge could be refined and extended.

Layout grids

The current implementation of LIGA assumes that grid units will be of equal measure. The grid constraints currently specified in the general knowledge base are common to all layout grids of this type. A specific layout may include additional constraints on the attributes of the grid. For example, a designer may specify that there is a relationship between sizes of the margins, or a relationship between gutter width and column width. These types of constraints are specified at the case level.

Although this system supports only grids with units of equal measure, there are well designed layouts that do not hold to this convention. Figure 7.2 shows a layout that appears to have a three column grid where the widths of the three columns are equal. In fact, the first column is slightly narrower than the second two. The first column is just wide enough to provide an adequate line length for the text. This narrower first column provides more space for the images which are the focus of the layout. Future work on this system would include developing ways to represent this type of grid design.

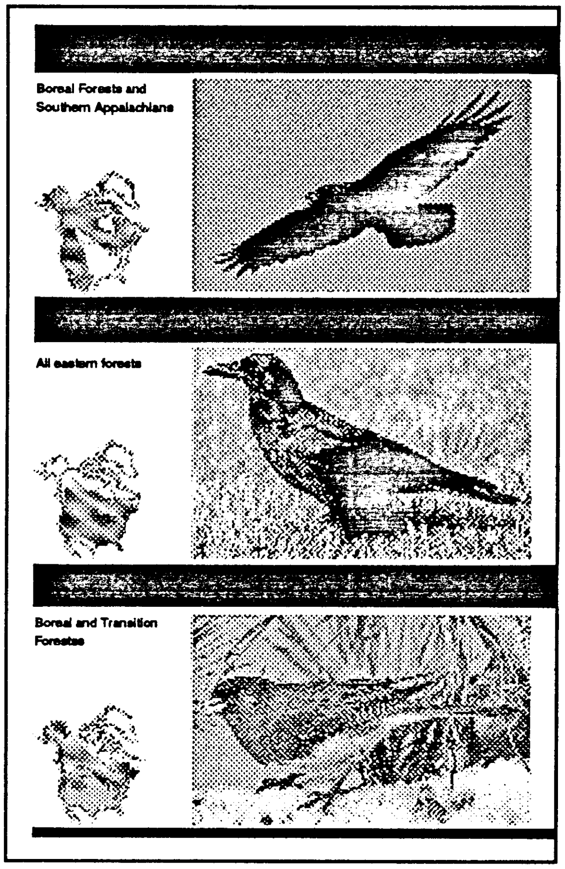

Boreal Forests and Southern Appalachians
All eastern forests
Boreal and Transition Forests

Figure 7.2 This layout appears to use a grid with three equal columns but in fact the first column is slightly narrower that the other two. Reproduced from [SUTT 86].

Taxonomy of layout elements

The taxonomy of layout elements used in this system is quite simple. The taxonomy could be extended by specifying subcategories of the text, rule-bar, and image categories. Such an extension would allow the encoding of more intelligence in the system. For example, subsets of text might be title, heading, body, and caption. These four types of text might have slightly different rules of legibility. Caption text might have a smaller minimum allowable size than body text, for example. The rule-bar category could be changed to "abstract marks" with subsets being rule, bar, and bullet. The image category could be broken down into continuous-tone image, symbol (or icon), illustration, and diagram. Again there might be usage rules, and rules of legibility, that differ for bullet and rule, or diagram and icon. Figure 7.3 shows how the taxonomy of basic layout elements might be extended.

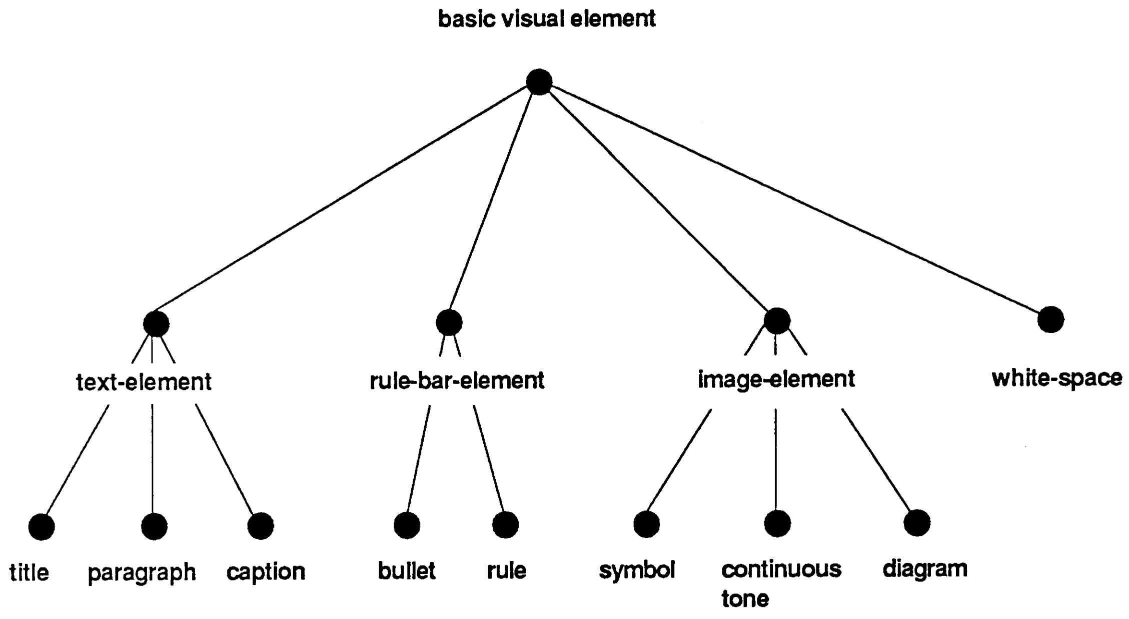


Figure 7.3 The taxonomy can be extended to further define properties of layout elements.

Rules of text legibility

LIGA's rules of text legibility that concern characters per line have the ability to change the number of columns in a grid. This change is made to increase or decrease the column width and achieve an acceptable number of characters per line. For some layout cases, this change in number of columns may violate the designer's original intention. and result in a poorly designed layout. For example, a designer may choose a five column grid to ensure asymmetry in the layout. Reducing the number of columns from five to four would destroy the asymmetry. In other layout cases, it may be perfectly appropriate to change the number of grid columns. additional knowledge is needed both in the cases and the rules of legibility to allow the system to make more informed decisions about adjusting the number of columns in a grid. This additional knowledge might include specification of a range of values for the gutter width. The rules of legibility could then try to achieve a legible line length by first adjusting the gutter width, then moving to the more extreme change of adjusting the number of columns.

The rules of legibility allow LIGA to adapt a layout to different display area sizes. However, LIGA cannot produce a layout if the display space is too small and the type size is at its minimum. Additional knowledge is needed in the system to deal with such situations. A human graphic designer would solve such a problem by either editing the content so that there is more room, or by violating some of the legibility rules. Knowledge of these design compromises [FEIN 88] is needed in order for the system to be able to handle a larger variety of display situations.

Extend knowledge base

Although there is a limited amount of general knowledge of layout design in the world, there is current research in the field. As more of this knowledge is developed, it can be incorporated into the general knowledge base of the system. For example, knowledge of color legibility could be included at the general level. Although there are rules of thumb about color usage in layout, such as "don't put yellow text on a white background," this information is too general to be encoded into an intelligent system. The computer would not have common sense knowledge of yellow or white. Current research that encodes knowledge of color properties [BARD 92] could eventually be included in a general knowledge base for an intelligent layout system.

System usability

Capturing and encoding design knowledge

The current implementation of LIGA proved to be usable by others to encode design knowledge. Five people selected existing layouts, analyzed them for the design relationships, and encoded those relationships using the constraint system and the general knowledge base provided in LIGA. Two of these encoded layouts were used in the case library described in Chapters 2 and 5. They are case #4 and case #5.

Although it is possible to encode case layouts using LIGA's components, the process of writing a line of LISP code for each of the 60 to 100 constraints that are required to describe a layout is somewhat tedious. In addition it is difficult to determine design constraints simply by observing an existing layout. This information is best gotten from the original designer of the layout or another experienced designer. These difficulties in encoding and determining design relationships point to two areas of further research: developing tools for designers to easily express visual relationships and developing systems that can detect design relationships.

Current research in sketch recognition and Programming by Example [LIEB 91] could be used to develop an interface that would allow a designer to graphically specify important design relationships. The second area of research is to develop a system that can, given a layout, detect design relationships and automatically generate constraints [KURL 91]. A system could have knowledge of typical relationships and could infer the relationships of a given layout. For example, a title and subtitle will always have relationships to each other that need to be expressed. In addition, a system might be able to find the grid and margins of a layout and infer constraint relationships between the layout elements and the grid.

Developing content and layout simultaneously

The approach presented in this thesis assumes that content is developed and well structured before information is laid out. Often content and layout are developed simultaneously because the visual relationships in the layout effect the meaning of content. The content may even change during the layout process because the visual layout reveals inconsistencies or new relationships within the content. Although LIGA, in its current implementation, could not be used in situations where content and layout are developed simultaneously, the approaches of case-based reasoning and constraints could be incorporated into a layout tool. Cases could be used as points of departure in the design. Constraints could be used as a tool for the designer; the designer could establish and change constraints during the design process.

Multimedia and dynamics

The need for computer systems that contain graphic design intelligence arises from current advances in electronic information technology. These advances include the ability to display multimedia and real-time information. Although the layout examples used in LIGA's case library display two dimensional static information, the approach presented in this thesis could be extended to and applied to layout problems that use multimedia and dynamics. A case could represent a layout of multimedia information that includes video and sound or a layout of three dimensional information that can be changed dynamically by the user. LIGA's general knowledge base could be extended to include representations of three dimensional objects as well as video clips, sound bites, and "buttons." Constraints could be applied to both the visual and temporal properties of these elements. The information types that LIGA uses for matching could include these new media types. Research in the application of case based reasoning to the design of multimedia information is currently being conducted by MacNeil [MACN 91].

Conclusion

The number of research issues involved in developing systems that can automatically generate well-designed displays of information is vast. This current research addresses a limited set of those issues, namely the representation of design knowledge for information layout. The contribution of this research lies in culling design knowledge that can be represented using existing artificial intelligence technology and in creating a prototype that shows to what extent this knowledge can be used to solve information layout problems.

Bibliography

[ABEL 85] Abelson, H. and Sussman, G. J. (1985). *Structure and Interpretation of Computer Programs*. Cambridge, The MIT Press.

[ALAV 91] Alavi A., et al., (1991). *BadWindows Reference Manual*. Internal document, Visible Language Workshop, Media Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[BARD 92] Bardon, Didier. (1992). "Adaptive Color in Dynamic Mapping: A Method for Predictable Color Modifications." S.M. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology. Cambridge Massachusetts.

[BEAC 83] Beach, R. and Stone, M. (1983). "Graphical Styles Towards High Quality Illustrations." *Computer Graphics*:, Proceedings of SIGGRAPH '83. Vol. 17, No.3.

[BACK 83] Backer, David. (1983). "Personalized Electronic Publications." *Proceedings of the National Computer Graphics Association,* Chicago.

[CART 85] Carter, R. et al., (1985). *Typographic Design: Form and Communication*. New York, Van Nostrand Reinhold.

[CASN 91] Casner, S. M., (1991). "A Task-Analytic Approach to the Automated Design of Graphic Presentations," *ACM Transaction on Graphics,* Vol. 10, No. 2. New York, Association for Computing Machinery.

[COOP 89] Cooper, M. (1989). *Design Quarterly* 142. Cambridge, The MIT Press.

[CRAG 71] Craig, J. (1971). *Designing With Type*. New York, Watson-Guptill Publications.

[FEIN 88] Feiner, S. (1988). "A Grid-Based Approach to Automating Display Layout," *Proceedings of Graphics Interface '88*. Palo Alto, Morgan Kaufmann.

[FINB 91] Fineblum, M. (1991). “Adaptive Presentation Styles for Dynamic Hypermedia Scripts.” S.M. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology. Cambridge, Massachusetts.

[GERS 73] Gerstner, K. (1973). *Think programs: Synopsis of the exhibit ‘Designing Programs/Programming Design.’* New York, Museum of Modern Art.

[HURL 77] Hurlburt, A. (1977) *Layout: the design of the printed page.* New York, Watson-Guptill Publishers.

[ICHI 88] Ichikawa, Tomoko. (1988). “An Application of Sign Systems in Instructional Manuals: Correlating Information Types and Sign Systems,” M.S. Thesis, Institute of Design, Illinois Institute of Technology, Chicago.

[JOHN 91] Johnson, M. B. (1991). “Build a Dude,” S.M. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology. Cambridge, Massachusetts.

[KAMA 91] Kamada, T. and Kawai, S. (1991). “A General Framework for Visualizing Abstract Objects and Relations,” *ACM Transaction on Graphics*, Vol. 10, No. 1. New York, Association for Computing Machinery.

[KOSA 91] Kosak, C., Marks, J. and Sheiber, S. (1991). “A parallel genetic algorithm for network-diagram layout.” *Proceedings of the Fourth International Conference on Genetic Algorithms,* UCSD, California.

[KURL 91] Kurlander, D. and Feiner, S. (1991). “Inferring Constraints from Multiple Snapshots.” Technical Report 008-91. Computer Science Department, Columbia University, New York.

[LIEB 88] Lieberman, H. (1988). “Communication of Expert Knowledge in Graphic Design.” Internal document, Visible Language Workshop, Media Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[LIEB 91] Lieberman, H. (1991). "Mondrian: A Teachable Graphical Editor." Internal document, Visible Language Workshop, Media Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[LUGE 89] Luger, G. F. and Stubblefield, W. A. (1989). *Artificial Intelligence and the Design of Expert Systems* . Reading, Massachusetts, The Benjamin/Cummings Publishing Company, Inc.

[MASU 92] Masuishi, T. et al., (1992). "6000x2000 Display Prototype." *Proceedings of the SPIE/IS&T Symposium on Electronic Imaging Science and Technology,* San Jose.

[MACK 86] Mackinlay, Jock. (1986) "Automating the Design of Graphical Representations of Relational Information." *ACM Transactions on Graphics,* Vol. 5, No.2. New York, Association for Computing Machinery.

[MACN 90] MacNeil, Ron. (1990). "Adaptive Perspectives: Case-based Reasoning with TYRO, the Graphic Designer's Apprentice" *Proceedings. of the 1990 IEEE Workshop on Visual Languages.*

[MACN 91] MacNeil, Ron. (1991). "Generating Multimedia Presentations Automatically using TYRO, the Constraint, Case-Base Designer's Apprentice," *Proceedings of the IEEE 1991 Workshop on Visual Languages,* Tokyo.

[MARK 90] Marks, J. and Reiter, E. (1990). "Avoiding Unwanted Conversational Implicatures in Text and Graphics" *Proceedings. Eighth National Conference on Artificial Intelligence.*

[MEGG 89] Meggs, P. B. (1989). *Type and Image.* New York, Van Nostrand Reinhold.

[MULL 81] Muller-Brockman, Josef. (1981). *Grid systems in graphic design.* New York, Hastings House Book Publishers, Inc.

[RIES 89] Riesbeck, C. and Shank, R. (1989). *Inside Case-Based Reasoning.* Hillsdale, New Jersey, Lawrence Erlbaum Associates.

[ROBN 91] Robin, Laura. (1991). "Temporal Adaptation of Multimedia Scripts," *Proceedings of the 1991 SPIE Symposium on Electronic Imaging,* San Jose, CA.

[RUBE 88] Rubenstein, Richard. (1988). *Digital Typography: An Introduction to Type and Composition of Computer System Design.* Reading, Massachusetts, Addison-Wesley.

[STEL 80] Steele, G. L. Jr. (1980). "The Definition and Implementation of a Computer Programming Language Based on Constraints," Technical report 595, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[SUTT 86] Sutton, A. and Sutton, M. (1986). *Eastern Forests The Audubon Society Nature Guides.* New York, Alfred A. Knoph.

[WEBS 85] Webster's Ninth New Collegiate Dictionary. (1985). Merriam-Webster Inc.

[WEIT 88] Weitzman, L. (1988). "Designer: a Knowledge-Based Graphic Design Assistant." MCC Technical Report Number ACA-017-88. Austin, Microelectronics and Computer Technology Corporation.