Automated Interpretation of Diagrams for Specification of Medical Protocols

John W. Egar, Angel R. Puerta, Mark A. Musen.

Knowledge Systems Laboratory
Stanford University, Stanford, CA 94305-5479 USA

We are building a visual programming environment that enables medical researchers to organize and describe complex research protocols (specified treatment plans) so that this information can be incorporated into decision-support systems. We have designed our system with a five-layer architecture and a collection of compilers for various visual languages. This design permits the interchange of modules and the coordination of different visual languages within the same system.

1. Objective

Although diagramming environments are often excellent tools for modeling complex relationships, they are also difficult to implement. Our research team is building knowledge-based tools that will help researchers to design and execute medical treatment plans [1, 2]. One of our tasks in this project is to create several visual programming languages that clinical researchers and computer scientists can use to model the problem-solving methods and the medical background information for many protocols in a particular field, and the structure of individual protocols. This paper describes how we have implemented these visual languages.

2. Design Considerations

Previous experience with domain-specific knowledge-acquisition tools [2, 3] has convinced us that flowcharts are a particularly useful way for expert physicians to describe the management of clinical trials for experimental therapies. We are now including classification trees, state charts, entity–relationship diagrams, influence diagrams, and data-flow diagrams to describe the necessary domain information and problem-solving methods [1].

Environments for multiple visual programming languages do exist [4, 5, 6, 7, 8, 9]. With few exceptions [4, 10, 11, 12], these environments have relied to some degree on *syntax-directed editing*. These editors guide the user so that she constructs only syntactically correct diagrams. This strategy requires a custom-tailored editor for each visual language. In its extreme form, syntax-directed editing forces the user to model in a top-down fashion, introducing successively less abstract placeholders according to rigid replacement rules. Because of these limitations, we have elected instead to treat diagrams as *syntax-neutral* documents that are compiled after they are saved.

Certain interface dynamics are also necessary for such a diagramming tool to be successful. The ease and speed of use of the tool is of particular concern. The ability to nest diagrams within nodes and edges, and thereby to avoid much visual clutter, is another criterion. Hypertext links to text and forms are desirable, as is the ability to interchange notational conventions to suit the expert rather than the system. Other design goals include reliability, maintainability, portability, and modularity. Finally, a simple graphical user-interface convention, such as drag-and-drop, is a valuable feature.

3. System Description

The implementation of such a battery of visual languages presents a formidable programming task. After building two custom-tailored diagram editors, we developed the following five-layered approach:

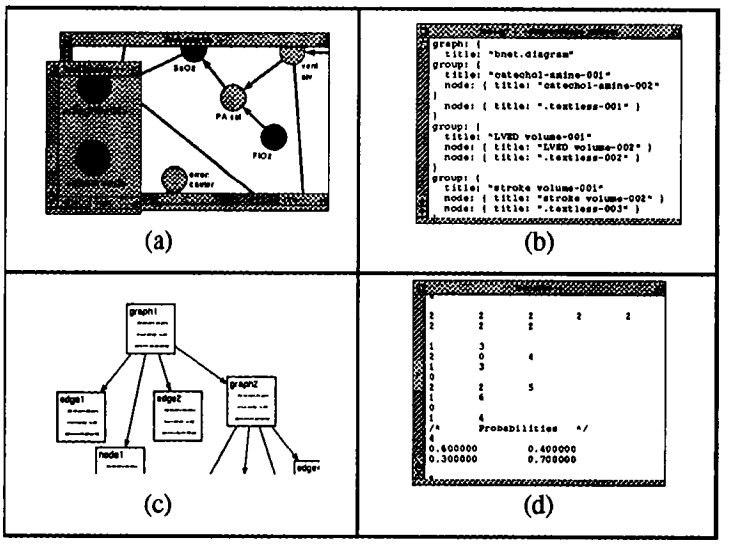

Figure 1. Sample data files passed between each layer. (a) Palette and diagram. (b) GRL file. (c) Network structure. (d) Target code.

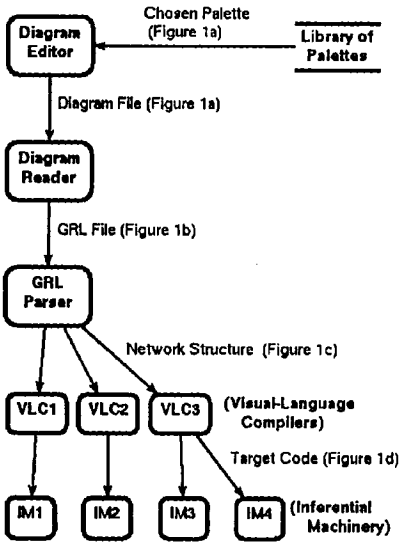

Figure 2. Data-flow diagram of system design.

1. The user interface for all diagramming is a general, syntax-independent diagram editor (Figure 1a).
2. A syntax-independent, but editor-specific, diagram reader generates a high-level, textual description of a saved diagram and its nested subdiagrams. The textual language we use is the Graphical Representation Language (GRL), described by Manke [13] (Figure 1b).
3. A single parser converts the GRL textual description into dynamic objects that maintain information on the position, appearance, labeling, connections, and other attributes of each node, link, and diagram in a nested group of diagrams (Figure 1c).
4. From multiple visual-language compilers, the system chooses one to produce appropriate target code for the individual diagram (Figure 1d).
5. One or more inferential programs, which operate on the target code produced in layer 4, generate parameters and procedures for the knowledge-acquisition tool or for the decision-support system.

The visual appearance of the nodes in a diagram is dictated by the objects in the palette used to construct the diagram. We can create new palettes easily by dragging graphical images into a blank palette. Thus, the domain expert can use those notational conventions with which she is most familiar, and no alterations need to be made to other layers of the system.

When the user has finished constructing a set of diagrams, she selects a compiler button, and the system produces either the appropriate target code or explicit error messages. Currently, the modeling system predetermines which visual-language compiler is used. However, if diagrams are drawn or nested within a diagram of a different paradigm, or if the particular paradigm used is not known in advance, the system should be able to match the appropriate visual-language compiler to each diagram or subdiagram according to attribute clues in the GRL description.

4. Status Report

We have fully implemented the first three of the five layers of our architecture. At the first level, we now use a commercial editor [14], which has a drag-and-drop interface. The three other editors that we tested (two built by our group, and one built by other researchers [15]) were slower and more awkward to use. We have designed and used more than a dozen palettes and paradigms relating to medical protocols. We have tested our amended version of GRL, and have found it sufficiently general to describe all relevant aspects of these diagramming paradigms. We have built a GRL parser, and have tested it with several visual languages. Furthermore, we have determined that the network of objects that the GRL parser creates is adequate for the three visual-language compilers that we have implemented.

Developing the visual-language compilers (in layer 4) is more problematic. Formal models of textual languages are well understood, and tools exist that facilitate compiler generation for these languages. General models for visual languages, however, are fundamentally more complex: Symbols in visual languages must be parsed according to their position, connections, size, appearance, and orientation with respect to other symbols, whereas symbols in textual languages are related only by concatenation. Researchers have only recently developed formal grammars and tools that aid in the generation of visual-language parsers [10, 11, 12, 16]. After we programmed compilers for belief networks, state-transition diagrams, and procedural flowcharts, several common graph-manipulation motifs emerged. For example, parsers of directed-graph languages tend to identify originating and terminating nodes, and then to simplify intermediate paths according to a few syntactical rules for the node types. We realized that these syntactical rules were easiest to describe declaratively, and that we might be able to generate visual-language parsers with such descriptions. Consequently, our system is converging on a graph-grammar production system, where declarative, subgraph-replacement grammars support the automatic generation of visual-language compilers.

5. Discussion

While implementing a visual programming environment for the design of clinical protocols, we have learned that the *responsiveness* and the *expressiveness* of such a modeling tool must be emphasized, even at the expense of user guidance. This shift in emphasis has led us away from syntax-directed, interpretive editors, and toward our present syntax-neutral, compiled environment. By employing just one syntax-neutral editor, we maintain a single interface behavior, despite the multiplicity of languages. By separating out the compilation of diagrams from the process of editing them, we allow faster editing, and, hence, offer a more responsive tool. Since the notational conventions for each visual language are specified in replaceable palettes, we can adapt our visual languages to the user's preferences. By translating diagrams and palettes from all the different languages into a common representation (GRL) we can insulate a user's choices of editor and palettes from the system designer's choices of visual-language compilers and inferential machinery. We can easily incorporate into these layers software produced outside our laboratory. Although we can express all visual languages that we have so far needed in modeling clinical protocols by using a node-and-link editor [14], we could introduce a different type of editor should the need arise. Thus, in addition to enhancing responsiveness, this layered design makes our system easy to maintain.

Our compiled approach does, however, introduce the need for visual-language parsing. Further, to provide the expressiveness needed to describe complex protocols, we must have *many* visual languages. Visual-language parsers are sometimes challenging to implement, but the ones that we have built have common features that make a formal graph-grammar production system a promising strategy for building other parsers. We believe that the specification and generation of visual-language parsers through attributed graph grammars will yield higher-quality compilers with far less programming effort, as compared to ad hoc programming. Such a specification could also lead to formal testing of the models constructed through diagrams. Given a clear mapping from medical ideas to the kinds of representations used in our decision-support system, we might be able to discern patterns and to detect errors in our computational models of clinical protocols.

6. Future Plans

We plan to develop a large library of visual-language compilers from declarative specifications. The visual languages will be used by clinical researchers to model medical information, data structures, and problem-solving methods. We believe that the heterogeneous nature of medical reasoning will require this variety of languages. Whether such an arsenal of tools will be sufficient, and which diagramming paradigms will be most successful for modeling clinical-trial management, remain to be tested.

We shall continue linking diagrammatic languages to existing programs, and to inferential programs associated with knowledge-acquisition tools and their targeted decision-support systems. The complex constraints that we have identified in those clinical protocols that we have already investigated suggest that an iterative cycle of modeling, implication viewing, and debugging may be necessary to achieve a consistent and reasonable protocol design. Hence, our plans include the implementation of visual languages for the acquisition *and* presentation of constraints.

We aim to implement at least a sufficient set of programming languages for the design of clinical trial protocols. Presumably, the same tools can be used to test how much textbook information, medical-record information, and medical problem-solving information can be expressed clearly through diagrams.

Acknowledgments

We conducted this work with the support of the National Library of Medicine under grants LM-05157, LM-07033, and LM-05208, of the Agency for Health Care Policy and Research under grant HS06330, and of the Digital Equipment Corporation.

We are most grateful to Alan Chung and Jonathan Schwartz for helping us with the Diagram! editor, to Josef Schreiner for guiding our visual programming research, to David Combs for building the foundations for this research, and to Lyn Dupré for providing editorial assistance.

References

[1] Egar J W, Puerta A R, Musen M A. Diagrammatic Acquisition of Medical Information for Clinical Trials. KSL-Report 91-59, Knowledge Systems Laboratory, Stanford University, Stanford, CA, October 1991, submitted to MEDINFO 92: Seventh World Congress on Medical Informatics.

[2] Musen M A. An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools. *International Journal of Man–Machine Studies* 1989, 31:673–698.

[3] Musen M A, Fagan L M, Combs D M and Shortliffe E H. Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool. *International Journal of Man-Machine Studies*. 1987, 26:105-121.

[4] Chang S-K. A Visual Language Compiler for Information Retrieval by Visual Reasoning. *IEEE Transactions on Software Engineering* 1990, 16:1136–1149.

[5] Lakin F. Spatial Parsing for Visual Languages. In Chang S-K, Ichikawa T, Ligomenides P A (eds). *Visual Languages*. New York: Plenum Press, 1986:35–85.

[6] Backlund B, Hagsand O and Pehrson B. Generation of Graphic Language-Oriented Design Environments. SICS Research Report R-89-8911, Swedish Institute of Computer Science, Stockholm, Sweden, 1989.

[7] Arefi F, Hughes C E and Workman D A. Automatically Generating Visual Syntax-Directed Editors. *Communications of the ACM* 1990, 33:349–360.

[8] Borges J A, Johnson R E. Multiparadigm Visual Programming Language. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*. Los Alamitos, CA: IEEE Computer Society Press, 1990:233–240.

[9] Casner, S. Building Customized Diagramming Languages. In Chang S-K (ed). *Visual languages and visual programming*. New York: Plenum Press, 1990: 71–95.

[10] Helm R, Marriott K and Odersky M. Building Visual Language Parsers. In Robertson S P, Olson G M, Olson J S (eds). *Human Factors in Computing Systems: CHI '91 Conference Proceedings*. Amsterdam: Addison-Wesley 1991:105–112.

[11] Nagl M. A Software Development Environment Based on Graph Technology. In Ehrig H, Rozenberg G, Rosenfeld A (eds). *Graph-Grammars and their Application to Computer Science: Third International Workshop*. New York: Springer-Verlag, 1987:458–477.

[12] Golin E J, Reiss S P. Parsing in a Visual Environment. Technical Report No. CS-89-06, Dept. of Computer Science, Brown University, Providence, RI, February, 1989.

[13] Manke S and Paulisch F N. Graph Representation Language: Reference Manual. On-line documentation, Department of Informatics, University of Karlsruhe, Germany, December, 1990.

[14] Steele K, Abell T, Chung A, Kedoin R, Rosner R, Ryan R, Schwartz J and Skinner B. *Diagram! V. 1.1*. Lighthouse Design, Ltd., Chevy Chase, MD, 1991.

[15] Paulisch F N and Tichy W F. EDGE: An Extensible Graph Editor. *Software Practice and Experience*, 1990, 20:63–88.

[16] Costagliola G and Chang S-K. DR Parsers: A Generalization of LR Parsers. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*. Los Alamitos, CA: IEEE Computer Society Press, 1990:174–180.