Helping System Users to Be Smarter by Representing Logic in Transaction Frame Diagrams

David Cox[1] and Simon Polovina[2]

[1] www.flipp-explainers.org
djcox@fuse.net
[2] Communication & Computing Research Centre
Faculty of Arts, Computing, Engineering & Sciences
Sheffield Hallam University, UK S1 1WB
s.polovina@shu.ac.uk

Abstract. We identify a lucid way of conveying complex information to users in a highly visual, easy to follow form. As explanation, we describe several ideas about system user instructions. Several key ideas are clarified using diagrams. A direction for exploration is offered, with the view that ICCS conferees will be aware of a simple, diagrammatic way to explain use of systems dealing with very complex real world problems.

1 Introduction

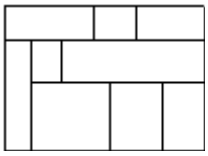
The connection between diagrams and logical reasoning is well-established [2]. User instructions for new systems in health care, science, education, and government, for example, become unclear when complex choices – like complex traffic intersections -- appear in single-path text pages rather than in roadmap, diagram form.

Patterns and rules. In contrast, instructions for using complex systems often have proven clear when presented in two parts: first, as landscape or architectural views of intersection *patterns* like baseball diamonds, soccer playing fields, or chess boards -- with, second, ultra-simple *rules* on moving through the patterns on the playing field or game board [1]. People are able to deal with extraordinarily many different paths to reach football goal lines, soccer nets, and baseball home plates. But, in text form, these myriad lines of possibility are 100% invisible behind the single-line disguise text always insists on wearing.

System by game. Driving a car is a familiar example of "system by game" [1]. The game board is the streets and highways on which the car is driven by its driver. The *patterns* are formed by the painted lines which define traffic lanes. Sometimes lane patterns split like logical *ors*; sometimes they merge at intersections like logical *ands*. The driver's goal is the destination of the trip. The *rules* are the traffic laws. The driver is truly reacting to patterns on a game board while applying the rules of the particular driving game being played. Text can *describe* patterns but can't *model* them. Diagrams can do both.

U. Priss, S. Polovina, and R. Hill (Eds.): ICCS 2007, LNAI 4604, pp. 460–463, 2007.

2 A Simple Game Board Diagram Example



On the left is an illustrative, content-empty example of a *type* of simple game board-like diagram – a type that often has been applied to real world systems. The site www.flipp-explainers.org demonstrates that *all* user instructions can be represented by the words, *"Start at the top, move down, don't cross vertical lines, end at the bottom."* In this example, 9 frames form 11 multi-frame user pathways.

Diagrams of this type can handle vast complexity. Figure 1 highlights an example of the MS-DOS operating system's DIR command that shows 9,700,000 process variations. (At www.jfsowa.com/figs/flippdir.gif it is reproduced online; we acknowledge the support given to us by John F. Sowa in general with our work [7].)

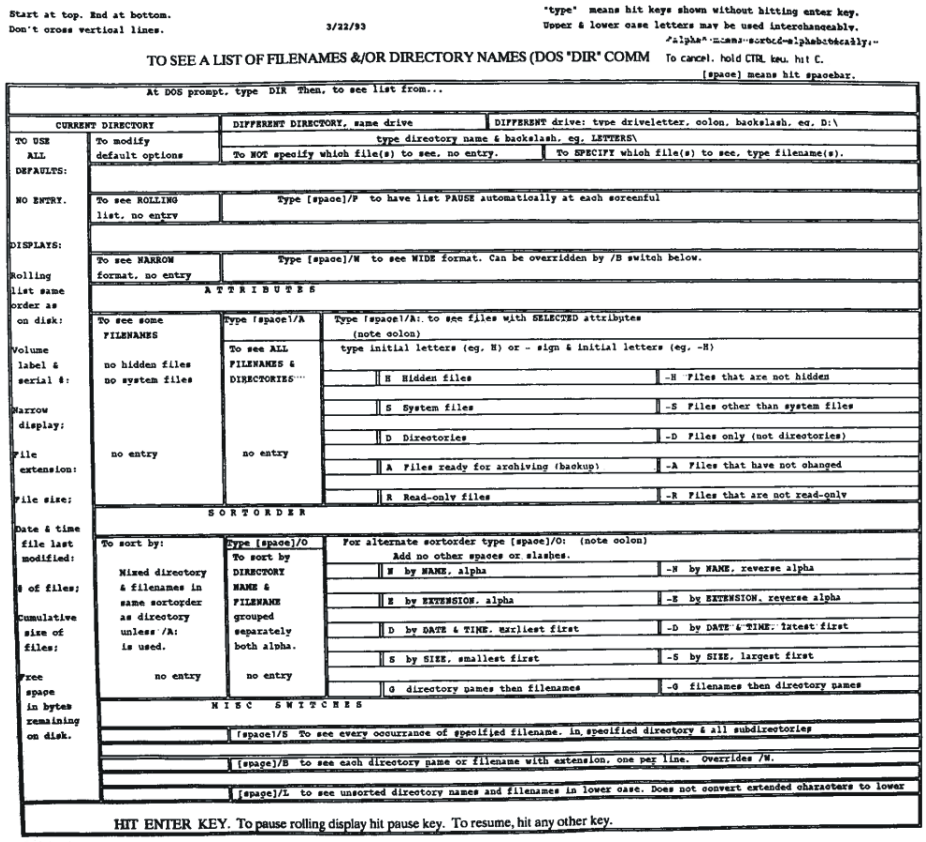


Fig. 1. DOS DIR command; not as straightforward as you might think...

Some actual system applications have used many more scenarios. A new manufacturing plant's 26 departmental system diagrams together represented one coherent system at startup. The number of diagrammed scenarios was – probably well

into the billions [1]. Even relatively simple applications have been seen by users as complex. An actual case study involving a tax calculation system with only eight scenarios had nonetheless been considered even by teachers to be quite complex [1].

3 Some Suggestions

Whilst [1] explains how to develop these panoramas of all scenarios in any given system, the following suggestions are worth noting:

Create many expectations; deliver on all of them. Build confident expectations for system users. A way to do this is by diagramming for users *all* scenario patterns in any system on interest, not just those that answer specific situations. This philosophy has proven very helpful in quick teaming of people who did not know each other. Hundreds of temporary creative problem solving teams welcomed being given abundant clear expectations about, for example, the team processes and tools before they worked together [1].

Represent user logic without language, symbols, or formulas. They *create* complexity. While the simple framework diagrams described above clarify logic relationships, the information *inside* the frameworks – the *content* -- can be in any form, any languages, any symbols, any formulas, any logic, any images, etc. Luckily, a given diagram holds content correctly even in different languages and forms. Logic is *form and connection*, not language, not symbols, not content [1].

Use diagram types that are both logical and convenient. Whilst Flow chart diagrams are perhaps the most popular means of describing complex processes, they suffer in that, among other things, they:

- don't show flow direction. Top-down is not standard, for example.
- don't have rules as to where entry and exit points are located (top, sides).
- don't reveal what, if anything, may be flowing along connector lines.
- don't always display user paths.
- don't put full information in boxes – often only one-word labels.
- don't use direct-connected frames.

Avoid throwing user instructions 'over the wall' to whom they may concern. Avoid one-way, truncated instructions. Design instructions so every available scenario is obvious to users. Instructions that work can create confidence.

See instruction frames as describing two-way transactions. The idea of frames as transactions was prompted by work by Hill [3] and Polovina [4]. Individual frames can be understood as ideally containing two-way transactions between a system and its users. Frames, as used here, seem to have no parallel in language. Since frames are not word- or sentence-limited, they are not like *phrases* in language, music, and art. Empty frames, like intersections, have no language counterpart. Frames can hold any mix of sentences, phrases, formulas, symbols, different languages, music, images, etc. While each frame prompts user action, a text sentence may carry no such implication. This means text's capacity for transaction territory-marking is about zero – or even negative when arousing user ire.

Attach 'local' definitions of concepts to system user instructions. Such definitions could include symbols and terms like: *logic*, *system*, *content, scenario, system user* (who might be system designers; system architects; the client who pays to have the system designed and maintained; the system inheritors; the system overseers; the system accountants and bankers; the system managers the trainers of system administrators – sometimes even the public. The opportunity for wrong assumptions and confusion is real. Definitions demystify.

Emphasize user logic; soft-pedal system logic. Whilst describing to users what a system is doing during its operation is common, it is often irrelevant. Users want just whatever logic controls their success. Note that what is sequencing through user logic structures is not information but rather the attention of the user. Meaning can arise from seeing closely related alternative scenarios that may work better. Users prefer the panorama of all scenario paths experts follow. Unfortunately, this contrasts with -- for just a few examples -- Microsoft's Word 2003, Norton's 2007 Internet Security, Adobe's Reader 7, and Google's Desktop – none of which display any user scenarios at all, let alone any panoramas.

4 Concluding Remarks

Given these experiences we suggest this direction for exploration:

What seems unavailable and urgently valuable is a computer capability with which almost anyone can create and conveniently revise diagrams where contiguous logic scenario panorama structure is retained automatically. This might be a program for self-adjusting diagrams as simple as children's hopscotch game diagrams with automatic logic-rediagramming. Basing it on producing transaction frame FLIPP Explainer diagrams is one obvious approach [1].

We accept that in this short paper we cannot properly convey the potential benefits, other than highlighting some of the pertinent issues. We are nonetheless of the view that our approach will provide users with a simple framework to tackle hitherto complex real world problems. Its further exploration by a wider community would therefore be well rewarded.

References

1. Cox, D.: Explanation by Pattern Means Massive Simplification (an E-book), http://www.flipp-explainers.org
2. Dau, F.: The Logic System of Concept Graphs with Negation: And Its Relationship to Predicate Logic. In: Dau, F. (ed.) The Logic System of Concept Graphs with Negation. LNCS (LNAI), vol. 2892, Springer, Heidelberg (2003)
3. Hill, R.: A Requirements Elicitation Framework for Agent-Oriented Software Engineering – Doctoral dissertation. Sheffield Hallam University (2007)
4. Polovina, S., Hill, R.: Transactions Framework for Effective Enterprise Management. In: ICCS 2007 Workshop Proceedings. Springer, Heidelberg (2007)
5. Text vs. patterns demonstration: http://www.flipp-explainers.org/demonstration.htm
6. Case study application: http://flipp-explainers.org/casestudy1.htm
7. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing (2000)