A Graphical User Interface Design Environment

Carl A. Edlund Michael Lewis
edlund@lis.pitt.edu ml@lis.pitt.edu
Department of Information Science
135 N. Bellefield Ave.
University Of Pittsburgh
Pittsburgh, Pa. 15260
(412) 624 - 9426
Fax: (412) 624 - 5231

Abstract

Advances in computer hardware and interface technologies have led to a corresponding increase in research into highly interactive Human-Computer Interfaces. Development tools that create these interfaces need to be flexible, general and produce easily modifiable objects. They also should convey important graphic design principles to the application designer. Existing design tools largely neglect integrating the initial selection of representation with the development of the presentation. The Automatic Graphical Analog Presentation Environment has been designed to produce highly interactive analogical interfaces. A representation is constructively generated from a catalog of primitives and the dynamics of the interface are matched to a formalized description of the user's task. The resulting composite is then evaluated for its cognitive effectiveness and efficiency. Several task domains are identified as initial testbeds for bench marking the implementation of the AGAPE architecture and the effectiveness of the produced displays.

Keywords:

Representation Engineering; Ecological display design.

Introduction

The advent of Direct Manipulation (DM) and visualization interface designs have led to a tremendous advance in the sophistication and complexity of interface development. Command-line or character-menu driven interfaces are no longer sufficient to convey the complexity of information and constraints inherent in many applications. Similarly, research into Representational Aiding has shown that the form in which a problem is represented can have a large impact on the user's performance of a problem-solving task. (e.g. [11]; [8]; [17]; [4]).

The effect of representation on task performance is particularly important for problem domains which require operator-interaction to maintain, monitor, and control a physical system and for interfaces that present information which is tailored to support a problem-solving task. With a DM interface, a user is able to apply their domain expertise directly toward reaching operational goals rather than needing to transform that knowledge based on a particular syntax of communication [22].

For example, airline reservation systems [4], job scheduling [10], and Diagram Assisted Reasoning [7] all require a large degree of interactive manipulation on the part of the user for problem solving. In each of these cases a variety of tabular presentations of the data could have been used by operators to perform a task. However, alternative representations of the same information can provide a clearer and more "usable" form of the problem. Such a representation would enhance the accuracy and performance of the user.

The process of designing visualization-style interfaces that incorporate DM interaction techniques is a critical step for providing a user with effective problem solving tools. However, before an interface specification can be developed, the "context" (a specific representation that the particular interface will be based upon) must be established. For example, designing the presentation of an interface for a process-control monitoring system a number of alternative designs are possible. The determination to use a particular representation, such as single-indicator dials and gauges (Figure 1), as opposed to an equivalent graphical analog (Figure 2) occurs before the interface presentation is developed. The traditional GUI in Figure 1 presents parameters graphically and has virtual controls that can be manipulated directly. However, the underlying system dynamics and constraints that relate the controls and displays are not conveyed. The display in Figure 2 illustrates the difference between interface representations. The interactions between objects, the underlying process, and the operator's actions are integrated into a single graphical display. For

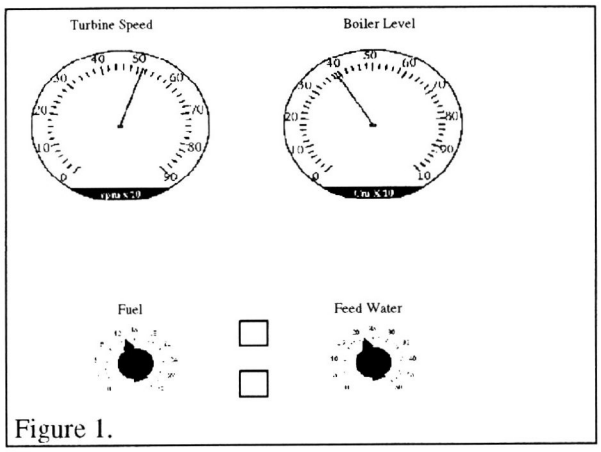

Figure 1.

example, the relationship between fuel-flow and boiler level can be directly perceived from the interface objects. Moving the "fuel" weight to the left increases the rate of fuel-flow and causes the boiler level to decrease and the turbine speed to increase.

Generally, it is the designer's skill and experience in selecting the context for a problem which determines the interface's effectiveness. This process of selection lacks quantitative models which

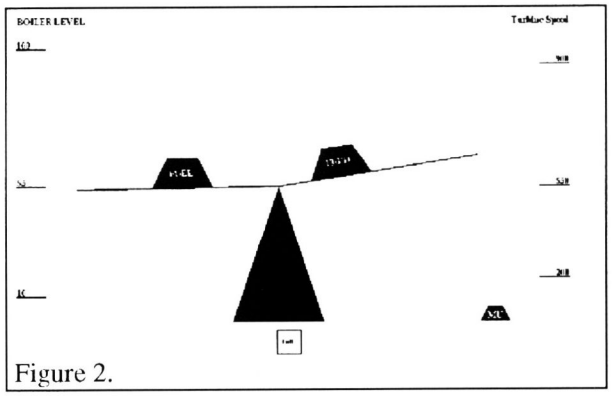

Figure 2.

would support a consistent approach. Models like the Ecological Interface Design (EID) [26] provide a framework for designers to ask questions that reveal the "invisible" constraints and structures which need to be made "visible". However, there currently is no mechanism for explaining how to make these visible or for quantitatively comparing the effectiveness of a representation.

The Automated Graphical Analog Presentation Environment (AGAPE) is a User Interface Design Tool (UIDT) that will assist the interface developer with the selection of the initial representation. The initial representation will then be used to generate an interface specification that can then be prototyped. In order for AGAPE to design the context representation for effective DM and visualization interfaces, a clear and concise model of the user's capabilities and limitations, and the problem task and dynamics is necessary.

User Model and Cognitive Difficulty

Lewis ([9], [10]) proposes an ecological model of *Interactive Situations* which characterize the "directness" and "intuitiveness" of an interface. In an Interactive Situation, a situation theoretic framework [1], [6] is used to model a problem representation, and the interactions of a user are represented based on an Ecological Information Processing (EIP) model. An interface design is treated as a situation that is defined in a recursive hierarchy as a typed relation between a set of typed objects. The user's actions are added as state transitions between related situations. Together these two models provide a framework for deriving a useful measure of the cognitive difficulty associated with a particular interface design.

From the user's perspective, the problem space for the task possesses a certain complexity, size, and shape. Applying situation decomposition methodology [9] to this structure can be expressed as a set of variables (relations) that discriminate states and constraints which govern changes in the values of those variables. Through the selection of the appropriate interface representation the abstract problem can be conveyed to the user in a "cognitively streamlined" manner. The representation for the interface situation should match the actions and effects of the abstract problem task with the user's attunements (expectations about dynamics and structure) and their ability to discriminate states through the use of simple perceptual operations.

The relations to which a user is attuned are determined by the types of objects and relations involved in the interface situation. For example, a situation involving physical objects could be constrained by attunements to exclude events in which two objects occupy the same location at the same time. Additional constraints, can then be imposed on a situation that is already constrained by attunement. In the game of chess, for example, a naive player's attunements would preclude pieces changing in color or shape, while the constraint preventing a knight from moving three squares in a line would require instructions.

It is convenient to think of user tasks as involving three problem spaces:

1. An intuitive space, **A•S**, defined by the problem environment, **S**, and its attunements, **A**.

2. A formal problem space, **C•S**, defined by the task constraints, **C**,
3. and an *effective* problem space, **f•A•S**, in which the intuitive space is augmented with the instructions, **f**, necessary to match the problem space of the task.

If we presume constraint through attunement to be less difficult than constraint through instructions we can rank representations for task difficulty by ordering them according to the difficulty of their instructions. A simple measure of this difficulty is provided by adopting the assumptions of Larkin and Simon [8], and Casner[4]: the difficulty of using a rule is proportional to the perceptual/ memorial operations needed to determine its applicability.

As a result, a variety of analogous interface representations may be matched to a particular abstracted problem description. The process of designing the interface representation for the problem then becomes the selection of the least complex analog. The expression:

$$\mathbf{C \bullet S \equiv f \bullet A \bullet S'}$$

identifies the set of possible analogies for the problem space **C•S**. Analogs are constructed by choosing an appropriate situation, **S'**; identifying the intuitive problem space, **A•S'**, that **S'** determines; and composing the instructions, **f**, which are needed to make the spaces congruent. Although this process is conceptually simple, determining the set of applicable attunements, **A**, and finding the simplest expression for the instructions, **f**, are difficult.

Interface Support of Problem Task

If problem-solving is modeled as traversal through the problem space, then the goal of the task is representable as a target-state. The specification of a particular target-state, or equivalence class of states, is represented as a set of constraints which must be satisfied. The user's problem-solving task is thus represented as a list of additional constraints in the representation of the abstract problem. This produces a unified representation of the problem space, and the problem task.

The design of interfaces that support a particular problem space and task can be divide into two styles [16]. The first is the integration of information in a presentation involving the use of primarily static graphics, such as charts, maps and diagrams. There is a long and robust history of research into effective graphic design techniques for this type of presentation. (e.g. [3], [15], [16], [20], [21], [24]). In most cases this literature focuses on case studies which examine what makes a display "good" and then generalizes a collection of rules for use as guidelines in designing further graphics of the same type. Despite extensive and detailed libraries of guidelines, it is generally still the expertise of the designer that is the telling factor in the creation of effective data visualizations and presentation graphics.

The second category is oriented toward highly interactive problem-solving in which the displays tend to be more dynamic and require a greater degree of DM. The objects and dynamics in Highly-Interactive Graphic User Interface (HI-GUI) directly present the features and dynamics of the underlying system.

The design of interfaces to support these two categories needs to move beyond the paradigm of the Conventional GUI. Conventional GUIs tend to be composed of standard graphic objects like push-buttons, scroll-bars and text-windows that are combined at the task-level into a user interface. The integration of information in the display is only through pre-specified relations, such as the mapping between a scroll bar and movement through a text document, or through the addition of application-specific objects and behaviors, which are expensive to produce and severely limited with respect to cross-domain re-usability. The benefit of this type of interface lies in the relatively simple, and rapid, construction from a standard toolkit. The transfer-of-training benefits for the user arises from their familiarity with a small number of manipulatable analogs which are consistent across applications. In other words, knowledge of push buttons and scrollbars is independent of knowledge of the applications in which they are used.

This kind of interface can be considered as an analogy of fixed-components which are composed according to a set of fixed synthesis rules, style-guidelines, and standards. As a result, it is difficult for these interfaces to display complex constraints and inter-relationships in their underlying system without extensive intervention by the designer. This intervention is exactly what the use of a development tool is attempting to avoid.

GUI Development Systems

Many automated approaches to graphical interface presentation design determine the display largely through the character of the data. Static-display generation systems (e.g. APT [14]) work this way. Systems where the display is additionally determined by a query or user-task analysis include

BOZ [4], SAGE [19], and VIEW [12],[13]. These systems assist the designer by evaluating alternative designs from a limited library of objects or relations.

The integration of design tasks and problem modeling allow for the possibility of automating a larger part of the development process. Such an integrated tool requires merging the UIMS's support environment and automatic interface generation capabilities with graphic design techniques, interface design heuristics, and user and task modeling tools. An integrated tool that merges these capabilities would allow the implications of selecting a particular representation to be automatically incorporated into the design of the presentation. Similarly, the use of special purpose toolkits will provide a means to automatically enforce standards and style guidelines. For example, rules and techniques from graphic design, interface design, standards compliance, human factors and user modeling could be added through modular libraries. The reduction of the required breadth of expertise of the designer allows them to concentrate on modeling issues for the system which is being represented and the tasks and goals of the user.

Generally, any system which attempts to automate the design process for the representation and presentation of graphical interfaces, has certain fundamental considerations which need to be addressed:

1. The interface must be constructed and designed modularly. In order to possess the flexibility, variability and the possibility of generating novel display formats, the representation must be constructed from a library of primitives using substitution and compositional rules.
2. The library of primitives must have a basis in the user's perceptual capabilities and the cognitive and graphic design literature ([5]; [14]; [4]; [23]).
3. The system needs to create a match between the graphical objects being used in the presentation and the objects in the original problem space. The mapping from problem-space to interface-space needs to be made intelligible to the user [9].
4. User-task and viewing goals need to be represented in the problem space. This is equivalent to the concepts of Representational Aiding, and the current emphasis in user-centered design (e.g. [26]). The user's task is taken into account during the initial specification and analysis and is thus pervasive throughout the development process.
5. A method needs to be provided for evaluating, comparing, and verifying the cognitive efficiency of a particular representation.
6. A mechanism for the designer to modify and refine the display must be available. Unit-testing and adherence to style or standard guidelines may require changes to the automatically generated interface. This is a standard component of application development cycles

The AGAPE system represents the integration of these system requirements with the models for situation theoretic problem solving and EIP.

The AGAPE System

AGAPE is a UIDT based on a situation theoretic model of a problem and a user's task, and on an EIP (Ecological Information Processing) model ([9], [10]) of the user. The Interactive Situation model is used to generate and evaluate alternative design representations for a particular problem definition. The input to the AGAPE system consists of a situation theoretic description of the problem situation. This description includes all relevant objects, attributes or features of the objects, the relationships among the objects, and the rule-like constraints which govern state-change in the problem solving space. The output is an interactive graphical situation which is rendered into a prototype graphical interface. Users interact with this interface-situation to explore alternatives, plan, and solve problems that involve the original situation.

The AGAPE system model is designed to automate the task of reformulating a problem space representation by finding analogies which optimize the user's problem-solving performance. AGAPE is distinguished from earlier automatic graphical presentation systems by its ability to graphically convey the possibilities and restrictions on actions and their subsequent effects.

The process of engineering the design representation can be thought of as constructing an analogous problem-solving situation to the original task-situation. The interface *analog* preserves the

relevant distinctions among objects, attributes, and relationships which were specified in the task-situation. In addition the analog is isomorphic to the task-situation with respect to the problem-space graphs. Automated methods of representational change that change the problem structure have been shown to be effective in improving machine problem-solving performance [18].

Although these methods can also be applied to human problem-solving, there is another class of reformulations that involve the context, but not the structure of the problem. This form of reformulation is structurally isomorphic and is related to analogical creation. Representation design for analogical problem representations have also been shown to enhance human problem-solving performance ([11], [8]). The goal of the proposed system is thus to engineer a representation that will enhance human problem-solving, rather than automated problem-solving.

To this end, three principles have been defined and incorporated into this system. The first principle states that those relations or properties which are attuned to by human perceptual capabilities may be substituted for abstract problem constraints. Problem constraints are explicitly-stated, non-perceptual rules that characterize the relations and dynamics of a problem space. Attunements are implicit characteristics of the representation which do not require explicit description.

The second principle is the replacement of non-perceptual constraints by temporal shifts in the interactive situation. The interface dynamics are defined to visibly portray constraints of the problem space through transitions between interactive-situation, and problem-space states.

The third principle is the reduction of the instructional complexity that is necessary for the problem-solver to apply over the course of the task.

The AGAPE Architecture

The overall AGAPE system organization is diagrammed in Figure 3. The functional organization of the system begins with the designer using the Problem Editor to enter a problem specification. This specification is the result of an interactive problem and task decomposition. The problem representation is then written in canonical form as a constrained situation type,

The problem situation is represented internally as an N-tuple in which each dimension is an object's attribute in the problem space. The dynamics of the problem are treated as transitions between states and are specified by a list of actions. Since this list of actions completely determines the behavior and dynamics of the system, the automata can be created as a Finite State Automata (FSA) and the type-declaration is closed under the set of actions.

AGAPE's input from the Problem Editor can then be expanded into a FSA which represents the full problem-space that a user may need to interact with in order to accomplish their task. This graph is important for the evaluation of an analogical situation. Analogs must have at least the same states and transitions as the abstract problem. For a particular analogical situation, the use of explicit rules and property transformations can prune particular states and transitions. This enables an analogical situation to match a large variety of problem situations. Pruning "un-wanted" edges and states through the specification of explicit instructions is accomplished by adding these prescriptive rules to the "error term", **f**. Property Transformations shift the properties and relations of the constraints, **C**, in the problem situation to the attunements, **A**, of the analogical situation, **S'**.

The FSA and the input description are then passed to the "Mind's Eye Planner" (MEP) module. It is here that the representation analogy is constructed through a three phase process. Initially the "Structural Generator" creates a viable state-space. This state-space definition is then passed to the "Relational Matcher" which introduces new properties and relations that operate on its objects. This composition

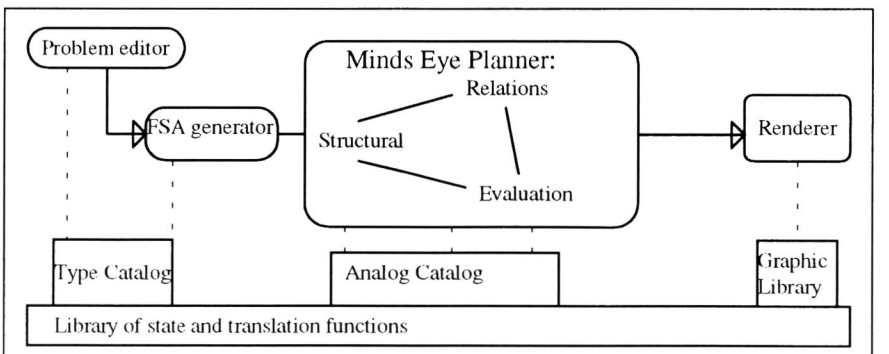

Figure 3. Automated Graphical Analog Presentation Environment

is then evaluated by the "Evaluation Module." Evaluation of the difficulty associated with the proposed analogical situation is based on several criteria. These criteria include the complexity of the necessary instructions and the cognitive difficulty associated with the mappings, objects and relations that are proposed in the analogical situation. This generate-and-test process is then repeated until applicable analogies are no longer generated. At this point, the "best situation seen" (the one scoring lowest in the complexity evaluation) will be passed to the Rendering Module to be prototyped.

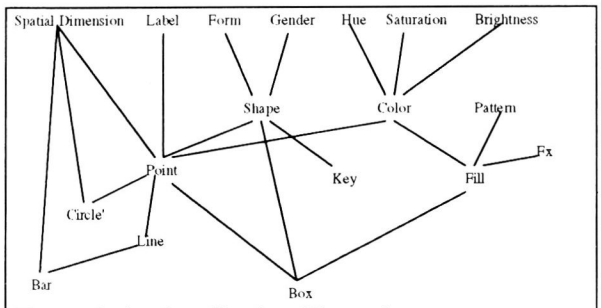

Figure 4. Analog Catalog Hierarchy

The Structural Generator attempts to create an analogical situation that is isomorphic in structure to the original problem. The contents of the Analog Catalog are substituted for the dimensions of the abstract problem specification. These dimensions are then composed into objects which will participate in the analogical situation. This non-instantiated object definition is only structural and is treated as a "type-class" for new objects. The dynamics, properties and relations that will operate on the object will be included in the next stage (relational) of generation.

The Relational Matcher accepts the structural definition of the analogical situation from the Structural Generator and introduces properties and relations to the object-type declarations. The primitives from the Analog Catalog are defined by a set of permissible operations, a set of limitations or constraints on their use, and a set of properties. The Relational Matcher takes these sets and creates a cross-product between the primitives and the constraints listed as part of the constrained situation type. The contents of this cross product are then tested for admissibility into the situation or representation that is being defined.

A property is admissible to the analogical situation if it 1) operates on some grouping of the primitive dimensions that are actively mapped and 2) if it equates to a constraint, property or definition of the abstract problem. The matching of relational properties follows the general procedures outlined in VanBaalen [25]. This methodology defines a "relation" to be a composition of the properties which characterize it. For example, an ordinal relation is transitive, non-symmetric, and non-reflexive. This hierarchy allows relational transformations which shift characteristics from explicit definitions to implicit properties. For example, a binary relation could be transformed into a single parameter function that maps one value to the other. In this way the fixed size property becomes subsumed into the character of the function.

Once a structure has been proposed and the admissible relations determined, the Evaluation Module is invoked. The Evaluation Module generates a difficulty score associated with the cognitive complexity involved in the analogical situation. Derived from the model of Cognitive Difficulty, the measure of difficulty examines several key components of the analogical situation: the mapping from abstract problem to analog; the inherent complexity of the objects and their attributes; and the complexity of the necessary instructions.

Within the Analog Catalog (Figure 4), each analogical primitive has an associated structure in which the parameters for its appearance and functionality are defined. The primitives are organized into two classes: Atomic and Compositional. Atomic primitives consume dimensions of the problem space when selected for use. Compositional Primitives collect primitives (as attributes) into a unified object. For example the Hue primitive is Atomic because it requires the mapping of a problem-space dimension. The Color Primitive however, is Compositional because it combines the renderable properties of Hue, Saturation and Brightness into a single general object that is then used as a sub-component of other Compositional Primitives. "Color" does not force dimensional mappings, but provides a default mechanism both for rendering the un-used components and for modularizing the objects.

Example

An example application from AGAPE may help illustrate the structural composition: a situation where there are temperature sensors at four chemical baths in a manufacturing process and three levels of alarm (None, Notify and Critical). A Notify alarm is upgraded to Critical if it is not acknowledged in a fixed time period. See Figure 5.

	Dimension:	Cardinality
Locations:	Nominal	4 (Bath-1; Bath-2; Bath-3; Bath-4)
Alarms:	Ordinal	3 (None, Notify, Critical)
Temperature	Interval	4 (baths-temperature)

Operator action: Acknowledge the alarm. Alarm status => None

Figure 5. Example Structural Creation

The analog catalog from Figure 4 will be used to develop the representation for this problem. For brevity, only the Point primitive, with its associated sub-components, will be expanded. The sub-components of Point are unable to fully consume the dimensions of the problem. Thus, this example will only produce variations of a Point-based display. A "Point" is defined as having two spatial dimensions, a shape, a label, and a color:

Point(RATIO:X; Y, NOM:Label, Shape(NOM:Form, BINARY:Gender), Color(NOM:Hue, ORD:Sat, ORD:Bri)

The structural matcher will attempt to create a mapping with the three dimensions of the original problem:

Type (Location(N-4), Alarm(O-3), Temperature(I-4))

Mappings that satisfy this condition fill slots in the Point class with all three of the problem dimensions. A "-" indicates a default mapping for the dimension. Among the numerous mappings are the following analogical structures:

= *Point(X(**I-4**), -, -, -, Colr(Hue(**N-4**), -, Bri(**O-3**)))*

-- Temperature is along the x-axis, the tank is color-coded and the Alarm level is the Brightness.

= *Point(X(**I-4**), -, -, -, Colr(Hue(**N-4**), Sat(**O-3**), -))*

-- Temperature is along the x-axis, the tanks are color coded and the Alarm is the Saturation

= *Point(X(**I-4**), -, Label(**N-4**), -, Colr(Hue(**O-3**), -, -))*

-- Temperature is along the x-axis, tanks are labeled by a string and the Hue is the Alarm level.

= *Point(-, Y(**I-4**), -, Shape(Form(**N-4**),-), Colr(-,-, Bri(**O-3**))*

-- Temperature is the y-axis, tanks are identified by Shape, the Alarm is the Brightness of the default color.

= *Point(-, Y(**I-4**), -, Shape(Form(**N-4**),-), Colr(-,Sat(**O-3**),-)*

--Temperature is along the y-axis, the tanks have unique shapes and the Alarm is the Saturation of the default color for the objects.

*Point(-, Y(**I-4**), -, Shape(Form(**N-4**), -), Colr(Hue(**O-3**),-,-))*

-- This permutation is rendered in Figure 6.

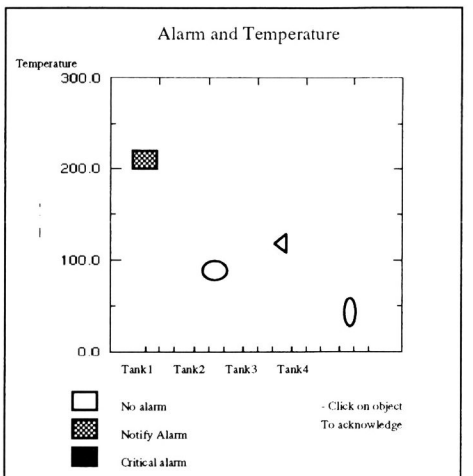

Figure 6. Example Alternative Rendering

For each mapping a set of relations would be introduced based upon the scaling, interactions, and explicit constraints included with the original problem specification.

Conclusions

AGAPE's methods for constructing and assessing the difficulty of analogs are selected for their ability to successfully identify and assess difficulty in a broad sample of previously studied problem representations. AGAPE's methods successfully identify a block-stacking variant of the Tower of Hanoi as its easiest problem representation. AGAPE's methods also correctly order isomorphic versions corresponding to the Monster-Globe (move), and Monster-Globe (change) problems [10].

Presented with the General Job Shop scheduling problem, AGAPE's methods pick the widely used Gantt chart as the most suitable presentation. Also, AGAPE's methods discover a novel representation of a keyed block-stacking task to fit the more tightly constrained Flow Shop problem [9]. AGAPE's methods also find the keyed shapes shown by Bauer and Johnson-Laird [2] to aid subjects in making disjunctive inferences as an effective problem representation. For static data presentation AGAPE will choose representations similar to those preferred by BOZ [4] or Makinlay [14] because AGAPE embodies the same criteria for selecting perceptually efficient representations.

An important contribution of this system is its mechanism for constructing, matching and testing alternative representations.

Acknowledgments

This work was supported by NSF grant IRI-9020603.

References:

1. Barwise, J., & Perry, J. (1983). *Situations and Attitudes.* Cambridge: MIT Press.
2. Bauer, M., & Johnson-Laird, P. (1993). *How Diagrams can Improve Reasoning: Mental Models and the difficult cases of disjunction and negation.* Psychological Science. Volume 4. Number 6. American Psychological Society. pp. 226 - 230.
3. Bertin, J. (1981). *Graphics and Graphic Information Processing.* Berg, W., & Scott, P. (Tr.) publisher: Berlin. New York.
4. Casner, S. (1991). *A Task-Analytic Approach to the Automated Design of Graphic Representations.* ACM Transactions on Graphics. Volume 10. Number 2. April. pp 111 - 151
5. Cleveland, W., & McGill, R. (1984). Graphical Perception: Theory, Experimentation and Application to the Development of Graphical Methods. *Journal of the American Statistical Association.* September. Volume 79. Number 387. pp. 531 - 553.
6. Devlin, Keith. (1991). *Logic and Information.* Cambridge University Press. New York.
7. Johnson, S., Barwise, J., & Allwein, G. (1993). *Toward a rigorous Use of Diagrams in Reasoning about Hardware.* pp. 169 - 212 In Allwein, G., & Barwise, J. (Eds.) *Working Papers on Diagrams and Logic.* Preprint Series. Indiana University Logic group. May.
8. Larkin, J.H. & Simon, H.A. (1987). *Why a diagram is (sometimes) worth ten thousand words.* Cognitive Science. Volume 11. pp. 65 - 100.
9. Lewis, M. (1991). *Visualization and Situations.* In Barwise, J., Gawron, M., Plotkin, G., & Tutiya, S. (Eds.) *Situation Theory and its Applications.* Stanford, California. CLSI Publications.
10. Lewis, M. (1992). *Why are situations hard?*, Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, Bloomington, IN, 939-944.
11. Lewis, M., & Toth, J. (1994). Situated Cognition in Diagrammatic Reasoning. *AAAI Technical Report on Reasoning with Diagrammatic Representations.* SS-92-02. Menlo Park, California: AAAI. pp. 47 - 52.
12. Friedell, M., Barnett, J., & Kramlich, D. (1982). *Context-Sensitive, Graphic Presentation of Information.* pp. 181 - 188. In Journal of the ACM, Siggraph. Computer Graphics Volume 16. Number 3. July.
13. Friedell, M. (1984). *Automatic Synthesis of Graphical Object Descriptions.* Computer Graphics. Volume 18. Number 3. (Proceedings ACM SIGGRAPH '84) July. pp. 53 - 62.
14. Mackinlay, J. (1986). *Automating the Design of Graphical Presentations of Relational Information.* ACM transactions on Graphics. Volume 5. Number 2. April. pp. 110 - 141.
15. Marcus, A. (1987). *Graphic Design for Computer Graphics.* pp. 320 - 326. In Baecker, R. & Buxton, W. (Eds.). (1987). *Readings in Human-Computer Interaction: A Multidisciplinary Approach.* Morgan Kaufmann Publishers, Inc. San Mateo, California.
16. Marcus, A. (1992). *Graphic Design for Electronic Documents and User Interfaces.* ACM press, Tutorial Series. Addison Wesley. New York
17. Newell, A. & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.
18. Riddle, P. (1990). Automating Problem Reformulation. In Benjamin, D. (Ed.) *Change of Representation of Inductive Bias.* (pp. 105 - 123). Kluwer Academic Publishers. Dordrecht.
19. Roth, S., Kolojejchick, J., Mattis, J. & Goldstein, J. (1994). Interactive Graphic Design Using Automatic Presentation Knowledge. In *Human Factors in Computing Systems, ACM-CHI '94 Celebrating Interdependence.* (pp. 112 - 117). Boston, Massachusetts. April 24 - 28, 1994.
20. Schmind, C. (1983). *Statistical Graphics: Design Principles and Practices.* John Wiley & Sons. New York.
21. Seligmann, D., & Feiner, S. (1991). *Automated Generation of Intent Based 3D Illustrations.* pp. 123 - 132. In *Proceedings ACM SIGGRAPH '91: Computer Graphics.* Volume 25. Number 4. July 1991. Las Vegas, NV. July 28-August 2.
22. Shneiderman, B. (1992). *Designing the User Interface: strategies for effective Human-Computer Interaction.* Second Edition. Addison-Wesley.
23. Spring, M. & Jennings, M. (1993). Virtual Reality and Abstract Data: Virtualizing Information. *Virtual Reality World.* Volume 1. Number 1. Spring 1993. pp. c - m.
24. Tufte, E. (1983). *The Visual Display of Quantitative Information.* Cheshire, Connecticut: Graphics Press.
25. Van Baalen, J. (1992). Automated Design of Specialized Representations. *Artificial Intelligence.* Volume 54. pp. 121 - 198.
26. Vicente, K., & Rasmussen, J. (1992). Ecological Interface Design: Theoretical Foundations. *IEEE Transactions on Systems, Man, and Cybernetics.* Volume 22. Number 4. July 1992.