Formal user models as a design tool for interactive systems

Richard Butterworth Ann Blandford David Duke

1998

WP13

Principal Investigator Prof Ann Blandford Middlesex University
Research Fellow Dr Richard Butterworth Middlesex University
Academic Collaborator Dr David Duke University of York
Research Fellow Jason Good Middlesex University
Industrial Collaborator Sue Milner Praxis Critical Systems Ltd
Academic Collaborator Prof Richard Young University of Hertfordshire

http://www.cs.mdx.ac.uk/puma/

Contact: Prof Ann Blandford
School of Computing Science
Middlesex University
Bounds Green Road
London. N11 2NQ. UK.
tel: +44 (0)181 362 6163 Project funded by
fax: +44 (0)181 362 6411 EPSRC grant
email: a.blandford@mdx.ac.uk number GR/L00391
Formal user models as a design tool for interactive systems

Richard Butterworth  Ann Blandford  David Duke

Abstract

We discuss and exemplify how formal user models can be integrated into a formal development life-cycle. Doing so allows for usability criteria to be addressed during design as opposed to in a post-hoc manner. We specify two web browsers and then show how a very simple user model can be used to comparatively analyse the usability of the two browsers.

We purposely use models no more complicated than is required by the evaluation. We discuss how these simple models can be refined in order to give more detailed (yet more costly) evaluations.

1 Introduction

User models allow software engineers to consider usability early in the design life-cycle. We assume that the design life-cycle is a process of iteratively proposing, evaluating and refining an abstract system model until the model is so refined that it is considered to be an implementation. Any model can be refined in several ways, each refined model being the start of a new refinement path. A design decision must be made about which refinement is the ‘best’. Traditionally software engineers make design decisions based on such criteria as code efficiency, portability etc. Usability requirements can be expressed early in the design life-cycle but it is not clear how the design process can take those requirements into account. Usability evaluation tends to be left until after implementation when usability is assessed by user testing.

This paper looks at how user models can be employed to evaluate various abstract models for their potential usability. A user model helps expose potential usability problems with device specifications. They do not do away with the need for user testing, but should help remove large usability ‘bugs’ before implementation, thereby removing the need for (very costly) implementation redesigns.

In [BBD98] we show how very abstract user assumptions can be used to evaluate a device model against general usability properties (a usability property typically being a statement of the form ‘the user finds things she wants to do easy to do’). This paper takes that work a step further; showing how a more operational (but still very simple) user model can be used to evaluate more specific properties. Our analyses are motivated by the principle of Occam’s razor; we only include such detail in the user model as is necessitated by the analysis.

Here we look at how the user converts a plan of action into the actions themselves, and show some of the usability problems that can be exposed just by this simple mechanism. We do not claim the mechanism described here to be a complete user model in any sense; it could be considered to be part of a richer user model. Consider Norman’s [Nor86] seven stages of task performance (depicted in figure 1). The mechanism we describe here (which we refer to as the planned action mechanism) is only that in the shaded area; i.e. how the user translates plans into actions. How the plans are formed, or where goals come from
is not our concern here, so we do not model it. We do, however, model how the current device state can be used to guide this translation from plans to actions (hence the double headed arrow between execution and the device). The mechanism can therefore capture more reactive behaviour [Suc87] than Norman’s original model.

This paper address interaction itself rather than interface issues. We assume the interface to be transparent, i.e. a complete projection of the underlying device state. How the interaction is affected by impoverishing this projection due to screen real estate would give us another level of evaluation, but one that we only touch upon here.

We first define two web browser device models (section 2) and then the planned action mechanism which captures our user assumptions (section 3). We then combine the device and user assumptions to give two system models, one for each device model (section 4). We then describe a usability evaluation on the two systems (section 5), suggesting which system is more usable and making explicit the qualifying assumptions we made in the evaluation. We then outline the strategy we used to evaluate the system models and state this strategy generally so that it can be applied to any iteration in the design life-cycle (section 6). Finally we discuss what we have done and look at ways in which the models and analyses can be refined (section 7).

The formal notation we use is the Temporal Logic of Actions (TLA) [Lam94]. In TLA both system models and properties are expressed as temporal logic formulae. Lamport gives a thorough introduction to TLA in [Lam94] but it should be possible to understand this paper with a basic knowledge of temporal logic and formal specification languages.
2 Two device models

Much of the formalisation presented in this section is similar to that in [BBD98]. However here we formalise two web browsing devices, one with a stack based history mechanism (formalised as a ‘stack with pointer’ in [DM98]) and one with a queue based mechanism, similar to that proposed by [TG97].

Before describing models of web browsers we must first describe the web that they browse. The web is modelled as a directed graph; web pages are nodes and links are directed arcs. We do not commit to what a page actually is; we simply assert that they exist, that there is a set of them that make up our web and that there is a set of links between them.

\[
\text{Page} \doteq \ldots \\
\text{pages} : \mathcal{P}(\text{Page}) \\
\text{links} : \mathcal{P}(\text{pages} \times \text{pages})
\]

We treat \textit{links} as a relationship; \textit{p links q} asserts that \((p, q) \in \text{links}\). \textit{i.e.} there is a link from page \(p\) to page \(q\).

\textit{pages} and \textit{links} are ‘fixed variables’; they do not change value during interaction, so this model does not cover more ‘dynamic’ webs exemplified by search engines.

2.1 The device state

The state of the browser is described by two variables; \textit{history} and \textit{index}. \textit{history} represents the history (or ‘go’) list maintained by the browser. It is a common mistake for users to believe that the history list is a complete record of pages the user has visited during their interaction; it is, in actual fact, an ‘edited highlights’ list. One of the pages on the history list is the current page, and the variable \textit{index} denotes the position in the history list of the current page.

\[
\text{history} : \text{Page}^* \\
\text{index} : \mathbb{N}
\]

The variable \textit{dev} denotes the device state, \textit{i.e.} the above two variables.

\[
\text{dev} \doteq (\text{history, index})
\]

Also of use to us is the fixed variable \textit{home} which denotes the home page that the browser starts with as the current page.

\[
\text{home} : \text{Page}
\]

2.2 Device actions

Both modelled browsers allow actions that move \textit{index} backwards and forwards along the history list. (The TLA action notation uses a primed variable to denote the value of the
variable after the action. Unprimed variables denote their value at the beginning of the action. Hence \( \text{index}' = \text{index} + 1 \) denotes the action of incrementing \( \text{index} \).

\[
\begin{align*}
\text{Bwd} & \triangleq \text{index} > 1 \land \\
& \quad \text{history}' = \text{history} \land \\
& \quad \text{index}' = \text{index} - 1
\end{align*}
\]

(8)

\[
\begin{align*}
\text{Fwd} & \triangleq \text{index} < |\text{history}| \land \\
& \quad \text{history}' = \text{history} \land \\
& \quad \text{index}' = \text{index} + 1
\end{align*}
\]

(9)

A browser allows the user to jump from the current page to any page linked to it. It is when a jump takes place that ‘editing’ of the history list occurs. The Netscape mechanism for doing this is to treat the history list as a stack; it throws away any pages in the history list after the current page and then adds the new page to the end of the list.

Tauscher and Greenberg [TG97] recommend a more queue-like mechanism with a maximum length of around 8 pages. If a page that is already on the history list is Jumped to then the list is unaltered and \( \text{index} \) is set to point to that page. Newly jumped to pages are added to the end of the list and old pages are lost from the beginning when the list is greater than 8 pages long.

We define two actions \( \text{Jump}_1(p) \) and \( \text{Jump}_2(p) \) to describe each of these mechanisms.

\[
\begin{align*}
\text{Jump}_1(p : \text{Page}) & \triangleq \text{history}(\text{index}) \text{ links } p \land \\
& \quad \text{history}' = \text{history}(1..\text{index}) \setminus \langle p \rangle \land \\
& \quad \text{index}' = \text{index} + 1
\end{align*}
\]

(10)

\[
\begin{align*}
\text{Jump}_2(p : \text{Page}) & \triangleq \text{history}(\text{index}) \text{ links } p \land \\
& \quad p \in \text{ran history} \Rightarrow \\
& \quad \text{history}' = \text{history} \land \\
& \quad \text{history}(\text{index}') = p \land \\
& \quad p \notin \text{ran history} \Rightarrow \\
& \quad \text{index}' = |\text{history}'| \land \\
& \quad |\text{history}| = 8 \Rightarrow \text{history}' = \text{tl history} \setminus \langle p \rangle \land \\
& \quad |\text{history}| < 8 \Rightarrow \text{history}' = \text{tl history} \setminus \langle p \rangle
\end{align*}
\]

(11)

(Note that the \( \text{tl} \) operator sequence \( \text{tail} \); \( \text{tl} \langle i_1, i_2, i_3, \ldots \rangle = \langle i_2, i_3, \ldots \rangle \).)

To complete the device specification we need to describe valid initial states for the device. In this case we assert that there is only the home page in the history list and that the index points to it.

\[
\text{devInit} \triangleq \text{history} = \langle \text{home} \rangle \land \text{index} = 1
\]

(12)

So far, we have modelled a web as a directed graph and described the state space and actions of a device that can browse such a web. We can describe the behaviour allowed by the two devices as follows...

\[
\begin{align*}
\text{devBeh}_1 & \triangleq \text{devInit} \land \Box[\text{Bwd} \lor \text{Fwd} \lor \text{Jump}_1(p)]_{\text{dev}}
\end{align*}
\]

(13)

\[
\begin{align*}
\text{devBeh}_2 & \triangleq \text{devInit} \land \Box[\text{Bwd} \lor \text{Fwd} \lor \text{Jump}_2(p)]_{\text{dev}}
\end{align*}
\]

(14)
These two specifications read as follows (where $n$ is one or two): ‘devBeh$_n$ is the specification of a device which starts in a state such that devInit holds. Subsequently any state change is either the stuttering change (i.e. the state remains the same) or is a valid state change caused by Bwd, Fwd or Jump$_n$.’

3 The planned action mechanism

We now describe a mechanism showing how the user invokes the device actions described in the previous section. The mechanism is given a plan and will select ‘operations’ to satisfy that plan. Which operations are chosen are determined by the state of the device that is revealed to the user via the interface.

We assume that at any one time the interface reveals the following information to the user; the history list history, the current page current, whereabouts in the history list the current page is (the variable index) and the set of all links exposed on the current page currentLinks. history and index are the device variables. current and currentLinks are derived from them and the web.

$$\text{current} \triangleq \text{history}(\text{index})$$  \hspace{1cm} (15)
$$\text{currentLinks} \triangleq \{ p \mid \text{current links } p \}$$  \hspace{1cm} (16)

Hence the device interface reveals all the information about the device state, but does not reveal all the web at once.

The user can deal with the current page in two ways; they can look at it, which involves reading it or dealing in some way with its content, or they can glance at it. Glancing simply involves noting that one is on a particular page and noting which device action was invoked to get the user there. Glancing does not particularly involve digesting a page’s content. The user has a high level plan to look at a sequence of pages. In order to get to a page that needs to be looked at, the user may pass through intermediate pages which get glanced at.

For example if the current history list is $\langle A, B, C, D \rangle$, the current page is $A$ and the user wishes to look at $D$ she may $\text{Fwd}$ her way through the history list, glancing at $A$, $B$ and $C$ and then looking at $D$.

The user therefore maintains memories of two histories; a history of the pages she has looked at and a history of the pages she has glanced at.

$$\text{lookedAt} \triangleq \text{Page}^*$$  \hspace{1cm} (17)
$$\text{glancedAt} \triangleq (\text{Page} \times (\text{Bwd} \mid \text{Fwd} \mid \text{Jump}))^*$$  \hspace{1cm} (18)

This is the only memory requirement we place on the user; we do not assume that she remembers anything more about the web such as which pages link which and so on. The variable memory denotes this memory space.

$$\text{memory} \triangleq (\text{lookedAt}, \text{glancedAt})$$  \hspace{1cm} (19)

We can now define the actions of $\text{look}$ and $\text{glance}_{\text{how}}$ which act upon the memory space, updating the appropriate history. (The parameter $\text{how}$ denotes which device action was
used to arrive at the page that is glanced at. For example \( \text{glance}_{\text{Bwd}} \) is the action of glancing at the current page and noting that the device action \( \text{Bwd} \) was invoked to get there.)

\[
\text{look} = \text{lookedAt}' = \text{lookedAt} \sim \langle \text{current}' \rangle \land \\
\text{glancedAt}' = \text{glancedAt}
\]

\[
\text{glance}_{\text{how}} = \text{lookedAt}' = \text{lookedAt} \land \\
\text{glancedAt}' = \text{glancedAt} \sim \langle \langle \text{current}', \text{how} \rangle \rangle
\]

We model a plan as a sequence of pages that user must look at.

\[
\text{plan} : \text{Page}^*
\]

The user is said to have satisfied a plan when \( \text{lookedAt} = \text{plan} \).

We use a sequence (rather than a set) because we want to capture that the order pages are read may be important. For example, if the user is trying to find out what elephants eat for breakfast and comes across two pages, one stating that ‘Elephants are pachyderms’ and one stating that ‘Pachyderms eat buns for breakfast’, the order of the receipt of this information is crucial. The user may receive ‘Pachyderms eat buns for breakfast’ first and ignore it, not knowing that elephants are pachyderms and therefore not realising that this information is relevant.

Now we describe how the mechanism commits to certain operations. There are three ‘atomic’ operations which correspond to the three actions offered by the device: \( \text{doBwd} \), \( \text{doFwd} \) and \( \text{doJump}(p) \). Furthermore there are three ‘high-level’ operations that the mechanism must ‘resolve’ into atomic operations: \( \text{lookAt}(p) \), \( \text{find}(p) \) and \( \text{goto}(p) \). A commitment to \( \text{lookAt} \) a page is resolved into making that page the current page and looking at it. A commitment to \( \text{find} \) a page is resolved into actions that make that page the current page and \( \text{lookAt} \) it. A commitment to \( \text{goto} \) is committed to when the required page is on the history list and is resolved into the appropriate number of \( \text{Bwd} \) or \( \text{Fwd} \)s.

\[
\text{operation} = \text{find}(p : \text{Page}) | \text{doBwd} | \\
\text{goto}(p : \text{Page}) | \text{doFwd} | \\
\text{lookAt}(p : \text{Page}) | \text{doJump}(p : \text{Page})
\]

The user maintains a list of operations which have been committed to. This list acts as a stack; the first commitment in the list will be the first to be acted on and new commitments will be added to the front of the list.

\[
\text{committed} : \text{operation}^*
\]

The commitment stack and the memory describe the cognitive state of the user, denoted by \( \text{cog} \).

\[
\text{cog} = (\text{committed}, \text{memory})
\]

As an example of the commitment stack in action, imagine the user commits to \( \text{lookAt} \) some arbitrary page \( p \) and that \( p \) is not the current page. The mechanism will replace that commitment with a commitment to \( \text{find} \) page \( p \) and then \( \text{lookAt} \) it. (\( \text{find} \) just concerns getting to a page, \( \text{lookAt} \) involves looking at it.) If then the user realises that page \( p \) is
two steps up the history list then the commitment to find page \( p \) will be replaced with
the commitment to goto page \( p \), (the commitment to goto a page is made when the page in
question is in the history list). The goto commitment is then resolved to two doBwds
which are atomic and can be executed.

Starting with the simplest cases first, we define the actions that deal with executing
atomic commitments. It is simply a case of performing the appropriate device actions and
removing the commitment. When the user invokes a device action she will glance at the
resulting device state.

\[
\text{doBwd} \triangleq \text{committed} = \langle \text{doBwd} \rangle \sim C \land Bwd \land
\text{committed}' = C \land \text{glance}_{\text{Bwd}}
\]

\[
\text{doFwd} \triangleq \text{committed} = \langle \text{doFwd} \rangle \sim C \land Fwd \land
\text{committed}' = C \land \text{glance}_{\text{Fwd}}
\]

\[
\text{doJump}_n \triangleq \text{committed} = \langle \text{doJump}(p) \rangle \sim C \land \text{Jump}_n(p) \land
\text{committed}' = C \land \text{glance}_{\text{Jump}}
\]

(Note the syntactic conventions used here; doBwd is the action that executes the com-
mitment, whereas doBwd is a token that denotes the commitment itself. In each action
the first line is the action precondition, the second describes changes to \( dev \), the third
the changes to the commitment stack and the fourth the changes to the user’s memory.
Furthermore note that we have used the action \( \text{Jump}_n(p) \) to denote either \( \text{Jump}_1(p) \) or
\( \text{Jump}_2(p) \) — it is not an issue here which device model is referred to. We shall return to
this later.)

\[
\text{execute}_n \text{ denotes the disjunction of these three actions.}
\]

\[
\text{execute}_n \triangleq \text{doBwd} \lor \text{doFwd} \lor \text{doJump}_n(p)
\]

We now define how operations are resolved. If the commitment stack is empty and the
plan has not been satisfied the user will commit to lookAt the next page from the plan.

\[
\text{newCommit} \triangleq \text{committed} = \langle \rangle \land \text{lookedAt} \neq \text{plan} \land
\text{dev}' = \text{dev} \land
\text{committed}' = \langle \text{lookAt}(p) \rangle \cdot \text{plan} = \text{lookedAt} \sim \langle p, \ldots \rangle \land
\text{memory}' = \text{memory}
\]

To lookAt the current page the user looks at what is there and then removes that
commitment from the stack. If the current commitment is to lookAt a page that is not the
current page then the user commits to find that page and then lookAt it.

\[
\text{doLookAt} \triangleq \text{committed} = \langle \text{lookAt}(p) \rangle \sim C \land \text{current} = p \land
\text{dev}' = \text{dev} \land
\text{committed}' = C \land \text{look}
\]
There are four cases to be dealt with when the user is trying to find an arbitrary page \( p \). Firstly, if \( p \) is the current page then the commitment to find it is simply dropped. Secondly if \( p \) is linked to the current page then the user commits to \( \text{doJump} \) to that page. Thirdly if \( p \) is in the history list then the user commits to \( \text{goto} \) that page. The last case we look at is how the user attempts to explore the web for a new page (i.e. one not on the history list or linked to the current page). A precise answer to this is well beyond our scope so we simply say that the user plans to \( \text{lookAt} \) any page that is not on the history list in the hope that doing so may expose links to the required page.

Finally we deal with a commitment to \( \text{goto} \) a page. Again we start with the null case where the user has committed to \( \text{goto} \) the current page; the commitment is dropped. If the page the user committed to \( \text{goto} \) is above the current page in the history list then the user commits to the appropriate number of \( \text{doBwd} \)s and similarly if the page in question is below then the appropriate number of \( \text{goFwd} \)s are committed to. (In this formalism a number \( n \) superscripted to a sequence denotes that sequence repeated \( n \) times; \( (\text{doFwd})^3 = (\text{doFwd}, \text{doFwd}, \text{doFwd}) \).)

\[
\text{planLookAt} \triangleq \text{committed} = (\text{lookAt}(p)) \sim C \land \text{current} \neq p \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = (\text{find}(p), \text{lookAt}(p)) \sim C \land \\
\text{memory}' = \text{memory}
\]  
\[
\text{lookAt} \triangleq \text{doLookAt} \lor \text{planLookAt}
\]  

\[
\text{found} \triangleq \text{committed} = (\text{find}(p)) \sim C \land \text{current} = p \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = C \land \\
\text{memory}' = \text{memory}
\]  
\[
\text{foundLink} \triangleq \text{committed} = (\text{find}(p)) \sim C \land \text{current links} p \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = (\text{doJump}(p)) \sim C \land \\
\text{memory}' = \text{memory}
\]  
\[
\text{onList} \triangleq \text{committed} = (\text{find}(p)) \sim C \land p \in \text{ran history} \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = (\text{goto}(p)) \sim C \land \\
\text{memory}' = \text{memory}
\]  
\[
\text{explore} \triangleq \text{committed} = (\text{find}(p)) \sim C \land \\
p \notin \{\text{current}\} \cup \text{currentLinks} \cup \text{ran history} \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = (\text{lookAt}(q), \text{find}(p)) \sim C \land q \notin \text{ran history} \land \\
\text{memory}' = \text{memory}
\]  
\[
\text{find} \triangleq \text{found} \lor \text{foundLink} \lor \text{onList} \lor \text{explore}
\]  
\[
\text{noGo} \triangleq \text{committed} = (\text{goto}(p)) \sim C \land \text{current} = p \land \\
\text{dev}' = \text{dev} \land \\
\text{committed}' = C \land \\
\text{memory}' = \text{memory}
\]
\[ \text{goUp} \triangleq \text{committed} = \langle \text{goto}(p) \rangle \overset{\text{index}}{\leftarrow} C \land \text{history}(i) = p \land i < \text{index} \land \text{dev}' = \text{dev} \land \text{committed}' = \langle \text{doBwd} \rangle^{\text{index} - i} \overset{\text{C}}{\leftarrow} C \land \text{memory}' = \text{memory} \]

\[ \text{goDown} \triangleq \text{committed} = \langle \text{goto}(p) \rangle \overset{\text{index}}{\leftarrow} C \land \text{history}(i) = p \land i > \text{index} \land \text{dev}' = \text{dev} \land \text{committed}' = \langle \text{doFwd} \rangle^{i - \text{index}} \overset{\text{C}}{\leftarrow} C \land \text{memory}' = \text{memory} \]

\[ \text{goto} \triangleq \text{noGo} \lor \text{goUp} \lor \text{goDown} \]

4 Two system models

Now we disjoin the user actions together to define next-step action \( \mathcal{A}_n \) of the system (note that the device actions are contained within the \( \text{execute}_n \) action).

\[ \mathcal{A}_n \triangleq \text{execute}_n \lor \text{newCommit} \lor \text{lookAt} \lor \text{find} \lor \text{goto} \]

To give a full specification we need to assert an initial condition. We already know the initial device condition (that the device starts with just the home page on the history list). Furthermore we assert that the cognitive state starts with no pages in lookedAt, the home page in glancedAt and no operations in the commitment stack.

\[ \text{init} \triangleq \text{devInit} \land \text{committed} = \langle \rangle \land \text{lookedAt} = \langle \rangle \land \text{glancedAt} = \langle \text{home} \rangle \]

(Note that \( \text{plan} \) is a fixed variable and therefore we do not need to assert its initial value.)

The system specification \( \Phi_n \) is then the conjunction of the initial condition with the stuttering repetition of \( \mathcal{A}_n \) and the assertion that \( \mathcal{A}_n \) is weakly fair.

\[ \Phi_n \triangleq \text{init} \land \Box [\mathcal{A}_n]_{(\text{cog.dev})} \land \text{WF}_{(\text{cog.dev})}(\mathcal{A}_n) \]

\( \Phi_n \) is the specification of a system that starts in a state described by \( \text{init} \). Each subsequent state change is either a valid \( \mathcal{A}_n \) step or is a ‘stutter’, \( i.e. \) the state does not change. Weak fairness of the action \( \mathcal{A}_n \) means that it cannot be enabled indefinitely without eventually occurring. \( i.e. \) the user does not stop interacting until \( \mathcal{A}_n \) becomes disabled, hopefully by the satisfaction of the plan. (\( \mathcal{A}_n \) may become disabled for other less desirable reasons however, such as a commitment to \text{lookAt} \ a page for which there is no finite path of \text{links} to.)

Recall that we did not commit to which device model the specification refers to. This we now do. We define two specifications; \( \Phi_1 \) and \( \Phi_2 \) such that we simply syntactically replace \( n \) with 1 or 2 throughout the specification \( \Phi_n \). Hence \( \Phi_1 \) denotes the behaviour of an interactive system which includes the stack based history mechanism and \( \Phi_2 \) denotes the same interactive system except that the history mechanism is queue based.
5 Evaluating for usability

In this paper we consider the case of a user revisiting pages. Wanting to revisit a given page and not being able to do so is an obvious source of frustration to users. We can semi-formally require of a system the following...

\[ \text{see a page} \leadsto (\text{want to look at that page} \Rightarrow \text{easy to get to that page}) \]  

(The temporal operator \( \leadsto \) reads ‘leads to’.) In other words we want it to be easy to visit pages that have already been seen.

Now we argue that it is easy to get to a page that is on the history list, so we can rewrite the above requirement as...

\[ \text{see a page} \leadsto (\text{want to look at that page} \Rightarrow \text{that page is on the history list}) \]  

... which we can then fully formalise as follows...

\[ \text{usable} \triangleq p = \text{current} \leadsto \left( \text{committed} = \langle \text{lookAt}(p) \rangle C \Rightarrow p \in \text{ran history} \right) \]  

Neither of our systems fulfill this property. Both history mechanisms are ‘lossy’, so we can easily propose situations where pages are lost from the history list even though we may subsequently wish to revisit them. So we then look at specific situations for which the property holds.

For the stack based mechanism consider a situation where the user has Bwded up the history list then Jumps to a new page (described more fully in [BBD98]). The end of the history list is deleted and replaced with the new page. The user cannot easily revisit any of the pages deleted.

Now consider the queue based mechanism. Assume a user Jumps along a sequence of more than eight pages and then wants to revisit a page more than eight Jumps back. Those pages will have dropped off the end of the list and therefore the user cannot easily revisit them.

Hence neither specifications \( \Phi_1 \) or \( \Phi_2 \) satisfy the usability property. So instead we look at a smaller scale problem such that the user wishes to reread two recently looked at pages and then continue browsing.

The user has Jumped along a sequence of pages, \( p_1 \ldots p_8 \) then decided that two of them need rereading (\( p_4 \) then \( p_5 \)) before proceeding to \( p_9 \) which is linked to \( p_8 \).

She Bwd to \( p_4 \) and reads it. What will she do next? According to the model she has two choices; to Fwd to \( p_5 \) or to Jump to \( p_5 \). We know this because the user will commit to \( \text{lookAt}(p_5) \) which is then resolved to \( \text{find}(p_5) \) followed by \( \text{lookAt}(p_5) \). Now both onList and foundLink are enabled and (after a bit more resolution) the user may either Jump or Fwd to \( p_5 \).

If the browser uses the queue based mechanism then it does not matter which she chooses because neither of them affect the history list at all (recall that, for the queue, Jumping to a page already in the history list simply moves the index to that page). Therefore \( p_8 \) remains in the history list, so the user can easily Fwd to it and then get to \( p_9 \).

However in the case of the stack based mechanism if the user Jump to \( p_5 \) then \( p_6 \ldots p_8 \) are lost from the history list and the user will have to reconstruct the path to \( p_8 \) from
memory. This is not too much of a problem in this example, but once we generalise to a problem where the user would have to reconstruct a path of arbitrary length then the system would have serious usability problems.

So we have identified a possible problem based on the fact that in certain (we believe common) situations the user is offered a choice between Jumping to a page and Fwding to it. In the model here a non-deterministic choice is made between the two and there may be problems if the user selects Jump with a stack mechanism.

We may decide to add a precondition to the action mechanism to stop the user doing such things, constraining the user to Fwd down the history list if there are pages that might want revisiting further down the history list. We would then have a constraint in the planning mechanism that we argue to be implausible; it requires that user has prior knowledge of what pages they will want to read at any point in the interaction. We do not assume that the user is consciously aware of plan; indeed it would be surprising if she was.

However, looking closely at the planning mechanism, we can see that there have been some rather strong assumptions made. Consider the action onList (equation 36). It says that a goto is committed to if the user wishes to find a page p that she knows to be on the history list. It could be argued that a more realistic action would be to commit to goto a page if the user believes that the page is on the history list. Although we have modelled the interface as transparent, most browser implementations do not make the history list readily apparent to the user. So the user is forced to rely on their beliefs about the state of the history list rather than its actual state. Relying on the user’s beliefs is problematic if they are wrong; the user may commit to goto a page that is not on the history list. We have not modelled error recovery so behaviour will stop without the plan being satisfied.

Another usability property for our modelled system is that the plan is eventually satisfied.

\[ \text{satisfy} = \Diamond \text{plan} = \text{lookedAt} \]  

(48)

The user’s beliefs about the history list are a ‘conceptual model’ of it. We have not included a conceptual model explicitly in the planning mechanism but we can do so by using the glancedAt memory.

Now assume that the user has been Jumping along an arbitrary sequence of pages \( p_1 \ldots p_n \). glancedAt will have the value \( \langle \ldots, (p_1, \text{Jump}), \ldots, (p_n, \text{Jump}) \rangle \). Now the user decides to revisit \( p_i \) (such that \( 1 \leq i \leq n \)). With the stack based mechanism the user can commit to goto\( p_i \) and is assured of getting there, but with the queue based mechanism \( p_i \) must be no more than eight pages ago (i.e. \( n - i \leq 8 \)) for this strategy to work. So without counting how many jumps the user has done (as unreasonable requirement to make of the user) the user may commit to an impossible operation and the system will not fulfill the property satisfy.

So with the queue based mechanism, in this situation the user needs to remember not only that they have already visited a page and done nothing but Jumps since, they also need to remember how recently they visited the page. With the stack based mechanism the user does not need to remember how recent the page was.

By weakening the assumptions about the user in the planning mechanism we have again exposed possible usability problems. This time the queue based mechanism is evaluated
as less usable than the stack based mechanism.

6 The evaluation strategy expressed more generally

Now let us review exactly what we did in the previous section a little more generally. Firstly we proposed a simple usability property but showed that neither specifications satisfied this property. In order to expose a difference between the two specifications we therefore suggested a situational constraint (a situational constraint allows us to ask ‘in a given situation is the specification usable?’) and saw if the usability property was now satisfied.

Stating this more formally we have the two specifications $\Phi_1$ and $\Phi_2$ which we decompose into the device and user assumptions...

$\Phi_1 = device_1 \land user$

$\Phi_2 = device_2 \land user$

We proposed a usability property $usable$ and then showed that neither satisfied the property.

$device_1 \land user \not\Rightarrow usable$

$device_2 \land user \not\Rightarrow usable$

We then proposed a situation (such that the user wishes to revisit two recent pages, let us call this situation $S$) such that...

$device_1 \land user \land S \not\Rightarrow usable$

$device_2 \land user \land S \Rightarrow usable$

So we exposed an apparent difference in usability between the two systems under in the situation $S$. We therefore tried to show that the problem lies not with the device specification in $device_1$ but with the user assumptions captured in $user$. So we proposed a further constraint (such that the user only $Jumps$ in certain circumstances, let us call this constraint $C$) such that...

$device_1 \land (user \land C) \land S \Rightarrow usable$

We then argued that this further constraint was unreasonable. Therefore we suggested that the fault lies with the device and we registered a design preference for $\Phi_2$.

Then we argued that $user$ was possibly too constrained so we suggested a new planning mechanism $user'$ (such that the user commits to $goto$ a page when she $believes$ it to be on the history list rather than when she $knows$ so) and another property $satisfy$ which has usability relevance. We then showed that...

$device_1 \land user' \Rightarrow satisfy$

$device_2 \land user' \not\Rightarrow satisfy$

There is a lot of information here, but the crucial thing is that it is stated formally and is therefore explicit and inspectable. It is not the role of this paper to decide whether $user$ is more plausible than $user'$ or whether $S$ is a common situation, but it does expose these factors clearly for others to discuss.
7 Discussion and further work

We have formalised device and user models and shown how usability design decisions can be guided by the evaluation of these models. We have kept the models as simple as possible, given the analyses performed. We have only modelled how the user translates plans into actions and suggested that the user may have problems with the stack based web browser in certain situations.

Simplifying models as much as possible reduces the cost of the analysis but there is a corresponding decrease in how sure we can be that we have real-world validity for our results. However a formal approach ensures that everything we model is explicit and inspectable, so any qualifications that we make to the result of our evaluation can be exposed to rigorous argument.

The model presented here is similar in concept to the GOMS model [CMN83] which describes how high level actions are hierarchically structured. We have shown how such a user model can be integrated with a device model (but also shown where the separation between the two occurs) to give a more holistic description of an interactive system (see also the work on Interaction Framework [BHB95] and Syndetics [DBDMng]).

A benefit of a formal approach is that we have been able to use considerable non-determinacy in the user models when we did not need to commit to details. Consider the action \textit{explore}; it was included in the user model for completeness, but we make no reference to it in the usability evaluation; therefore the action describes very many possible strategies that the user may employ to explore new parts of the web. If we were to evaluate a browser for usability in situations where the user is exploring, we would need to commit to much more detail in this part of the model for our results to have any significance.

There are many ways that the model given here can be refined to look at other aspects of web-browser usability. To conclude we sketch out some of those refinements.

7.1 Conceptual models

Users often have incorrect conceptual models of the browser [CJ96] and confusion results (confusion being what the user suffers from when the device does not do what is expected of it). We began to look at how an implicit conceptual model could be captured using \textit{glancedAt}. Further work could involve including an explicit mental representation of the device state.

7.2 Interface representation

In the evaluation section we also began to consider what happens if the interface is not a complete projection of the device state and we suggested how we could weaken the planning mechanism to cope with this.

We effectively then have two planning mechanisms: one that commits to actions that the user knows will achieve a result and one that commits to actions that she believes will achieve a result.

This discussion then raises an interesting point about what we are actually capturing in the user model; is it what the user \textit{actually} does or what the analyst \textit{wants} the user to
do? We argue that the user’s reliance on possibly mistaken beliefs is caused by inadequate representation of the device state. In most implementations the user needs to pull down a menu in order to see the history list; it is not readily apparent.

The designer is likely to want to bias the user away from behaviour where she relies on her (possibly incorrect) beliefs and towards behaviour where she relies on (definitely correct) knowledge about the device. The mechanism for this biasing is the user interface [BC97]. The interface designer might therefore suggest that the history list is continually visible as a selectable list and that Bwd and Fwd become up and down arrows on that list. The history list and the effect of invoking Bwd, Fwd and Jump would then be immediately obvious.

7.3 Cognitive modelling

We have taken no account of user knowledge here; we have simply assumed that the satisfaction of the plan achieves some goal. Modelling how such plans are constructed would take us into the realm of cognitive models as used in Programmable User Modelling [YGS89, BY96].

Typically a goal will be of the form ‘I want to know information with the property X’. The user will then start to produce sub-goals which involve going to pages with relevant information on them. The fixed variable plan in our model captures the outcome of this process. A cognitive model will allow us to look at issues such as how the browser supports the formation of sub-goals, or whether the ‘anchor’ of a link has enough relevant information in it such that user knows that following this link will (or will not) get her closer to the achievement of her goal.

7.4 Conclusions

We have shown how simple user models allow usability issues to be considered in a design life-cycle. This evaluation technique is intended as a complement to (not a replacement for) existing user centred and empirical findings.

We intend to develop a corpus of these user models expressed at various levels of detail along with corresponding usability properties that can be sensibly evaluated with these models.

Acknowledgements

The authors are grateful for enlightening discussions with Richard Young and Jason Good. This work is funded by EPSRC grant GR/L00301. See the world wide web site at http://www.cs.mdx.ac.uk/puma/ for further details.
References


