Formal user models and methods for reasoning about interactive behaviour

Richard Butterworth    Ann Blandford
David Duke            Richard Young

Early summer 1998
WP17

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
fax: +44 (0)181 362 6411
email: a.blandford@mdx.ac.uk

Project funded by EPSRC grant number GR/L00391
Formal user models and methods for reasoning about interactive behaviour

Richard Butterworth  Ann Blandford  David Duke  Richard Young

Abstract

User models allow for usability decisions to be made about abstract system designs. Programmable User Models are one such approach to user modelling; they have been developed over the past decade so that they now cover a wide range of informal, semi-formal and fully formal approaches. In this paper we briefly review the approaches investigated, thus setting out a context for Programmable User Modelling and propose a formal model of the cognitive assumptions that underly the approach. In particular we contrast approaches taken in recent formal work performed by the PUMA project where a trade-off is made between clarity of cognitive assumptions and the level of abstraction of the models. This broad review allows us to discuss notions pertinent to the field of formal HCI in general, namely what role craft skill plays in formal HCI.

1 Introduction

An aim of a theoretical approach to human-computer interaction is to allow for usability criteria to be evaluated predictively. Evaluating a system for usability by user testing is problematic because the system must be built (or at least convincingly mocked-up) before it can be tested; many design decisions and commitments must be made before testing. Our theoretical approach aims to use cognitive science theory to approximately evaluate the potential usability of an abstract system model.

The PUMA project aims to codify cognitive science theory such that it can be integrated with abstract models of devices\(^1\). Techniques for expressing devices abstractly abound in the formal methods literature (e.g. [AI91, Spi89, Hal90] etc.). There further exists a considerable body of literature showing the integration of HCI and formal methods concerns (e.g. [HT90, HT97, RS96] etc.).

Much of the recent work on PUMA has concerned constructing system models that embody cognitive assumptions but which are abstract enough to be manipulated as part of a formal proof, or mathematical argument. However the act of abstracting the models to the point that they are simple enough to be rigorously evaluated can cause the cognitive assumptions to become very implicit within the model. This recent approach can be contrasted to previous approaches where the cognitive theory is codified formally and then imported wholesale into the system model. This approach resulted in models for which the cognitive assumptions are clear and explicit, but are too complex to be rigorously evaluated.

\(^1\)Note and take care with our terminology; a ‘device’ is the automated part of a system, the ‘system’ is the closed composition of all the agents we are modelling including the user population. A ‘usable device’ is a device that a given user population finds (in some way) easy to use. A ‘usable system’ is the composition of a device with a user population that finds the device usable.
This paper takes a general look at formal user modelling; it lays down the underlying cognitive theory and discusses how we have used this theory in building abstract system models. We also look at the strengths and weaknesses of this underlying theory, discuss the advantages and disadvantages of using this theory abstractly and discuss the prospects for this sort of theoretical work in HCI.

As this is a paper discussing a general approach to formal HCI we shall not attempt to demonstrate our approach by specific exemplars. Such exemplars can be found elsewhere [BBG97, BBD98b, BBD98a].

2 PUM and PUMA — a brief history

To place the work reported here in context, we need to give a brief summary of past work on developing cognitive models and integrating them with device specifications.

2.1 Early work with Soar models

Early work focused exclusively on the development of hand-crafted cognitive models, implemented using the Soar [New90] cognitive architecture. Once a model had been implemented, it could be run to establish what possible behaviours were predicted [YGS89]. Such hand-crafting was very time-consuming. More importantly, each model relied on the skill of the analyst and there were, inevitably, inconsistencies between the ways that different models were implemented. Once implemented, it was also very difficult for anyone other than the analyst to reconstruct, and acquire an abstract understanding of, the assumptions that had been made about the modelled user.

2.2 The Instruction Language

To reduce the element of craft skill in implementing models, a higher level specification language — the Instruction Language — was developed to allow the analyst to describe the knowledge encoded in the model in a way that is largely independent of the underlying (Soar) architecture. Using the Instruction Language (IL), the analyst can describe the knowledge the modelled user has about objects in the domain, relationships between those objects, and conceptual operations that can be performed to change the device state. The analyst should also provide a device specification in a compatible form. Once this knowledge has been specified, there are alternative possible courses of action:

- If the knowledge is completely and rigorously defined then it is possible to compile it, to produce the corresponding Soar productions, which can then be run to give behavioural predictions [BY93]. This compiler embodies the core assumptions about the modelled user's problem-solving capabilities, and thus provides a link between the hand-coded models and those described in the IL.

\footnote{The only compiler that has been implemented was created for a now obsolete version of Soar that did not support interaction between a modelled user and separately specified device. It is discussed here to clarify the link between hand-coded models and those specified via the Instruction Language.}
• Alternatively, the analyst who has an adequate understanding of the problem solving mechanism can use the IL description to reason rigorously, but informally, about the likely interactive behaviours (‘hand simulation’) [BY96].

2.3 Formal cognitive models

To make the assumptions about the problem-solving mechanism explicit and inspectable, we can express them formally. For example, Blandford, Butterworth and Good [BBG97] specify a problem solving mechanism embodying simple means-ends reasoning, and show how such a description can be used as a basis for rigorous, inspectable, reasoning about interactive behaviour. At this stage, the analyst still needs craft skill in writing IL descriptions, but the skill of cognitive modelling (using either Soar or hand simulation) is largely replaced by the requirement of skill in formal methods. In the generation of this description, we also abstract away from some of the alternative reasoning and learning mechanisms that are, in principle, made available through the Soar architecture or through hand simulation, so the models are becoming less rich but more precise.

2.4 Abstract system modelling

As an alternative approach, we have also used a simple specification of user and device behaviour as a basis for proof of properties of an interactive system [BBD98b, BBD98a]. In this work, we have aimed for minimalist models of the user behaviour in order to make proof tractable, with the result that the assumptions that are being made about user behaviour are explicit and inspectable, but the assumptions that are made about the underlying user cognition are not.

In this paper, we focus on the nature of the link between these last two approaches — that is, between the formal description of the assumptions underlying the cognitive architecture that can be used for reasoning about behaviour and the abstract specification of user and device behaviours that can be used as a basis for proof of properties of the interactive system.

3 A system model that incorporates cognitive assumptions

To formalise a general system model we express it in a formal notation: an initial question is which formal notation is most appropriate for this task?

A formal specification describes an abstract mechanism that generates behaviour, where a behaviour is a sequence of states (or equivalently a sequence of state changes). Our way of thinking of such a mechanism is as a set of legal initial states and a ‘next-state’ relation that describes legal steps from one state to another. A legal behaviour for this mechanism is one which starts in a legal initial state and each subsequent state change in that behaviour is legal according to the next-state relation. However this next-state relation only determines the ‘safety’ condition of the specification; it describes things that
legally can happen. Typically the mechanism will also include ‘liveness’ conditions which describe what will happen.

For example, consider a specification of an email system: the safety part of the specification may describe actions \( \text{sendTo}(M, u) \) and \( \text{receiveFrom}(M, u) \) (which formally describe sending a mail message \( M \) to user \( u \) and receiving a message \( M \) from user \( u \) respectively). Safety asserts that these two actions are all that can occur in the system; a behaviour that includes actions that are not described by \( \text{sendTo}(M, u) \) or \( \text{receiveFrom}(M, u) \) is illegal. However this is not a ‘complete’ specification at all. To complete the specification we would have to add liveness assertions that if \( \text{sendTo}(M, u) \) occurs then \( \text{receiveFrom}(M, u) \) must eventually occur (i.e. if a message is sent to \( u \) then \( u \) must eventually receive it).

Lamport’s TLA (Temporal Logic of Actions) [Lam94] conforms neatly to this view of a mechanism. Puritanically speaking, it should make no difference which particular specification notation we use so long as we can express our model clearly in it. We could employ a thoroughly reductionist approach to notation and express the model purely by set theoretic notation, but we would have to explicitly build in several issues (for example the ‘stuttering problem’) that are not too relevant here and are effectively hidden by notations such as TLA. Furthermore TLA has a well documented proof system and the purpose of constructing this general model is to determine specific system models that are amenable to proof.

### 3.1 The device specification

We denote the device state \( \text{dev} \). The device specification consists of a set of actions \( \mathcal{A} \) that describe legal transitions in the value of \( \text{dev} \).

\[
\mathcal{A} = \{A_1, A_2, \ldots, A_n\}
\]  

(1)

There is also a predicate \( \text{devInit} \) that describes legal initial states of the device. The device specification \( d\text{Spec} \) is expressed as follows...

\[
d\text{Spec} \equiv \text{devInit} \land \square \left[ \begin{array}{c} A_1 \lor \\ A_2 \lor \\ \vdots \\ A_n \end{array} \right]_{\text{dev}}
\]  

(2)

Glossed in English this asserts that the described device starts in a state described by \( \text{devInit} \) and then henceforth (the temporal operator \( \square \) denotes ‘henceforth’ or ‘always’) each change in state is described by any one of the actions in \( \mathcal{A} \). (The square brackets and the sub-scripted \( \text{dev} \) assert that the behaviour can ‘stutter’, the details of which are irrelevant here. Interested readers are directed to Lamport’s explanation of TLA [Lam94].)

This specification contains no liveness condition. This is because we view the device as being purely ‘driven’ by the user; it does not do anything unless ‘told’ to by the user. Without any user assumptions built into the model we cannot assert that the user will in fact tell the device to do anything and hence a legal behaviour for the device is to do nothing. This is a point to which we shall return later when we discuss the limiting assumptions of this model.
The crucial point to take from this device specification is that there is a collection of actions that are offered to the user.

### 3.2 Changes in belief state — user operations

When a user invokes a device action then she may correspondingly update her beliefs about the world. Furthermore the user may update her belief state by passively observing the state of the device. An ‘operation’ describes this update in the belief state together with the corresponding device action.

We denote the belief state $bel$. There is a set of operations $\mathcal{O}$ where each operation consists of a description of a legal change in belief state along with a device action (which may be the null device action).

$$\mathcal{O} = \{(O_1 \wedge a_1), (O_2 \wedge a_2), \ldots, (O_m \wedge a_m)\}$$  \hspace{1cm} (3)

$O_i$ denotes the update in belief state and $a_i$ is either one of the device actions from $\mathcal{A}$ or it describes no change in the device state.

$$\forall i : 1 \ldots m \bullet (a_i \in \mathcal{A}) \vee (a_i \neq dev = dev')$$ \hspace{1cm} (4)

$dev = dev'$ is the null device action; it asserts that the value of the variable $dev$ is the same after the action occurs as it was before\(^3\). There is also a predicate $belInit$ describing what are legal initial belief states.

The specification of a system that is correct according to these operations is as follows. ...

$$oSpec \triangleq \left( devInit \wedge belInit \right) \wedge \square \left[ \left( O_1 \wedge a_1 \right) \lor \left( O_2 \wedge a_2 \right) \lor \ldots \left( O_m \wedge a_m \right) \right] \wedge WF \left[ \left( O_1 \wedge a_1 \right) \wedge \left( O_2 \wedge a_2 \wedge \ldots \left( O_m \wedge a_m \right) \right) \right]$$ \hspace{1cm} (5)

This specification asserts that the system starts in a state such that the device state is legal according to $devInit$ and the belief state is legal according to $belInit$. Each subsequent state change is one of the operations from $\mathcal{O}$. The third conjunct asserts that the operations are ‘weakly fair’, in other words they cannot be indefinitely enabled without eventually occurring, or, the user will continue to interact until it becomes impossible to do anything.

Note that the set of operations $\mathcal{O}$ need not contain all the device actions from $\mathcal{A}$; the user need not know about all the actions offered by the device. Furthermore different operations may contain the same device actions. For example consider deleting marked text from a word processed document; there may be two operations ‘delete’ and ‘cut’ both of which invoke the device action ‘cutText’ and both of which update the user’s beliefs to reflect that the marked text is deleted from the document. The operation ‘cut’ would also describe the user’s belief that the deleted text is copied to the paste board.

$oSpec$ describes any behaviour trace where the device and belief states change according to the set of operations $\mathcal{O}$. It makes no assertions however about what order or when the

---

\(^3\)Note the notation for actions; a primed variable denotes that variable’s value after the action has completed, an unprimed variable denotes its value before the action.
operations occur, it effectively describes the set of behaviours such that operations occur in a random order. \textit{oSpec} is likely to be sufficient to evaluate interesting usability properties. For example one might wish to show that the user’s beliefs about the device state are always ‘correct’ (in some sense). If this can be shown to hold true of \textit{oSpec} then it may not be necessary to develop the model to give more detail. If random behaviours cannot get the beliefs about the device to be incorrect, then rational behaviours cannot either (rational behaviours must be a sub-set of random behaviours).

3.3 The cognitive architecture

The cognitive architecture attempts to limit the random behaviours specified by \textit{oSpec} to ‘rational’ behaviours. Our notion of rationality is based on Newell’s [New81] definition...

‘If an agent has knowledge that one of its operations\(^4\) will lead to one of its goals, then the agent will select that operation.’

This is problem solving rationality — it asserts that the user as an agent will always try to move from its current state towards some goal state.

We take a goal as being a belief state that the user wishes to attain. Note that the belief state will typically include the user’s beliefs about the device state, so, for example, a goal to get the device into a state such that a web browser displays a certain page is in fact a goal to get the user’s beliefs into a state such that the user believes the web browser displays that page.

We assert that the user employs a relationship over belief states, so that she can judge whether one belief state is closer to a goal than another. Given three belief states \(x\), \(y\) and \(Z\) we denote the fact that \(y\) is believed to be closer to \(Z\) than \(x\) as follows...

\[ x <_Z y \]

Using this relationship we describe two strategies the user employs to move her belief state closer to her goal state: reactive behaviour and planned behaviour. Reactive behaviour says ‘I am here at \(x\), I know I can get directly to \(y\) by using operation \(O\). \(y\) is closer to my goal than \(x\) so I shall perform \(O\).’ Planned behaviour says ‘My goal is to be at \(Z\). I know that if I was at \(x\) I could perform \(O\) and get to \(Z\). \(x\) if closer to where I am now than \(Z\). Therefore I shall make my sub-goal to get to \(x\) and plan to do \(O\) when I get there.’

Broadly speaking reactive behaviour moves the current state towards the goal, whereas planned behaviour moves the goal towards the current state. Furthermore the effect of planning is to widen the goal state and this will hopefully make reactive behaviour easier, because the user is ‘aiming’ at a larger collection of belief states that are closer to the current belief state than would be the case if no planning were done.

The user maintains a goal stack; initially there is one goal in the stack. The user interacts until there are no goals left on the stack. New sub-goals are pushed onto the top of the stack and once the belief state matches a sub-goal then it is popped from the stack. If the current belief state matches a sub-goal that is in the stack but not at the top of the

\(^4\)Newell uses the term ‘actions’ here; but the entities he refers to equate to operations in our terminology.
stack then that sub-goal and all the sub-goals above it on the stack are popped. This is to capture reactive behaviour that may move the belief state closer to the ultimate goal than the top of the goal stack. Effectively we allow sub-goals to be dropped even if they have not been specifically satisfied.

To formalise this architecture we must assert the following things...

1. the user interacts until the goal stack is empty, i.e. every action is preconditioned with the assertion that the goal stack is not empty,

2. the result of an operation must move the belief state towards a goal in the goal stack (reactive behaviour),

3. a transition must be added to describe how the architecture pushes sub-goals onto the goal-stack, and

4. a transition must be added to describe how satisfied sub-goals are popped from the goal stack (these last two factors describe planned behaviour).

The goal stack is represented by a sequence of belief states denoted $gs$, sub-goals are pushed onto and popped from the front of the sequence. We assert that there is a set of ‘rational operations’ $R$ that consists of all the operations from $O$ with the added preconditions corresponding to points 1 and 2 in the above list. Formally...

$$R = \{R_1, R_2, \ldots, R_m\}$$

$$R_i = gs \neq \langle \rangle \land \exists G \bullet (G \in \text{ran } gs) \land (\text{bel} < G \text{ bel'}) \land (O_i \land a_i) \land gs' = gs$$

$R$ consists of the same number of rational operations as there are operations in $O$. The first line of the definition of a rational operation $R_i$ asserts that for a rational operation to occur the goal stack is not empty. The second line asserts that the resulting belief state $bel'$ must be closer to a goal $G$ from the goal stack than the current belief state $bel$. (Note that a sequence is captured as a mapping from natural numbers to the sequence elements, hence the range of a sequence is the set of all the elements in that sequence.) The third line asserts that the update caused to the belief and device states by $R_i$ are the operation $(O_i \land a_i)$ from $O$. On the last line it is asserted that performing a rational operation has no effect on the goal stack; $gs = gs'$.

As well as these rational operations there are two further cognitive actions commit and drop which capture points 3 and 4 in the above list respectively.

$$\text{commit} = gs \neq \langle \rangle \land \exists \text{pre}, o \bullet o \in O \land \text{bel} <_{\text{top}} gs \text{ pre} \land (\text{pre})[o](\text{top } gs) \land \text{gs}' = (\text{pre})^{-} \text{gs} \land (\text{dev, bel}) = (\text{dev, bel})'$$

The first line of the definition asserts that commit can only occur if the goal stack is not empty. The second line asserts that there should exist a precondition belief state $pre$ and
an operation \( o \) such that \( o \) must be an operation from \( \mathcal{O} \) and \( \text{pre} \) must be closer to the current belief state than the top of the goal stack. The third line asserts that the operation \( o \) must take the belief state \( \text{pre} \) to the belief state on the top of the goal stack\(^5\). The forth line asserts that \( \text{pre} \) is pushed onto the top of the goal stack. The fifth line asserts that \( \text{commit} \) has no effect on the device and belief states.

\[
\text{drop} \triangleq \begin{array}{l}
gs \neq \emptyset \land \\
\exists X, Y \cdot \gs = X \setminus \{\text{bel}\} \setminus Y \land \\
\gs' = Y \land \text{bel} \notin \text{ran} X \land \\
(\text{dev}, \text{bel}) = (\text{dev}, \text{bel}')
\end{array}
\tag{9}
\]

The first line of the definition of \( \text{drop} \) asserts that it can only occur when the goal stack is not empty. The second line asserts that the current belief state should occur somewhere in the goal stack, \( i.e. \) there are two sequences of beliefs \( X \) and \( Y \) such that the goal stack is \( X \setminus \{\text{bel}\} \setminus Y \). The third line asserts that \( \text{bel} \) and all the sub-goals above it are popped from the stack. Furthermore if there are multiple occurrences of \( \text{bel} \) in the goal stack then the lowest occurrence of \( \text{bel} \) must be popped; \( \text{bel} \) should not remain anywhere in the goal stack. The fourth line asserts that \( \text{drop} \) does not effect the belief or device states.

Now we define a specification of a rational system \( r\text{Spec} \). The system state \( \text{sys} \) is denoted by the product of the three variables \( \text{dev}, \text{bel} \) and \( \gs \).

\[
\text{sys} \triangleq (\text{dev}, \text{bel}, \gs)
\tag{10}
\]

The initial state must be legal according to \( \text{devInit} \) and \( \text{belInit} \) and furthermore there must be exactly one goal in the goal stack. The next state relationship is the disjunction of all the rational operations and the two cognitive architecture actions \( \text{commit} \) and \( \text{drop} \). All the operations are asserted to be weakly fair.

\[
r\text{Spec} \triangleq \left( \text{devInit} \land \text{belInit} \land \exists g \cdot \gs = \{g\} \right) \land \Box \left[ \begin{array}{c}
R_1 \lor \\
R_2 \lor \\
\vdots \\
R_m \lor \\
\text{commit} \lor \\
\text{drop}
\end{array} \right]_{\text{sys}} \land WF \left[ \begin{array}{c}
R_1 \lor \\
R_2 \lor \\
\vdots \\
R_m \lor \\
\text{commit} \lor \\
\text{drop}
\end{array} \right]_{\text{sys}}
\tag{11}
\]

### 3.4 \( r\text{Spec} \) as a candidate description of a cognitive architecture

The system description we have developed here incorporates a candidate for a cognitive architecture. We make no assertions about it being in some way the best or it having more or less real world validity than any other cognitive architecture.

We have built in the idea of there being a cognitive belief state which is updated in tandem with the device state as device actions are invoked. These updates are captured

\(^5\)Using the semantic interpretation brackets \([\[]\] as part of the definition of an action probably twists TLA’s notation in a way it is not supposed to go. We use it here for clarity, but if our goal was a ‘strict’ TLA specification we would have to define \( \text{commit} \) in another (probably more complicated) way.
in the set of operations $O$. The closer to relationship $<$ captures an ordering over states describing the user’s ideas about which states are closer to each other. $O$ and $<$ are cognitively maintained variables.

We have not looked at how these variables can be changed during an interaction, for example a user may discover that the result of performing an operation was not what was expected. The user may subsequently change the belief update that occurs with that operation. In a similar way the user may discover that her ideas about which state is in fact closer to another are wrong and she may therefore wish to change the $<$ relationship. A simple answer to this apparent deficiency in the model is to assert that $O$ and $<$ are part of the belief state $bel$. Therefore we could define operations which change other operations and the $<$ relationship. Doing so adds another layer of complexity to the model but does not introduce anything mathematically intractable.

We do not assert that these cognitive variables have any particular concrete equivalent inside the user’s head; they are simply representations we use to try and capture the idea of rational behaviour and how such behaviour may be generated. The rationality we have defined is based around the idea of means-end analysis — a very simple problem solving technique. There are more powerful problem solving techniques documented, but remember our ultimate goal is to predictively evaluate interactive devices. A simple, weak problem solving strategy is likely to be sufficient for this task, a strong problem solver would ‘solve’ devices that are in fact difficult to use and therefore not give us a useful prediction of their usability [YGS89].

The device itself is modelled in a very simple way; it is wholly in the control of the user who simply ‘pushes’ it from state to state. Real devices may exhibit behaviour that is independent of what the user is doing. To capture these sort of devices we may wish to assert that there are ‘external’ and ‘internal’ device actions; the user can invoke the external actions but the internal actions may have a ‘life of their own.’ This would mean that $dSpec$ may include device liveness conditions.

There are other types of goal driven behaviour that are not well addressed by this model. We have captured ‘liveness’ goals i.e. the user interacts until eventually she fulfils her goals and then she stops interacting. There are also ‘safety’ goals which more concern the user attempting to stop the device from getting out of a goal state. Obviously this relates to a device model with independent behaviour. Games are a good example of this sort of goal driven behaviour. There is also exploratory behaviour that is not naturally represented as being goal driven.

4 Relating $rSpec$ to PUM-IL specifications

The behaviour of the specification $rSpec$ is determined by the set of actions $A$, the operations $O$ and the closer-to relationship $<$. Based on this we can sketch out a strategy for defining a semantics for PUM-IL (PUM Instruction Language) specifications. i.e. we could take a PUM-IL specification and translate it into $A$, $O$ and $<$ thus defining a specific specification based on $rSpec$. 
4.1 PUM-IL specifications

A PUM-IL specification consists principally of a collection of ‘operators’ of the following form...

- \textit{operator:Name} \triangleq \text{the name of the operator}
- \textit{purpose} \quad \text{the purpose to which the operator is put}
- \textit{precond} \quad \text{operator precondition}
- \textit{filter} \quad \text{the commitment filter}
- \textit{tracked} \quad \text{the update in beliefs caused by invoking the operator}
- \textit{device} \quad \text{the device action caused by the operator}

The purpose clause of an operator describes why the user commits to performing it, it does not necessarily describe the outcome of performing the action, but asserts the belief that performing this operation will reduce the difference between the current state and the goal state.

The precondition clause describes the condition that must hold before the operator can be invoked. The user may commit to an operator before its precondition is satisfied, if this is the case then the precondition is adopted as a new goal and the user will work towards this new goal in order that she can perform the committed-to operator.

The filter clause’s primary role is to describe under what circumstances an operator should be committed to. (Note the difference between the precondition and filter clauses — the precondition describes when an operator can be done, the filter describes when it can be committed to.)

The tracked clause describes how the user’s beliefs are updated once an operator is performed independent of the device display. For example the paste board is usually invisible to the user, so to interact correctly the user needs to track its contents.

The device clause describes the device action that is associated with the operator.

4.2 A sketch of PUM-IL semantics

These PUM operators are similar to, but not isomorphic with, the operations in the set $\mathcal{O}$. If an operation is denoted $\text{bel'} = f(\text{bel}) \land \text{dev'} = g(\text{dev})$ then the device action $\text{dev'} = g(\text{dev})$ relates directly to the device clause of an operator. The tracking clause relates to the resulting belief state $\text{bel'}$ of the operation and the precondition clause relates to the initial belief state $\text{bel}$ of the operation.

The purpose and filtering clauses are more subtle and relate more to the closer-to relationship $\circ$. For example consider an operator $\text{OPER}$. Its filtering condition is satisfied by some belief state $x$ and the result of doing it is tracked to be some belief state $y$. The purpose clause asserts that there is some goal $G$ which $\text{OPER}$ moves the system towards. Hence we can assert that $y$ is closer to $G$ than $x$: $x <_G y$.

Furthermore the filter clause can be used to prevent the user committing to an operator the outcome of which may be a goal, but the precondition of which is further away from the goal than the current state, so the filter clause contains further information which relates to the closer-to relationship. By analysing these purpose and filtering conditions we can build up a definition for $\circ$. 
<table>
<thead>
<tr>
<th></th>
<th>Simulation models</th>
<th>Proof models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level of abstraction</strong></td>
<td>Fixed to the level of abstraction that the original cognitive theory is expressed at. Furthermore to get a model that can be simulated automatically many decisions must be built in that reduce non-determinism, this will further reduce the level of abstraction at which the model can be expressed.</td>
<td>Can be considerably abstracted in both the device model and user assumptions.</td>
</tr>
<tr>
<td><strong>Reusability of models</strong></td>
<td>The cognitive theory incorporated in a simulation model can be easily ported from one example to another.</td>
<td>Cognitive assumptions are pared to a minimum, not only for a particular example but also for a particular question being addressed about that example. Hence the cognitive assumptions retained in a proof model may be inappropriate for other applications of the model.</td>
</tr>
<tr>
<td><strong>Method of evaluation</strong></td>
<td>The model can be simulated automatically or by hand by using the model to argue that a particular behaviour trace is legal or not with respect to the model. Alternatively the model can be used to generate behaviour traces. Simulation models can be used to argue the existence of legal behaviour.</td>
<td>An proof model can be used to prove that general properties of behaviour are consistent with the model, in particular we can use proof to show universal properties of behaviour. (A proof model can also be used to the existence of behaviours as a simulation model can.)</td>
</tr>
<tr>
<td><strong>Link to cognitive theory</strong></td>
<td>The link to cognitive theory is clear — there is (approximately) a one-to-one translation between the underlying cognitive theory and the formalised user model.</td>
<td>The act of abstracting the cognitive assumptions and the act of integrating these assumptions with the device model mean that the assumptions made become very implicit in the model.</td>
</tr>
</tbody>
</table>

Figure 1: A comparison between simulation and proof models

Although one can imagine a full semantics being defined for PUM-IL specification it would be of limited value to do so. One would have to take an existing PUM-IL specification, translate it via these semantics into an rSpec-like system specification and then use that specification to reason about usability properties. The effort in doing so would almost certainly outweigh any of the insights gained. However the process of trying to sketch out a semantics for PUM-IL has forced us to disambiguate certain issues within PUM specifications.

5 Approaches to formal user modelling

The PUMA project has been looking at ways of bringing the sort of user assumptions captured in rSpec into formal system descriptions that can then be reasoned about.

We can put the models we have produced into two categories which we shall call ‘simulation models’ and ‘proof models’. A summary comparison between simulation and proof models is shown in figure 1.
5.1 Simulation models

Simulation models are presented in [BBG97, BGY98]. The strategy for producing a simulation model is to start by constructing a model of a cognitive architecture and a device model. These two models are then married, usually by constructing a model of the user’s beliefs about the device as the link between the two. What results is a model for which the cognitive assumptions are explicit and clear. The derivation of the cognitive assumptions is simple and straightforward, typically it is a one-to-one translation of accepted cognitive theory.

If one were to simply take a specification of the form $r\text{Spec}$ and complete it by ‘filling in’ the operations and the relationship $<$ then that would generate a simulation model.

Such a model would be complex, detailed and operational in nature. It would have to be evaluated by simulation; the analyst would use the model to argue whether or not certain behaviour traces are legal according to the model. This is of course fine for existence proofs of certain types of behaviour. Typically we are interested in whether some goal state is reachable according to the model. If we can describe a behaviour trace that reaches the goal state and show that it is legal according to the model then we have shown reachability for that goal. In a similar way we can show the existence of erroneous behaviour that is legal for the model.

These simulation evaluations do not generalise however, they are, in essence, existence proofs; they demonstrate the existence of given behaviours. It is difficult (if not impossible) to argue about properties involving universal quantification over behaviours, such that the goal state will always be reached or that there is no erroneous behaviour.

5.2 Proof models

In order to overcome the problems encountered by the specific nature of the evaluation of simulation models we have also presented much simpler proof models which are pared to the bare minimum. Our strategy for producing proof models is to start with a device model expressed in terms of the actions that the device can perform. The user model forms a constraint over this model by describing under what conditions the user will invoke the device actions. These rules about when an action should be invoked are derived from the sort of cognitive theory expressed in $r\text{Spec}$ but are considerably abstracted. There is therefore not the clear link from cognitive theory to interactive system model that there is in simulation modelling.

The simplicity of a proof model means that it is tractable to proof in its evaluation; we can discuss general behavioural properties. However each particular proof model is bound to a particular question, for example the models presented in [BBD98b, BBD98a] both address the problems that users encounter when using the history list mechanism of a web browser. It is not clear that we could use these models to comfortably discuss other issues such as exploratory behaviour. In order to do so we would most likely have to reconstitute much of the model.

A useful analogy for considering proof models is that of a jelly mould. The cognitive theory forms the mould which constrains the resulting shape of the interactive system model, however, as with jellies, it may not be possible to reconstruct the shape of the
original mould from the resulting jelly. Furthermore to extend the analogy a little reuse of proof models is problematic — once a jelly is set it is messy to change its shape, it is easier to start again and set a new jelly.

However the abstract nature of the proof models we have developed allows for much stronger ‘universal’ questions to be asked of the models. Simulation models can only probe questions existentially. This is not to claim that proof models can prove that a system is ‘universally’ usable. However proof models can at least be framed to exhaust a specific class of issue. Knowing what issues to inspect then becomes a key question for the HCI people involved in developing a technology.

6 Discussion — craft skill in formal HCI?

The rationale behind formal approaches to HCI is to reduce the craft skill inherent in most HCI work.

An idealistic formal HCI approach would propose abstract models of devices, integrate them with formally expressed user assumptions and evaluate them against usability properties. Such a model can then be refined towards implementation by adding designed detail to the device model and more concrete user assumptions. Each refinement step can be proved to be a mathematically correct refinement. The resulting implementation can then be shown to inherit the usability properties of the abstract models.

This idealised approach relies on there being an accepted body of user assumptions and properties that can be argued to have usability relevance. There exists no such body of work in HCI. Formal HCI has therefore played a role in laying out assumptions that are made and attempting to generalise them. The push towards abstraction encouraged by formal modelling allows the analyst to stand back from much of the confusing detail that can abound in interactive systems. It is often claimed that the main benefit of formal HCI approaches is that they force the analyst to think about the user and device in a more general and clear headed manner. The actual models they produce do not finish up being used directly in the sort of formal system synthesis process outlined above.

So, practically, formal approaches to HCI have shown that they can allow analysts to think more clearly about the interactive system they are dealing with, but they have not removed the craft skill. There is still considerable craft skill involved in constructing and evaluating a formal model of an interactive system. However formal approaches force the craft skill in constructing interactive systems to become explicit. A cynic may suggest, however, that formal HCI merely moves craft skill from constructing actual interactive systems to constructing interactive system models.

Claims that formal modelling reduces craft skill may be misguided; creating a model is by definition a creative process. (See Dix’s arguments about the formality gap [Dix91, Chapter 11].) Formality allows a reduction in the craft skill involved not in creating models, but in translating models from one form to another. There is craft skill in design; even in a fully formal development process the designer is offered choices in which design step to make next. Formality allows the designer to show that the design step chosen is in some sense correct (typically a correct refinement), but does not make the decision that a particular design step is more correct than other correct steps. That decision is left to the
craft skill of the designer. Formal HCI allows for usability issues to be brought into the question of what is a ‘correct’ next design step but still leaves the ultimate decision to the designer.

This paper has laid down the assumptions we have used in constructing abstract models of interactive systems. The translation from \( r\text{Spec} \) to an actual proof model is a craft. What we actually do is abstract on \( r\text{Spec} \) and it is not infeasible that we could propose an ‘abstraction calculus’ which takes \( r\text{Spec} \) to a specific proof model so that we could mathematically show that a specific proof model is indeed an abstraction of \( r\text{Spec} \). However there is still craft skill involved here; a process of abstraction allows decisions to be made just as a process of refinement does. If we simplistically think of abstraction as the act of throwing away detail then there is nothing to stop us throwing away all the cognitive assumptions. The craft skill lies in judging how much or how little abstraction needs to be done to get a model that can be reasoned about rigorously and still embodies cognitive assumptions.

Acknowledgements

Thanks to Jason Good who soothed the main author’s brow during the email discussion that generated many of the points in this paper. 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


[BY93] A. Blandford and R. M. Young. Developing runnable user models: Separating the problem solving techniques from the domain knowledge. In J. Alty,


