Combining systems and manuals†

Harold Thimbleby
Department of Psychology,Stirling University, STIRLING, Scotland, FK9 4LA
Tel: +44 786 467679; FAX: +44 786 466741
EMail: hwt@compsci.stirling.ac.uk

Like many interactive systems, hypertext is operated by button pressing. It is therefore possible to combine an interactive system with its own hypertext manual. Numerous advantages follow: adaptive intelligent interactive help; correct documentation, in natural or mathematical language; automatic generation of conventional manuals optimised for various tasks; and detailed analysis.

This paper motivates the approach, and describes a representative system, *Hyperdoc*. Hyperdoc enables research questions about good user interfaces and good user manuals to be investigated.

Key words: hypertext, manuals, finite state machines

1. Introduction

User interfaces are getting more and more features, but there are very few systematic ways to make them any better. User interfaces should be empirically evaluated and then improved, but in practice many products are designed and then sold to users, with little opportunity for improvements. It is therefore crucial to have support for usability analysis at design time. This paper is about Hyperdoc, a user interface shell for simple interactive systems, and what can be done with it. Hyperdoc runs interactive systems combined with their manuals. It provides interactive help, but it can also generate conventional manuals following any specified criteria. It has turned out to be surprisingly flexible. Hyperdoc's main purpose, though, is to facilitate design analysis.

Hyperdoc achieves solid results with precise but general assumptions, and enables certain usability criteria to be assessed as early as possible in the design cycle. It provides a flexible framework where research into usability can be pursued effectively.

Much as we might covet many modern powerful user interface development systems, they make little contribution to design unless we have access to them. Hyperdoc is an accessible design tool: competent programmers reading this paper could implement Hyperdoc within a few days.

2. Hyperdoc overview

Hyperdoc is a user interface shell that simulates a large class of user interfaces, specifically finite state machines. Hyperdoc combines the system simulated with its user manual. It does this so closely that adding or deleting sections in the user manual extends or contracts the user interface, and conversely, modifying the user interface modifies the manual. The manual

---

† To appear in *Human-Computer Interaction'93*, Proceedings BCS Conference HCI'93, Loughborough, UK.

therefore always corresponds exactly to the system it describes. If desired, Hyperdoc allows the simultaneous use and design of a system, thus greatly easing iterative approaches to design. A system designer can modify the finite state machine while it is being used: so easily, there isn't even a separate 'mode' needed to modify the machine. This is extremely useful for interactive development and evaluation when working with potential users.

Hyperdoc supports sophisticated analysis of the user interface. The analysis can be used in several ways. First, Hyperdoc can provide adaptive intelligent help for the simulated system (context-sensitive help is generated automatically, with no effort from the designer). Secondly, Hyperdoc can generate conventional paper manuals optimised in any defined way. (It can construct 'minimal manuals' and protocols for 'cognitive walk throughs.') Thirdly, Hyperdoc can perform usability analysis. Finally, most interestingly, Hyperdoc can be used to find ways to improve the usability of a system, which it can do so by automatically modifying the system — which, of course, modifies the manual (and intelligent help) appropriately and automatically.

A separate system has been designed that 'drives' Hyperdoc to provide demonstrations of it. Demonstrations can do anything to Hyperdoc that a user can do to the simulated system, as well as programmed things that are synchronised with the system being demonstrated. (It is possible to make demonstrations continue from where a user is in the actual system. Alternatively, by tracking what a user does, a demonstration can give the user a guided tour of functions they have not yet tried.)

Hyperdoc's approach is to stay within the 'limitations,' the clarity and tractability, of finite state machines. This has the immense benefit of importing a huge body of well known theoretical concepts and measures. It guarantees any results, practical or theoretical, are not tied to an idiosyncratic framework or implementation, and it guarantees results can be widely understood, tested and applied. The restriction to finite state machines somewhat restricts the realism of simulations: Hyperdoc does not attempt photographic realism, though to do so would not interfere with the features it does provide.

Most real gadgets are more complex and quirky than Hyperdoc can simulate. Hyperdoc does not have any mechanism for timeouts (where the system makes state transitions autonomously, typically, when the user does not press any button for several seconds). Most such features that might have been introduced have questionable value in improving user interfaces.

We have not paid any attention to details of natural language generation. We can generate badly written manuals! With very little effort, we could generate badly written manuals in several languages — though this isn't quite the joke it seems, since nationalisation of manuals is normally an unreliable and an expensive process delaying product delivery.

Hyperdoc is implemented as a HyperCard system, combined with LISP and Mathematica for more sophisticated support than is feasible to implement in HyperCard alone. Though HyperDoc provides all the interactive simulation and help facilities directly, it loads and saves LISP or Mathematica expressions to define the combined system and manual. Thus Mathematica can be used to modify a system design, perhaps to optimise for some mathematically-formulated usability criteria.

3. Background and related work

Hyperdoc continues a line of work (Thimbleby & Witten, 1993).

Systems have been combined with manuals before. Literate programming (Knuth 1992) is an example of combining systems and their *internal* documentation. Literate using (Thimbleby, 1986) was suggested as a way of combining interactive systems with their *user* documentation, in order to be able to generate reliable examples in the documentation.

Hyperdoc is an extension of literate using, where we consider the user manual to be a hypertext.

Both STATEMATE (Harel, *et al.*, 1990) and Hyperdoc share the advantage of maintaining system and documentation together; however, in STATEMATE the documentation is not interactive in its own right. STATEMATE is aimed at developers; Hyperdoc is aimed at users developers, and researchers.

Hyperdoc currently provides shortest paths to the user's goals, but other schemes were discussed in (Thimbleby, 1991b) and shown to be a contribution to solving the 'getting lost in hypertext' problem. The notion of doing non-trivial things in hypertext is implicit in programmable hypertext systems, such as HyperCard (Coulouris & Thimbleby, 1993).

Since Hyperdoc permits users or designers to modify a system, or its manual, or to execute it, in any order, it is *equal opportunity* (Runciman & Thimbleby, 1986). Manual and system have equal opportunity to participate in the overall design; either can be considered the output generated by the other.

Buchner and Funke (1993) have done empirical work on the usability of finite state machines. Further discussion of simple interfaces can be found in Thimbleby (1991c) and some principles for more sensible design are discussed in Thimbleby (1991a; 1993a).

4. Hyperdoc features

Hyperdoc has been used to study several video recorders and other systems. In the discussion that follows, we will draw on simulations of video recorder front panels.

Hyperdoc's user interface looks like the simulated system: buttons can be labelled and positioned on a control panel, which adjusts its size to fit. Feedback is textual (symbols can be used if a suitable font is available), though global audio feedback can be selected (e.g., key beep on state change). Hyperdoc also provides menus, for design, for analysis and for exploring intelligent help.

4.1. Intelligent help

Conventional manuals suffer from synchronisation problems: the user reads one section of the manual, which may refer to a state of the system other than the one it is in. Inevitably, the user will make mistakes. (This problem is considerably worsened by timeouts in the system, where it makes state changes without telling the user!)

Hyperdoc provides two sorts of menu for intelligent help: the user can ask 'how to?' and 'why not?' questions. There is no synchronisation issue.

There are two ways to ask 'how to' questions. The **How?** menu requires an exact question, such as: "how do I get the VCR to auto-stop recording in 210 minutes?" The **Subtasks** menu, more conveniently, breaks questions down into 'subtasks,' which enable the user to ask how to change part of the system's state, such as: "how do I get the VCR to pause (leaving whatever else it is doing as unchanged as possible)". In this example, the user would only select pause from the menu's choices. The subtasks correspond, in some ways, to an 'ideal' choice of buttons for the VCR.

Very often a system's peculiar design results in the user using it sub-optimally. I have been using a VCR for two years, and Hyperdoc surprised me with a better way of getting a tape out and switching off when playing a tape. To do this, which I have done many times, I had always pressed **stop/eject** to stop the tape playing and a second time to eject the tape, then **operate** to switch off. Hyperdoc's two-press **How to?** answer was, "You are playing a tape; how can you off with tape out? Press Operate. Press Stop/eject." In some sense I knew this better way, since I had told Hyperdoc exactly how the VCR worked!

To ask 'why not' questions, Hyperdoc provides only one menu, similar to the **How?** menu. (A subtasks-type menu would not be difficult to implement.) The user selects a goal they wanted to achieve from the **Why not?** menu, and Hyperdoc attempts to answer why they are not doing that. Typical answers mention that: the user has not yet reached the goal (i.e., more needs to be done); pressing some button got the user further from the goal (i.e., the user may have made an earlier mistake); or that a button press had no effect. The **Why not?** menu then provides an option to help the user achieve the goal they presumably desired to reach (reverting to the **How?** answer form). Here is an example **Why not?** answer, "Why not rewind a tape? You tried pressing 'Rewind' but it had no effect. Instead of pressing 'Record' before that, you should have pressed 'Stop/eject'. Do you want to know how to rewind a tape now?"

Hyperdoc answers questions by using a shortest path algorithm. The hypertext manual is used to construct coherent answers. **How?** questions are based on the shortest path from the current state to the desired state, whereas **Why not?** questions are answered by reversing and weighting the user's path to the current state negatively, then asking a **How to?** question. A 'why not?' answer therefore includes the possibility that it might have been better to have done something different earlier, which the negative weights cater for, rather than to progress only from the current state. To stop explanations reaching too far back into the past, which the user might have forgotten, weights decay. (We found weighting the last action minus one, and earlier ones infinity gave adequate answers. The decay should depend on the structure of the simulated system.)

As ideas for further work: Note that weights can be varied, either to suit the overall aim of the system or specific questions, or dynamically. For example, if weights are reduced every time the user makes a transition, then Hyperdoc's answers will tend to be expressed in terms of routes with which the user is familiar. The ratio of repetition (for learning) and efficiency (for doing) could be varied to suit the user or short- or long-term performance criteria. The help answers can therefore be designed to optimise performance for given tasks: learning or action, reference, emergency response, fault diagnosis, or for machine-related criteria (such as long life).

Or again: expert users might train the system (say by solving typical problems), then the advice Hyperdoc provides would teach paths used by experts in their solutions. Since Hyperdoc can save system definitions (including the hypertext material), it is an easy matter to present the interactive help as conventional manuals, arranged, for example, 'as an expert would solve the problems,' or whatever, depending on the weight training adopted.

Hyperdoc currently provides only one recommendation in its interactive help, though the user can choose to remember it or to get Hyperdoc to do it (after giving any advice, Hyperdoc can perform the suggested operations itself). More interestingly, Hyperdoc could provide alternative advice (e.g., chosen for quick action, or for familiar operations). The user could then choose which approach they want at the time.

4.2. Analysing designs

Hyperdoc provides several interactive ways of analysing the system design it is simulating. All analysis can be expressed in terms of application-oriented concepts extracted from the combined hypertext manual.

Hyperdoc can check trivial properties, such as whether the system is strongly connected. This is useful during development of a system, to avoid trap states. Hyperdoc can also perform structural modifications, such as contraction (combining states) and Cartesian multiplication (combining two finite state machines in all possible ways).

More advanced properties are determined by running programs in other systems, such as Mathematica and Combinatorica (Skiena, 1990).

As well as obvious measures (such as the probability that a randomly pressed button does something, which is proportional to the mean out degree), very many numerical (and other) measures readily suggest themselves. We mention just five:

The all-pairs shortest path is the minimum number of button presses to get from whatever the system is doing to anywhere else. The mean of this is a measure of how much a user needs to know to solve any problem; multiplied by the number of states gives a measure of how much the user needs to know to solve all problems. For the few systems we have been able to compare, the value decreases convincingly with informal evaluations of usablity. The logarithm (base number of buttons) of the number of states is the theoretical minimum, and it is rarely reached.

The optimal Chinese postman tour is the minimum number of button presses to check that the user correctly understands the system. It gives a theoretical low bound on the time taken for a usability test, and can be shown to be (almost always) vastly lower than the expected length of a random test procedure (adequate user testing of a system must take years, even for simple systems).

The logarithm of the chromatic number is the minimum number of indicators that ensure some visible change on any state transition. One can easily count or identify state transitions that are inadequately coloured (which can confuse the user because indicators do not change).

How good is the system at supporting error recovery? One less the maximum over all states of the minimum length of cycles passing through each adjacent pair of states is the best worst-case number of actions the (knowledgeable!) user has to embark on to recover from a single ‘overshoot’ accident. (A correct undo makes this value one.)

One may wish to design a system so that advice is, so far as reasonable, independent of the initial conditions. There is a trade-off in how to achieve this. One system we examined could only achieve advice similarity if all advice started, “Switch off”; but another we examined instead had almost as many buttons as features, and this meant that regardless of its state a button generally achieved the same-named goal. The theoretical measure relating to this is diversity.

Finally, mathematical analysis does not preclude visual analysis (though, we will omit pictures here). Mathematica can readily generate conventional circle-and-arrow drawings. A useful approach is to delete all ‘similar’ edges (such as to the off state), since these have little structure and only serve to clutter diagrams. Ranked embeddings are a variation where the position of the state is such that it shows, for example, the least number of button presses to get to that state from a designated set of states. Some systems are clearly lop-sided when represented like this, and this may indicate a design problem, or a trade-off that could be — or should have been — analysed.

4.3. Combining system and hypertext

Both interactive systems and their documentation are frequently criticised for being difficult to understand and to use (Thimbleby, 1993a). Many user manuals do not even fully and correctly describe the systems they attempt to document. Designing good interactive systems is extremely difficult: some poor manuals can be explained by the obvious difficulty of clearly and correctly explaining a badly designed system.

Hypertext is a form of finite state machine, where button presses take the user to new states and new text becomes visible (in some systems, graphics and sound may also appear). It is consequently possible to construct the hypertext manual of an interactive system so that their finite state machines are isomorphic, and then they can be combined. The result is an interactive system with hypertext manual, *but equally*, a hypertext manual that is the interactive system it describes — it becomes context sensitive help. When the structure is the

same, there is no need to distinguish the buttons of the system and the buttons of the hypertext manual.

Hyperdoc can derive conventional manuals automatically. Conventional paper manuals are trees in more ways than one: they are made up from sections, containing subsections, containing sub-subsections, to paragraphs of text. A hypertext document, however, is a graph (a user of hypertext interacts with the corresponding finite state machine that traverses the graph as buttons are pressed). A conventional manual is an embedding of the hypertext graph into an ordered tree, typically also a spanning tree of that graph. For any one hypertext there will be very many trees: optimal manuals therefore need to be selected carefully In fact, the poor structure of conventional manuals is a reflection of the difficulty of performing this optimisation. Better, then, if it were done automatically.

If a user is to know the most efficient way of using a system in every eventuality, the all pairs shortest paths reference manual is appropriate, though lengthy. For non-trivial systems more structure is required. For example, the following heuristic generates a 'minimal manual' as a minimal spanning tree: (*a*) the documentation of a vertex is its label and labelled set of out edges; (*b*) edges, from vertices spanned by a subtree, with a common end vertex are documented at the root of that subtree; (*c*) the number of edges is minimised. The manual is constructed from a preorder walk of the tree; subtrees generating sections, subsections and so forth. (There is an interesting similarity with such an approach and a heuristic for drawing statecharts.)

It is a routine task to identify bridges, vital edges and hinge vertices in a graph. These concepts correspond to critical concepts in manuals (e.g., without knowing about a bridge a user cannot reach a subgraph). The critical components can be weighted to be described first, or otherwise highlighted. Minimising the depth of a bridge in a manual corresponds to enabling the user to know how to switch between 'modes' easily; maximising corresponds more to training wheel systems (Carroll, 1990), where it is assumed that the user should be protected from components on the far side of bridges.

Since manual material is generated automatically, it precisely defines the system, and, being readable, is ideal to check out with clients who would have been intimidated by a conventional formal definition of equal precision and accuracy. However, the manual need not be generated in natural language. It can be generated as a production system, for example, if this is more familiar or more useful to the designer; certainly doing so can enhance design insights (Monk, 1990).

The complexity measures (length or weight) of optimal manuals can be considered complexities of the corresponding systems. Clearly, one easily can obtain the measures without generating the manuals. Since manuals are generated from objective criteria, claims about manual design (for example, that redundancy should be removed) could be tested rigourously. Arguably, the most robust empirical results in user interface design are about manuals (Oatley & Draper, 1992). We now have a way of making manuals and systems indistinguishable: therefore these results now apply to the (appropriate class of) interactive systems. More-or-less: *designing the manual designs the system.*

5. Manual generation experiments

At present, we do not know how to characterise good manuals in a way that can be formalised, though Carroll's work (1990) is extremely promising. Experiments along the lines suggested above show that minimal manuals, even for small systems, lack clarity. We need to discover whether this is because of using the wrong objective functions, whether redundant recoding is essential, whether structure should be weighted, or whether the systems we have analysed are poorly designed and therefore don't admit of good manuals. See example text below, which is extracted from a LISP program output.

For the following (play a tape fast forward; pause playing a tape; play a tape fast backward) you can press **Play** to play a tape:

If you are playing a tape, but have paused it, additionally you may:

Press **Forward** to fast forward.

Press **Rewind** to rewind a tape.

For the following (play a tape fast forward; play a tape fast backward) you can press **Pause** to pause playing a tape:

If you are playing a tape fast forward,

you cannot do anything else.

If you are playing a tape fast backward,

you cannot do anything else.

Typographical layout of manuals is essential (André (1989) shows a badly set manual that contributed to death). Hyperdoc enables typography to be automated, and directly and reliably related to the semantics.

6. Conclusions

Hyperdoc represents a new way of managing system development, combining both the technical design and analysis *and* the manual design. Hyperdoc is based on the idea of tightly integrating interactive systems and hypertext manuals.

There is now no reason why manuals have to be written after the system (this is the conventional software development position: you cannot write a definitive manual until after a system is implemented and fixed). Designing a manual does not now have to wait until after the system design is finalised. Improving the manual readily leads to improvements in the system.

With Hyperdoc, manuals can be generated automatically for a variety of purposes: from user manuals optimised for certain tasks, through pictorial maps, to formal manuals written in mathematical notations augmented (as required) by natural language explanations. All such manuals are certain to be correct.

Hyperdoc makes no distinction between using a system and developing it. Even a running simulation can be modified. This is extremely useful in user testing; for example, an unclear help text, or an obscure button action, can be fixed and retested immediately.

As an interactive system itself, Hyperdoc suggests new ways of developing interactive systems with on-line help. Hyperdoc certainly makes available analytic tools (which in principle were available to designers anyway), and closely ties analysis into design (as when Mathematica is used to modify a design following some rule). What Hyperdoc uniquely does is to make design much more pleasant, much more reliable, and much more flexible. It opens up the design process by combining the currently sequentially applied skills of system designer, manual writer, and evaluator; it therefore greatly increases the scope of worthwhile user contribution earlier in the design process.

Many people who have used Hyperdoc have asked why its user interface is not better. The answer is that, so far, it has been used to faithfully simulate and analyse existing system designs. That its user interface then appears poor suggests, not a limitation in Hyperdoc, but that conventional systems could easily be improved. Using Hyperdoc encourages designers to improve user interfaces — because of the ease of doing so, because combining manual and system encourages one to simplify the system design to make the manual more comprehensible, because Hyperdoc provides a powerful interaction framework that directly suggests improvements of the simulated system, and because of readily accessible analysis.

Finally an ironic point, illustrating the foregoing claim. HyperCard was used to implement Hyperdoc, and with no effort on my part, it provided all simulated systems with an undo.

Thus undo (and some other desirable features) can be provided at no cost — and yet undo is not provided on any of the systems we have studied.

Acknowledgements

Many people have suggested ideas that have enhanced Hyperdoc. The following suggested ideas when I was attentive: Stuart Anderson, Andy Cockburn, George Coulouris, Alan Dix, Steve Draper, Joachim Funke, Michael Harrison, Peter Ladkin, Gary Marsden, Andrew Monk, Prue Thimbleby, Nina Warburton.

References

André, J (1989) "Can Structured Formatters Prevent Train Crashes?" *Electronic Publishing — Origination, Dissemination and Design*, 2(3), pp.169–173.

Buchner, A & Funke, J (1993) "Finite-state automata: Dynamic task environments in problem-solving research," *Quarterly Journal of Experimental Psychology*, 46A(1), pp.83–118.

Carroll, J M (1990) *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press.

Coulouris, G F & Thimbleby, H W (1993) *HyperProgramming*, Addison-Wesley.

Draper, S W & Oatley, K (1992) "Action centred manuals or minimalist instruction? Alternative Theories for Carroll's Minimal Manuals," in *Computers and Writing, State of the Art*, Holt, P O'B & Williams, N, editors, pp.222–243, Intellect Press.

Harel, D, Lachover, H, Naamad, A, Pnueli, A, Politit, M, Sherman, R, Shtull-Trauring, A & Trakhtenbrot, M (1990) "STATEMATE: A working environment for the development of complex reactive systems," *IEEE Transactions on Software Engineering*, **16**(4), pp.403–414.

Knuth, D E (1992) «*Literate Programming*», Center for the Study of Language and Information, Stanford University.

Monk, A F (1990) "Action-Effect Rules: A Technique for Evaluating an Informal Specification Against Principles," *Behaviour and Information Technology*, **9**(2), pp.147–155.

Runciman, C & Thimbleby, H W (1986) "Equal opportunity interactive systems," *International Journal of Man-Machine Studies*, **25**(4), pp.439–451.

Skiena, S (1990) *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley.

Thimbleby, H W (1986) "Experiences with literate programming using CWEB (A variant of Knuth's WEB)," *Computer Journal*, **29**(3), pp.201–211.

Thimbleby, H W (1991a) "Can Anyone Work the Video?" *New Scientist*, **129**(1757), pp.48–51.

Thimbleby, H W (1991b) "Heuristics for cognitive tools," in NATO ASI Series F, *Proceedings NATO Advanced Research Workshop on Mindtools and Cognitive Modelling, Cognitive Tools for Learning*, Kommers, P A M, Jonassen D H & Mayes, J T, editors, pp.161–168, Springer Verlag.

Thimbleby, H W (1991c) "The Undomesticated Video Recorder," *Image Technology*, Journal of the British Kinematograph, Sound and Television Society (BKTS), **72**(6), pp.214–216.

Thimbleby, H W (1993a) “The Frustrations of a Pushbutton World,” *Encyclopaedia Britannica Yearbook of Science and the Future*, pp.202–219, Encyclopaedia Britannica.

Thimbleby, H W & Witten, I H (1993b) “User Modelling as Machine Identification: New Methods for HCI,” *Advances in Human-Computer Interaction*, H. R. Hartson & D. Hix, editors, **IV**, pp.58–86.