Toward the Development of Highly Interactive Software Visualization Systems: A *User-Centered* Approach

Christopher D. Hundhausen
Visiting Fulbright Scholar
Department of Computer Science
University of Karlsruhe
Federal Republic of Germany
E-mail: chundhau@ira.uka.de

1 Introduction

Predicated on the intuitive idea that a mapping between an executing program and computer graphics can give one insight into the program´s dynamic behavior, software visualization (SV) systems purport to provide techniques for facilitating four central activities involving that mapping: its (1) design, (2) specification, (3) observation, and (4) manipulation. Early SV systems such as Balsa (Brown 1987) and Tango (Stasko 1990) defined conceptual models in which different actors performed those activities.. In the Balsa model, for example, *client programmers*—*algorithmaticians* and *animators*—were responsible for designing and specifying the algorithm-to-graphics mapping, whereas the *end users*—s*cript authors* and *script viewers*—actually observed the mappings (see Figure 1). Although, in theory, the same person could have assumed all four roles, in practice they were played by different people, and at different times.

That division of labor can be attributed, in the main, to the sharp difference in the levels of expertise that the Balsa system demanded of client programmers and end users. Because they needed to possess a detailed knowledge of both an underlying graphics system and an animation language, client programmers required a much higher level of expertise than end-users, who required no knowledge whatsoever of either the algorithm or the techniques used to animate it, in order to work with the Balsa system. Such a conceptual model, which encourages SV's central activities to be conducted by different people, and at different points in time, has been the dominant SV system paradigm for nearly a decade.

Recently, however, SV researchers have begun to explore the benefits of an alternative SV model—one that encourages SV's central activities to be undertaken by the same person (see Figure 2). Mukherjea and Stasko (1993a) have recently described the first system to adopt such an alternative conceptual model. Their Lens system defines a graphical environment into which an algorithm may be interactively loaded; users can then manipulate the user interface to define a mapping between the algorithm and graphics, and run the algorithm to observe the mapping.

This brand of SV—what I shall call *highly interactive SV* (HISV) here—has formed the focus of my research during the past year. Notice that if SV is to be manifested in a HISV system, its users can engage in at least four distinct activities, which correspond closely to the four central activities of SV identified above:

(1) conceptualizing a visualization for the program;
(2) mapping that conceptualization to the visualization language defined by the system;
(3) manipulating the system's user interface to specify the visualization; and
(4) manipulating the system's user interface to observe and explore the resulting visualization.

Together, these activities form an experimental framework —one centered on the system user—that I have found to be quite useful in exploring the problem of HISV.[1] Below, I begin by examining the motivation behind the development of HISV technology. I shall then discuss my past research, results, and insights vis-á-vis each activity in my experimental framework, in an attempt to illustrate its usefulness as an approach to HISV.

---

[1] I am indebted to Professor Sarah Douglas for this perspective on HISV, which evolved out of my independent study work with her at the University of Oregon during the spring of 1993.

2 Why HISV?

Recently, history was made in the SV community when Stasko, Badre, & Lewis (1993) (henceforth SBL) published the first ever empirical study to explore the effects of an SV system on learning. Although the study failed to provide concrete empirical evidence in support of SV,[2] it did provide its authors with some interesting insights into "the use of algorithm animation as an instructional aid and the conditions under which algorithm animations may be most beneficial" (p. 65). In analyzing the failure of the study's text-and-animation subject group to perform as well as expected, SBL offered what I believe to be a remarkable, albeit intuitive, insight: "For [one] to benefit from [an algorithm visualization], [one] must understand both [the algorithm-to-graphics] mapping and the underlying algorithm upon which the mapping is based" (p. 65). Proceeding from this insight, the authors conjectured that the study's text-and-animation group may not have benefited as much from SV precisely because it did not understand the algorithm-to-graphics mapping.

Considering the SV technique used in the study, I do not find it surprising that the text-and-animation group would have had trouble comprehending the mapping. Text-and-animation subjects employed the Tango animation system, whose conceptual model accorded with the traditional SV system paradigm presented in Figure 1. As *end users*, they did not take part in the design or specification of the algorithm's mapping to graphics; that mapping was defined by a Tango *client programmer* before the study took place. Thus, even if they were able to obtain some kind of understanding of the algorithm from the texts they read on it, text-and-animation subjects had no concrete basis for comprehending the mapping. We see, then, that the traditional SV system conceptual model (see Figure 1)—and, in particular, the division of labor it encourages between client programmer and end user— may have prevented the text-and-animation subjects from obtaining all of the benefits that SV may have to offer.

As I see it, the important lesson of the SBL study is that, at least within an educational setting, SV system developers ought to ensure that their users understand both the algorithm being visualized, and its mapping to graphics. Under the assumption about "educational setting" made by the SBL study—viz., that system users will learn about an algorithm through some other source before visualizing it—that task boils down to ensuring that system users understand the algorithm-to-graphics mapping. As SBL themselves point out, the logical way of elucidating the mapping is to allow users to construct the mapping themselves. In other words, an SV conceptual model that bridges the gap between client programmer and end user by encouraging SV´s central activities to be undertaken by the same person may prove more useful than the traditional conceptual model within an educational setting.. As SV researchers, we are thus provided not only with compelling motivation to explore the potential of such an alternative conceptual model, but also with the challenge of manifesting that conceptual model in a usable system.

3 A *user-centered* experimental framework for exploring HISV

Inspired by the SBL study, I have been actively involved in HISV research during the past year. As someone well aware of the porous theoretical foundation upon which SV has been forwarded,[3] I have been committed to employing empirical studies as the basis of my investigation. My meta-goal has been to bring the perspective of HCI empirical studies research to bear on the development of HISV systems. As I shall explain below, I have been using *constructive interaction* (CI)[4] in combination with *conversational analysis* (CA),[5] to study the four HISV activities presented above, and their interrelations. The ultimate goal of this research is two-fold: (1) to advance the state of the art in *empirical evaluation* of interactive visualization software, an area in which very little has been done; and (2) to advance the state of the art in HISV systems.

3.1 The Empirical Method: Conversational Analysis of Constructive Interaction

Borrowed from a branch of sociology called *ethnomethodology*, the CA/CI combination I have been using has been advocated by several researchers within the HCI community (see, for example, Suchman 1987; Roschelle 1990; Douglas 1992). In brief, CI entails videotaping two person, same-sex teams as they collaborate to complete a given exercise—often involving an experimental software system. Since teams members must work together to complete

---

[2] For an analysis and critique of the study, see (Hundhausen 1993a).

[3] For a thorough investigation of the problem, see (Hundhausen 1993b).

[4] See (Miyake 1986).

[5] See (Suchman 1987).

the exercise, they often make inferable, if not explicit, their understanding or misunderstanding of the tasks and concepts involved. Using CA techniques pioneered by Suchman (1987), software designers can then perform detailed analyses of these episodes; such analysis has already given software designers insight into (1) the ability of their software to act as a resource in mediating and facilitating shared understanding[6]; and (2) the points at which breakdown of the interface occurs, and possible design solutions that can remedy those breakdowns.[7] Below, I illustrate the technique's usefulness as a basis for exploring HISV.

3.2 Applying CA/CI to HISV

My past year's research activities have used the four central activities of HISV as a framework for exploring the problem of HISV. In particular, I have been involved in two separate lines of experiments: one that has considered activity (1) (conceptualization), and one that has considered activities (2) and (3) (mapping to visualization language, and specification). Below, I briefly describe both of these projects, as well as their most interesting results. To round out the framework, I touch briefly on activity (4) (manipulation), although I have not yet explored this activity experimentally.

Conceptualization

Under the direction of Professor Sarah Douglas at the University of Oregon, my research on algorithm conceptualization originally began as an apprenticeship in using the CA/CI combination to evaluate the usability of interactive software. As the basis software for the usability study, I obtained an experimental version of *Lens*, an interactive SV system developed at Georgia Tech University.[8] In exchange for the use of the system, I agreed to provide the system's authors with a thorough usability report on the system.[9]

After our first experiment, in which we videotaped a team of users as they defined a bubblesort animation in Lens, we decided that a slightly modified study format might give us insight into two questions important to SV in general: (1) How do humans conceptualize algorithms in the first place?; and (2) What implications might such conceptualizations have for the design of HISV systems? To get at these questions, we added a preliminary "construction paper" exercise to our original Lens experiments, in which we provided our next two pairs of subjects with several art supplies—a full spectrum of colored construction paper, a scissors, and different colored pens—and charged them with the following task: "If you were to explain the bubble sort algorithm to someone who had never seen it, how would you do it? Feel free to use any of these resources (construction paper, scissors, etc.) to assist you in your explanation."

In comparing the characteristics of our subjects' homemade and Lens-defined visualizations, we found striking differences, which led us to conclude that (1) our subjects' homemade visualizations possessed qualities that were beyond the capabilities of Lens; and (2) the way in which subjects naturally specify such visualizations using paper and pencil differs greatly from the method for specifying a visualization in Lens. Based on the results of this experiment, I have formulated the following hypothesis:

> ***The informed SV system hypothesis:*** A SV system that allows its users to specify non-problematically visualizations that accord with their own conceptualizations of the program will be more usable than a system that does not.

In future research, I look forward to working toward the confirmation or disconfirmation of this hypothesis.

Specification (language + interface)

For the past five months, I have led a research group that has been focusing specifically on the problem of specifying an algorithm-to-graphics mapping. Having chosen graph algorithms as our domain of interest, we have already conducted two experiments revolving around the specification of a visualization for the Ford-Fulkerson algorithm (which computes the maximum flow on a flow network). Our goal for these experiments has been two-fold: (1) to

---

[6] See, for example, (Roschelle 1990), pp. 21–24.

[7] See, for example, (Douglas 1992), p. 3.

[8] For details on the system, see (Mukherjea & Stasko 1993b).

[9] For complete details of the study, see (Hundhausen 1993c).

identify the visual language in terms of which humans think about and express visualizations of graph algorithms; and (2) to learn about how best to manifest such a language in an HISV system.

In the first experiment, we gave our subjects complete freedom in specifying such a visualization. The most significant observation we made was that the visualization they designed relied heavily on the *direct manipulation* of the graphical representation of the graph that they created. In our second experiment, we used a white board to set up a hypothetical SV world, complete with the source code, a magnetic source code pointer, and various graphical materials (translucent magnetic paper, colored magnets, and colored markers) for designing a visualization. Having familiarized themselves before the experiment with (1) a C++ implementation of the Ford-Fulkerson source code; (2) an example graph on which to consider the algorithm; and (3) three existing visualization specification methods—annotation, declaration, and direct-manipulation[10]—our subjects were asked to design and specify a visualization for the Ford-Fulkerson algorithm within our hypothetical SV world.

The specification method they chose turned out to be a mixture of annotation and direct-manipulation. Furthermore, their specification employed an intriguing three-stage mapping process. In the first stage, they highlighted blocks of code, and associated an abstract event name with each block. Next, they opened up a new window, in which they rewrote the algorithm in terms of those abstract event names; the resulting code strongly resembled a high-level pseudo-code description that one might find in a textbook. Finally, referring to a graphical representation of the target graph, they associated graphical events with each of the abstract events.

Two preliminary results of this still ongoing series of experiments stand out. First, users like to describe visualizations by manipulating concrete graphical objects—an observation that my earlier experiments with bubblesort would confirm. Indeed, in all of the experiments, subjects chose first to draw a concrete graphical representation of some abstract program object, and then to define the visualization by directly manipulating that object to depict algorithm events. Second, it appears that not everyone thinks about an algoirithm in terms of the same interesting events; abstract events that one user might take for granted in defining a visualization—for example, the breadth-first search in the Ford-Fulkerson algorithm—might play an important role in another user's visualization of the algorithm. Thus, it seems that an SV system ought to allow its users to map the algorithm to graphics at varying levels of abstractions. In future research activities, I look forward to developing a prototype system based on these observations, and to using it as a basis for further experiments.

Manipulation

Although I have not yet conducted any experiments that directly address the activity of visualization manipulation, I suspect that users might appreciate the ability to click on interesting components of a visualization to obtain additional information. What that information might be, how users might use it, and how it could be specified in activities (2) and (3), remain interesting topics for future work.

4 Conclusion

In the preceding paragraphs, I have used the results of SV's only published empirical study as a basis for motivating an alternative conceptual model for SV systems. To explore the potential of that alternative model, I have proposed an experimental framework based on the SV system user's central activities, together with an empirical research method for examining each of those activities. I believe that my initial experiences with using the CA/CI technique to study visualization conceptualization and specification not only well illustrate my approach's value, but also point to the need for further empirically-based SV research, which should play a central role in the development of more usable HISV systems.

[10] To learn about these three specification methods, our subjects studied brief descriptions of each method adapted from (Roman & Cox 1993).

References

Brown, M. (1988). *Algorithm animation.* Cambridge, MA: The MIT Press.

Douglas, S.A. (1992). Using conversational analysis to discover breakdown in the design of user interfaces. *Proceedings of CSCW Workshop on Ethnographic Methods and Design.*

Hundhausen, C.D. (1993a). Subverting the comparative research paradigm: The potential for ethnomethodology in evaluating the effects of algorithm visualization on learning. Unpublished graduate seminar paper, Department of Computer & Information Science, University of Oregon.

Hundhausen, C.D. (1993b). The search for an empirical and theoretical basis for algorithm visualization Unpublished graduate seminar paper, Department of Linguistics, University of Oregon.

Hundhausen, C.D. (1993c). Exploring the Potential for Conversational Analysis in the Evaluation of Interactive Algorithm Visualization Systems. Unpublished master's final project paper, Department of Computer and Information Science, University of Oregon.

Hundhausen, C.D., & Malony, A.D. (1993). ObjectView: A software design architecture for breakpoint-based program visualization. Technical Report CIS-TR-93-22, Department of Computer and Information Science, University of Oregon, Eugene, OR.

Miyake, N. (1986). Constructive interaction and the iterative process of understanding. *Cognitive Science 10*, 151–177.

Mukherjea, S., & Stasko, J.T. (1993a). Applying algorithm animation techniques for program tracing, debugging, and understanding (pp. 456–465). *Proc. 15th IEEE International Conference on Software Engineering*, Baltimore, MD,

Mukherjea, S., & Stasko, J.T. (1993b). Lens: A visual debugging environment. Technical report, Georgia Institute of Technology.

Naps, T.L, & Hundhausen, C.D. (1991). The evolution of an algorithm visualization system (pp. 259–266). *Proceedings of the 24th Annual Small College Computing Symposium*, Morris, MN.

Roman, G.C. & Cox, K.C. (1993). A taxonomy of program visualization systems. *Computer* (December), 11–24.

Roschelle, J. (1990). Designing for conversations. Paper presented at the AAAI Symposium on Knowledge-Based Environments for Learning and Teaching, Stanford, CA.

Stasko, J. (1990). Tango: A Framework and System for Algorithm Animation. *Computer 23*, 27Ð39.

Stasko, J., Badre, A., & Lewis, C. (1993). Do algorithm animations assist learning? An empirical study and analysis (pp. 61–66). *Proceedings INTERCHI '93*, Amsterdam, The Netherlands.

Suchman, L. (1987). *Plans and situated actions: The problem of human machine communication*. New York: Cambridge University Press.

Figures

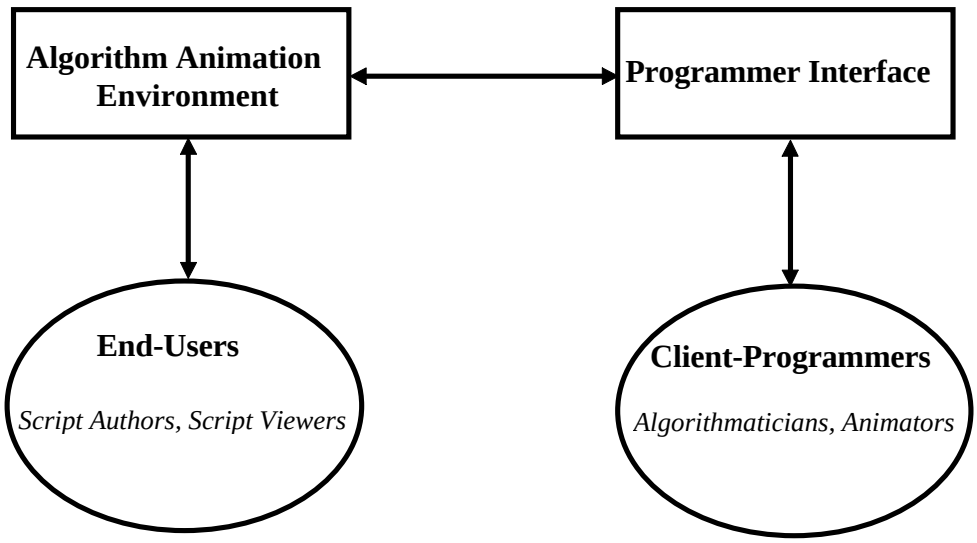

Figure 1. The Balsa conceptual model

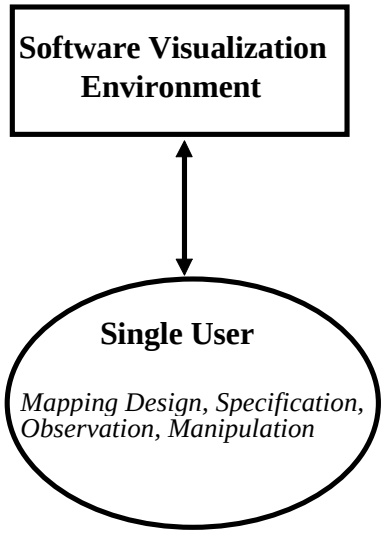

Figure 2. The HISV conceptual model