A User Model for Avoiding Design Induced Errors in Soft-Key Interactive Systems

Paul Curzon and Ann Blandford

Interaction Design Centre, Middlesex University, London, UK
{p.curzon,a.blandford}@mdx.ac.uk

Abstract. Hard-key user interfaces are ones where the interface does not change during an interaction. Soft-key interfaces, on the other hand, change the meaning of inputs such as buttons as the interaction progresses. We demonstrate how interactive systems with either kinds of interface can be verified in HOL by combining a generic user model and a device specification incorporating a specification of the interface. In particular we show how design problems which result in users systematically making errors can be detected. We have extended the user model from previous versions to detect new errors related to soft-key interfaces.

1 Introduction

Interactive systems are computer systems where a human operator interacts with a computer through an interface to achieve a goal of the operator. A simple form of interface is a hard-key interface where the inputs have an unchanging effect throughout the interaction though they may sometimes be inactive. An example is the traditional style of vending machine with buttons for making specific selections. Those buttons have permanent labels. The meaning of the button is the same throughout the interaction. Our previous work in the area of the formal verification of interactive systems has concerned hard-key interfaces [8, 9].

Only very simple interactive devices use hard-key interfaces, however. More complex interactive systems use soft-key interfaces. With such an interface, the meaning of inputs (eg buttons) and outputs change during the interaction. Indeed, if implemented in software, the inputs and outputs may only exist for portions of the interaction. A simple example of such an interface is used in many cash machines which have a series of buttons round a screen. Labels on the screen indicate what each button does at any particular time during the interaction. For example, at one point the buttons are used to indicate the service the user requires, whilst later in the interaction the same buttons are used to indicate the amount of money desired. A GUI-based PC is a complex form of soft-key system. A "Button" is now any clickable button, drop-down menu etc that appears and disappears during the interaction.

When using particular interactive systems, HCI studies have shown that users make systematic errors that occur due to the user behaving perfectly rationally. As a result they occur persistently. By modifying designs to take rational user

behaviour into account such systematic errors can be eradicated. For example, early cash machines contained a flaw that led to users taking money but leaving their cards. With more recent designs this can not happen as the cash is not released until the card is taken. Even if all user errors can not be eliminated, the removal of such systematic ones would have a major effect on the usability of the system. Consequently, verification methodologies ought to detect design flaws that allow such errors to occur and to be useful must be able to deal with soft-key interfaces. We present one such methodology.

There are, two main approaches to formal reasoning about system usability. The first focuses on a formal specification of the user interface; Campos and Harrison [7] review several techniques that take this approach. Most commonly it is used with model checking based verification. Depending on the system, questions can be asked about whether a given event can occur or whether properties hold of all states. A non-model checking approach is that of Bumbulis *et al* [3] who verified properties of interfaces based on a guarded command language embedded in HOL. Back *et al* [1] use a general framework based on contracts. They illustrate how properties can be proved and data refinement performed of a specification of an interactive system. However, in general, such techniques do not directly support reasoning about the design errors that lead to users making systematic errors. Usability properties that are checked are necessarily device specific and are reformulated for each system verified.

The alternative, formal user modelling, involves generating a formal specification of the user and one of the computer system, in order to support reasoning about their conjoint behaviour. A formal specification of the user, unlike one of a computer system, is not a description of the way a user should be, since a user cannot be designed in the way that computer systems can. Rather it is a specification of the way they are [4]. Good interactive system design takes account of the user's limitations. Examples of formal user modelling include the work of Duke *et al* [10], Moher and Dirda [12], Paterno' and Mezzanotte [14] and Butterworth *et al* [5]. However, of these only the latter focuses on reasoning about user errors. They use TLA to reason about behaviour traces and reachability conditions within an interaction. A disadvantage is that each model has to be individually hand-crafted to each new device.

Here we combine the approaches. Primarily we work with a formal user specification based on established results from cognitive science [13]. In addition we explicitly model the user interface. In particular we model aspects of the way information about the effects of actions is presented to the user. We then use the user model with the device specification to prove properties of the combined system. The user is described in terms of the things they wish to achieve, the actions they may perform in order to achieve those goals, their device-independent knowledge about the task, and the knowledge about the effect of actions that they can obtain from the device interface. As we work in a common framework to that used for traditional system verification, the two activities are unified.

In our previous work [8] [9] we described a formal model of user behaviour written in higher-order logic. A novel feature of the model is that it is generic.

Rational user behaviour is specified once as a higher-order relation and just instantiated with task and device information when verifying a specific interactive system. We demonstrated how this user model could be used to detect the presence or absence of a range of design errors in interactive devices that HCI studies have shown lead to users making systematic errors. In particular we demonstrated that the approach could be used to detect the possibility of post-completion errors (where the user terminates the interaction too soon), classes of order errors with a cognitive cause, and certain delay errors. However, we implicitly assumed that the devices considered had hard-key interfaces. The nature of the interface was not modelled directly.

Here we extend the work in a new direction. Rather than just studying a range of errors, we investigate more complex styles of interface. In particular, we consider its use with soft-key interfaces. We show how the hard or soft-key nature of the interface can be modelled explicitly in a way that allows us to verify either kind of system. We also show that our methodology can detect the presence of further classes of design errors that arise as a result of soft-key interfaces. Changes have been made to the original version of the user model and the way it is instantiated to make it more accurately model rational human behaviour. In particular, we specify that the user only takes an action if there is some reason to believe that that action will have a particular effect[1]. An interface where this can occur is liable to suffer from persistent usability problems. We also introduce a "trigger" point to actions after which the user is unable to physically stop themselves taking the action committed to. We have used the HOL theorem prover [11]: all verification work has been fully machine-checked. All definitions and theorems are written in higher-order logic.

2 Case Study

To illustrate our methodology we consider two similar vending machine designs and prove they are free from the errors we consider. We outline the designs here, looking at how a user of the systems would be formally described in the subsequent sections. The first system has a hard-key interface and the second a soft-key interface. We use simple examples here to demonstrate the approach. However, any interactive system that could be described in terms of a relational specification and for which specific user actions and goals can be formulated could in theory be treated in a similar way. Whilst operator error with a vending machine is merely irritating, in a safety-critical interactive system such as an Air Traffic Control System it could be life-threatening.

The vending machines we consider require users to insert a coin, make a selection of chocolate (we simplify this to a single choice/button), and take change. Processing time of one cycle is needed after the selection is made. Despite its simplicity, without careful design, such a machine has the potential for users making a range of errors. Users could, for example, systematically forget to take

[1] We are not concerned with situations such as a user exploring the interface.

their change, repeat actions or insert coins and make selections in the "wrong" order – all for rational reasons. Indeed, vending machines with such design problems are widespread. The important point however is that vending machines without these problems also exist.

Here we consider two designs that avoid the specific problems addressed: one with a hard-key interface and the other an identical machine but with a soft-key interface. Our designs accept coin and selection in any order. Once both have been completed they release the change. A message flashing by the change slot indicates this. However this only occurs after a short delay. A "wait" message indicates to the user when the machine is busy. Importantly, this processing is scheduled after the coin has been inserted and the selection made. A sensor on the change slot flap releases chocolate only after the change is taken.

3 Modelling Hard and Soft-Key Interfaces Explicitly

When a user enters an interaction they will try to take actions that will help them achieve their goal. However, they will only take an action if they see an opportunity to do so. For the user to be able to rationally take an action the device needs to present appropriate information about the nature of inputs at the appropriate time. The user acts on the information presented. In our early work, we made no attempt to model the way that users know what an input does. We assumed implicitly that inputs were labelled so that the user was aware of their effect. In essence we assumed the buttons were permanently and clearly labelled with their meaning. It is relatively simple to introduce into our modelling approach a limited form of information about the interface itself.

Labels (even permanent ones) are in effect an output from the machine. They can therefore be modelled as boolean signals (history functions from time to a boolean value) as with other outputs following the traditional relational hardware specification approach. At time instances when the message is visible the history function is true and when the message is not visible the function is false. Thus the output for a permanently printed label indicating the use of a given hard-key would be true at all instances. A soft-key on the other hand would have a history function whose value depended on the time: sometimes true and sometimes false. A history function that was everywhere false would represent an unlabelled button. Whilst an expert might be able to use a device with unlabelled buttons even they would be prone to making errors concerning them. As we will see, by modelling the labels related to inputs in this way we can use them in the user model to restrict when a user might rationally take such an action. We currently use a single signal to combine two soft-key effects. The function is true only at times when both a virtual button exists and is labelled. To more accurately model interfaces these two functions could be split.

To model both soft and hard-key signals within our verification approach, we thus associate with each (virtual) input of the device an additional label output indicating its use (even if no such label exists). The device specification specifies the values of these label signals in the same way as for other outputs. Such a label

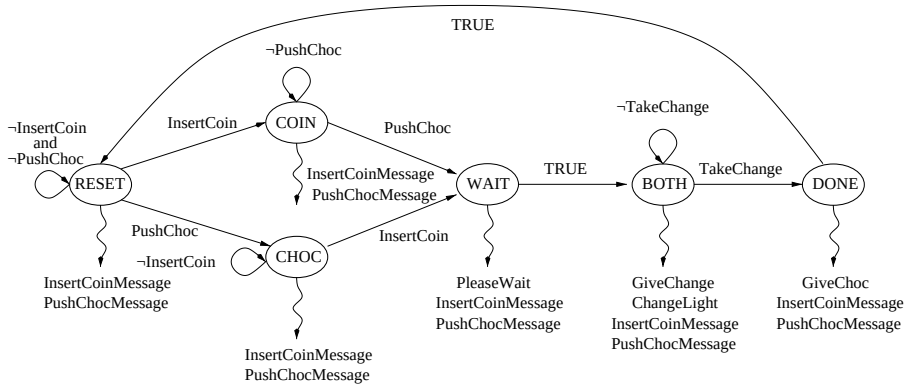

Fig. 1. Specification of a Vending Machine with a Hard-Key Interface

does not necessarily need to be in the form of text. It could be a picture or icon, or it could be the artifact itself. For example, normally it is clear what a coin slot is for without a label. Its very shape advertises its purpose. Deciding whether a given artifact, text or button does in fact advertise a given message is beyond the scope of our formal verification. We assume that a HCI expert would analyze the system to determine what messages are communicated – as well as determine other information required in the user model – by analyzing the specific task. Our methodology analyses the effectiveness of the design, given the results of such an informal analysis. As with any formal approach the verification results are only as good as their assumptions. An advantage of the approach is that it makes explicit information that the HCI expert must collect about the task and device. This would provide an early opportunity to detect usability errors. The verification would then ensure errors had not been missed.

The hard-key version of our vending machine design is essentially the same as that considered in [9]. The difference here is that we explicitly describe the interface. In particular we specify when the coin slot and selection button communicate their use. As it is a hard-key interface we assume both have labels indicating their use that are permanently visible. There are outputs from the device permanently presenting this information as can be seen in Fig. 1.

Our second machine has the same design as the first except for its interface. Rather than having permanent buttons, it has a touch screen interface. Rather than having permanent labels to indicate actions, text messages appear on the screen. This interface is modelled by setting the output signals to false in states where those buttons/messages are not presented. The coin slot is covered and so not visible in periods when the machine will not accept coins. The finite state machine specification is given in Fig. 2. In this informal diagram, outputs are indicated to be false by omitting them. In the formal specification, they are explicitly set to either true or false for each state. In particular, the coin slot exists and is labelled in the RESET and CHOC states. Similarly the chocolate button exists and is labelled in the RESET and COIN states.

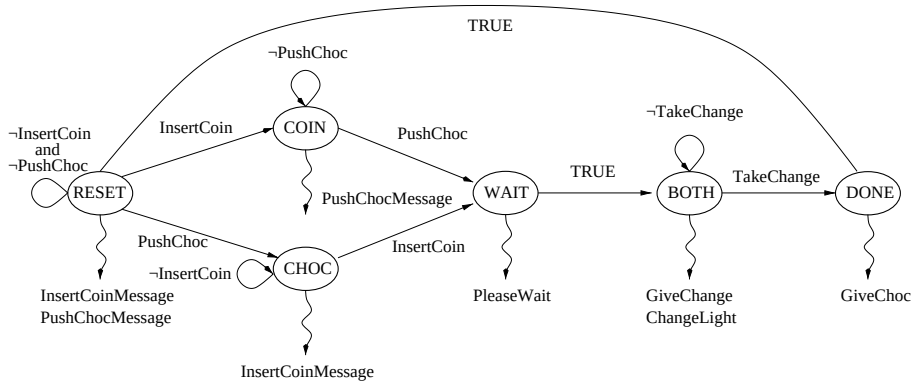

Fig. 2. Specification of a Vending Machine with a Soft-Key Interface

In the verification described here we formalized the user interface as part of the device finite state machine. It could just as easily be specified as a separate module. An identical device specification could then have been used for both designs, with each having separate, distinct interface specifications. Our verification methodology would work equally well with either approach.

4 A Generic User Model and Task Completion Theorem

In our approach, we not only model the device and its interface, but also the human component of the interactive system. The user model does not describe what a user *does* do, just what a user *could* do rationally. Our model makes no attempt to describe the likelihood of particular actions: a systematic user error is either possible with a design or not. Since we consider classes of error that can, by appropriate design, be completely eliminated, this strict requirement is in our view appropriate. The user model presented here is an extended version of the prototype described in [8]. In this section we describe its main features[2].

The user model is described by a relation on inputs (observations) and outputs (actions) just like any component of a system in the relational specification approach. The relation is true if the input and output history functions correspond to rational user behaviour. It is generic, taking a series of arguments corresponding to the details relevant to a specific machine. Some of the information supplied is task dependent and some device-dependent. Instantiating the user model involves specifying types for the user and machine state and providing concrete values for the arguments to the user model. By providing these specific details as arguments to the relation, a user model for the specific interaction under investigation is obtained. The important point is that the underlying cognitive model does not have to be provided each time, just lists of relevant

[2] A full version can be found at www.cs.mdx.ac.uk/staffpages/PaulCurzon

actions, observations, etc. The way those actions are acted on by a user is modelled only once. To illustrate the use of the generic model, we illustrate each part by reference to the arguments that must be supplied for that part of the model. An identical instantiated user model can be used for both examples we consider. In the remainder of this section, we first describe the temporal operator that underpins our user model. We then consider the formalization of different kinds of behaviour: reactive behaviour, behaviour resulting from a user's knowledge of the task, mental triggers and termination behaviour.

4.1 The Next Action

Our user model is based on a series of non-deterministic (disjunctive) temporally guarded action rules. Each describes an action that a user *could* rationally make. The rules are grouped corresponding to the user performing actions for specific related reasons. Each such group then has a single generic description. Each rule has a very similar form consisting of a guard and an action. They state that if the guard holds at some point then the NEXT action taken by the user is that given. By *next* in this context we do not mean the action taken on the next cycle, but rather the first action taken by the user after the current point in time. The action could be taken at any time in the future provided no other action is taken in the meantime. The rules each have the form:

```
guard t ∧ NEXT flag user_actions action t
```

To define NEXT we first need some subsidiary definitions. A signal is STABLE if it has the same value throughout a given period and a list of signals are stable (LSTABLE) if all have the same, given, value over the interval:

```
STABLE P t1 t2 v = ∀t. t1 ≤ t ∧ t < t2 ⊃ (P t = v)

(LSTABLE [] t1 t2 v = T) ∧
(LSTABLE (CONS a l) t1 t2 v = STABLE a t1 t2 v ∧ LSTABLE l t1 t2 v)
```

LF states that all but one of the signals in a list are false at a given time, with the odd one indicated by its position pos in the list.

```
(LF n [] pos t = T) ∧
(LF n (CONS a l) pos t = (((n = pos) ∨ ¬(a t)) ∧ LF (SUC n) l pos t))
```

NEXT can now be defined. It takes as arguments a flag signal that will be false throughout the period under consideration. We will return to its use later. A second argument is a list of all the possible user actions that could be taken. The definition is stating that all but one of these are *not* taken. The argument, a gives the position in the list of the action that will be taken. The final argument gives the current time. We are interested in the next action from this time.

```
NEXT flag al a t1 =
  ∃t2. t1 ≤ t2 ∧ LSTABLE al t1 t2 F ∧ (LF 0 al a t2) ∧ (EL a al) t2 ∧
       (flag (t2+1)) ∧ STABLE flag (t1+1) (t2+1) F
```

The definition states that there exists some future time t_2 when the action will be taken. All the signals will be false until that time. At that time all will continue to be false apart from the action identified by its position in the action list (EL accesses a given position (a) in a list (al) returning the entry there – here an action signal). The definition states that that signal will be true (ie, the action taken) at time t_2. The final two clauses assert that the flag will be false until the point just after the action is taken when it will be true. This definition forms the core of the user model. In the subsequent subsections we will look at its use to specify different aspects of rational behaviour.

NEXT is given a list of all user actions to specify which actions are not being taken next. This list is device dependent. For our devices the possible actions are inserting a coin; pushing a chocolate selection button; finishing the interaction; taking change; actively pausing and mentally triggering inserting a coin; triggering making a selection and triggering taking change. A corresponding list is supplied as an argument to the generic user model relation:

```
[InsertCoin ms; PushChoc ms; UserFinished us; TakeChange ms; Pause ms;
  TriggerCoin us; TriggerChoc us; TriggerChange us]
```

Some of these signals are part of the user state and others are part of the machine state and so an accessor function applied to the appropriate state is used (us for the user state and ms for the machine state). In the user model the type of the user state is a polymorphic type variable so we are also implicitly instantiating this type when providing arguments to the user model relation.

4.2 Reactive behaviour

The simplest group of non-deterministic rules describe *reactive* behaviour, where a user reacts to an external stimulus such as a light that clearly indicates that a particular action should be taken. For example, if a light comes on next to a button, a user might, if the light is noticed, react by pressing the button. All the rules have the basic form below.

```
REACT flag al stimulus action t = (stimulus t ∧ NEXT flag al action t)
```

If, at time t, stimulus is active, the NEXT action taken by the user out of the possible actions, al, at an unspecified later time, may be action. As there may be a range of signals designed to be reactive, the user model is supplied with a list of stimulus-action pairs: $[(s_1, a_1); \ldots (s_n, a_n)]$ A recursive relation, given a list of such pairs, extracts the components and asserts the above rule about them. They are combined using disjunction, so are non-deterministic choices. Guard and Action extract the components of the pairs.

```
(REACTIVE flag as [] t = F) ∧
(REACTIVE flag as (CONS s stimuli) t =
  (REACTIVE flag as stimuli t) ∨ (REACT flag as (Guard s) (Action s) t))
```

The action in the reactive rule will generally be specified as a "trigger" action: an internal mental decision that sets the human motor system in action to take the actual action. The action itself happens a short time later. This process is modelled separately and is discussed below.

In our vending machine design, there are two reactive signals: `ChangeLight` prompts the user to trigger taking the change; the `PleaseWait` light prompts the user to wait (i.e intentionally do nothing).

```
[(ChangeLight ms, TriggerCHANGE); (PleaseWait ms, PAUSE)]
```

This list is an argument to the user model. The actions to be taken in this list are represented by numbers giving positions in the full list of actions. `PAUSE`, for example, is defined to be the number 4.

4.3 Communication Goals

People enter an interaction with some knowledge of the task that they wish to perform and do not just react to stimuli. In particular, they enter the interaction with communication goals: a task dependent mental list of information they know they must communicate to the device [2]. For example, on approaching a vending machine, a person knows that they must make a selection. Similarly, they know they must provide money. While inserting coins is not strictly a communication goal in cognitive science terms, here we treat it in the same way. Communication goals are important because a user may take an action as a result not of some stimulus from the machine but as a result of seeing an apparent opportunity to discharge a communication goal. For example, if on approaching a rail ticket machine the first button seen is that with the desired destination on, the person may press it, irrespective of any guidance the machine is giving. No fixed order can be assumed over how communication goals will be discharged if their discharge is apparently possible. Otherwise user errors will occur where they are discharged in the "wrong" order.

Communication goals are modelled as guard-action pairs as for reactive signals. The guard describes the situation under which the discharge of the communication goal can be attempted. It will include a label signal indicating that the input exists and that it corresponds to the desired action. The action of the pair is a mental trigger (of the form introduced for reactive behaviour) to take the action that will discharge the communication goal. We include an additional guard to this rule, stating that the trigger will only be fired if the user's main goal has not yet been achieved.

```
¬(goal t) ∧ (guard t) ∧ NEXT flag user_actions action t
```

As for reactive behaviour a list of pairs are supplied to correspond to each communication goal. A similar recursive definition to `REACTIVE` above is defined.

As the user believes they have achieved a communication goal, it is removed from their mental list (this differs from reactive signals whose presence is determined by the device). This is modelled by a daemon within the user model. It

monitors the actions taken by the user, removing any from the list used for the subsequent cycle. This is specified using a relation FILTER. It recurses down a communication goal list of pairs. If the action of any pair is true at the time of interest (meaning the action was triggered at that time), then that pair is removed from the list. FILTER_HLIST asserts that at any time the communication goals at the next time are the filtered version from a cycle earlier.

```
(FILTER [] t = []) ∧
(FILTER (CONS a actions) t =
  (if Action a t then  FILTER actions t else CONS a (FILTER actions t))

FILTER_HLIST hlist = ∀t. hlist (t+1) = FILTER (hlist t) t
```

For our task of obtaining chocolate from a vending machine there are initially two communication goals. When a user wishes to use a vending machine (whatever its design) they know they must possess and insert a coin, and that they must make a selection. Both can only occur if the user sees the opportunity to do so i.e. if the inputs exist and are labelled. For the coin slot the signal InsertCoinMessage from the device does this. For the selection the corresponding signal is PushChocMessage. They are specified as permanently true in the device specification for our hard-key interface as both are permanently visible. For the soft-key interface they will only be true at times when the machine will accept that action. The actions in this list are the appropriate mental trigger actions. The communication goal list thus has the form:

```
[((UserHasCoin us) ∧ (InsertCoinMessage ms), TRIGGERCOIN);
  (PushChocMessage ms, TRIGGERCHOC)]
```

This list means that if the user has a coin and the insertcoin message and coin slot are visible the user may mentally trigger coin insertion and so insert a coin. Alternatively if the push chocolate button label is visible, the user may ultimately push their selection. The guards make the actions a more accurate description of rational user behaviour. They mean the user only takes an action when they can see that it does correspond to the communication goal they wish to discharge. This example illustrates how guards can be general expressions and are not restricted to just signals. It should be noted that in the above context the conjunction is actually a higher-order, lifted conjunction. It does not take boolean arguments but boolean functions that, if supplied with a specific time, act as a boolean conjunction for that time point.

We must also specify what the user's goal is as it forms part of each communication goal guard. For the vending machine it is to obtain chocolate: UserHasChoc us. This is just an accessor function applied to the user state. It is a signal about the user's possessions. We must also provide a higher-order relation argument to the user model which specifies what possessions the user has and in what circumstances they are obtained and given up. For example, the user gains chocolate if they take it from the machine. A pre-defined generic relation POSSESSIONS is used that enforces real world axioms on physical possessions. For our example, we supply it with the details of the user having chocolate, change and coins and how they relate to user actions within a concrete relation CPOSSESSIONS.

4.4 Mental Triggers

Rather than associate an external stimulus directly with an external action using the disjunctive rules, we associate them with new mental actions that trigger the process of taking the action. More specifically they are associated with a point in time when the body's motor system has been triggered and cannot be stopped. These mental actions are included in the list of all possible actions. They are used as the actions of the rules discussed so far. A further category of rules is then introduced whereby if one of these internal actions is taken on a cycle then the next action will be the externally visible action it triggers. Unlike the other rules the internal trigger must have been fired on the previous cycle not the current one. This means there is always at least a one cycle delay between the trigger and external action.

```
trigger (t-1) ∧ NEXT flag user_actions action t
```

A recursive function, `TRIGGERS` combines a list of triggers into a series of choices.

The user model must be supplied with a list argument linking mental triggers with external actions. This uses the same format as the other arguments: a list of pairs where the first is a guard and the second is the corresponding action that may be taken if the guard is true. For our interactive system, we associate triggers with taking change, inserting a coin and making a selection:

```
[(TriggerChange us, TAKECHANGE); (TriggerCoin us, INSERTCOIN);
 (TriggerChoc us, PUSHCHOC)]
```

Mental triggers are given a higher priority than the other non-deterministic rules. If a trigger is fired then it will be the next action taken. Only if no fired trigger is outstanding do the other rules come into play. This is specified using if-then-else rather than disjunction.

if `ActionTriggered trigs t` *then* `TRIGGERS flag user_actions trigs t`
else *non-deterministic rules*

The test of the *if* expression, `ActionTriggered`, scans down the list of trigger pairs, and checks if the guard of any is true on the cycle of interest, `t`.

Modelling mental triggers means we can detect problems where the interface presents a series of options to the user but then, with no input from the user, removes or changes the meaning of one of those options. If the person has mentally triggered an action then even if they are aware that something has changed they will be unable to stop themselves and so do the wrong thing. In our user model the delay between trigger and actions models this. The trigger signals also give a way that user goals can be specified in terms of the actions the user intends to take rather than in terms of the actions they do take. This problem arose (and was fixed) in the design of a safety-critical application: the Computer Entry and Readout Display of an air-traffic control tool designed by Praxis. The original design meant that an air traffic controller could delete a high priority emergency message without having read it, when they were intending to delete a lower priority message that they had read [15].

4.5 Completion

In achieving a goal, subsidiary tasks are often generated. For the user to complete the task associated with their goal they must also complete all the subsidiary tasks. Examples of such tasks include taking the card back from a cash machine and taking change from a vending machine [6]. One way to specify these tasks would be to explicitly describe each such task. Instead we use the more general concept of an *interaction invariant*. This makes it easier to make the user model generic. The underlying reason why these tasks must be performed is that in interacting with the system some part of the state must be temporarily perturbed in order to achieve the desired task. Before the interaction is completed such perturbations must be undone. For example, to get money from a cash machine the card must be inserted and later returned. A condition on the state that holds at the start of the interaction, must be restored by the end. We specify the need to perform these completion tasks indirectly by supplying this interaction invariant as a higher-order argument to the user model. The interaction invariant is thus an invariant in a similar sense to a loop invariant in program verification. Full task completion involves not only completing the user's goal, but also restoring the invariant by completing all the subsidiary tasks generated in the process.

For a vending machine, the task is not completed just when the user has chocolate, but when they have also taken their change. The interaction invariant, `INVARIANT`, can be based on the value of the user's possessions. After the interaction this value should be at least as great as it was at the start (time 1). When the coin is inserted the value will drop and only return to its initial value once both chocolate and change are taken. The value of a user's possessions is calculated from possession count and value fields using a relation `VALUE`.

```
INVARIANT poss state t = (VALUE poss state t ≥ VALUE poss state 1)
```

This relation applied to the relation specifying the user's possessions described above is an argument to the user model.

We assume that on completing the task in this sense of goal achieved *and* invariant restored, the interaction is terminated by the user, irrespective of any other possible actions apart from actions already mentally triggered. This is modelled in the same way as mental triggers using if-then-else constructs rather than disjunction. A special user action `finished` indicates that the user has terminated the interaction (for whatever reason). We assume that once an interaction has been terminated then it remains terminated. If the user has not already terminated the interaction then any triggered actions will be followed. If no triggers are outstanding and both the goal has been achieved and the invariant restored then the user will terminate the interaction. Otherwise one of the non-deterministic rules will be fired. These rules are incorporated into a nested *if-then-else* expression with the mental trigger rules:

```
if finished (t-1) then  NEXT user_actions finished t
else if  ActionTriggered trigs t then  TRIGGERS flag user_actions trigs t
else if  (invariant t) ∧ (goalachieved t) then  NEXT user_actions finished t
else  non-deterministic rules
```

Cognitive psychology studies have shown that users also persistently terminate interactions when only the goal itself has been achieved [6] if the device design allows it. The above termination rule is therefore not sufficient to model actual human behaviour. An extra non-deterministic rule is needed:

`goalachieved t` $\wedge$ `NEXT flag user_actions finished t`

The model also terminates an interaction when no rational action is available. This rule acts as a final default case in the user model. In practice in this situation people could behave in a range of ways including pressing buttons at random. Our model treats a situation where no rational action is available as a user error.

4.6 The Flag

A further aspect of the user model is the flag argument of relation `NEXT`. It is of purely technical use to allow the user model to be specified over intervals rather than point time-wise. The `NEXT` relation asserts that one of the rules discussed always applies on a cycle. However, as each rule asserts a property of an interval rather than of a single cycle this would mean that on consecutive cycles two different rules could each inconsistently assert what the next action taken is if the device or user state changed in the meantime. The flag is simply used to "turn off" these rules on cycles after `NEXT` has been asserted until the cycle after its action has been taken. Thus only one rule is ever asserting facts about the state for any one cycle. One component of the user state is designated to act as the flag and the user model contains an additional outer disjunction combining the *if-then-else rules* discussed above and negation of the flag

$\forall$`t.` $\neg$`(flag t)` $\vee$ *`if-then-else rules`*

4.7 Correctness Theorem

We now consider the theorem to be proved about an interactive system. The usability property we are interested in is that if the user interacts rationally with the machine, based on their goals and knowledge, then they will complete the task for which they started the interaction. As noted earlier, task completion is more than just goal completion. The property that we require to hold is that eventually both the goal has been achieved and the interaction invariant restored. If the user model can terminate the interaction before the whole task is completed then the design contains a usability problem that will lead to users making systematic errors.

The user model and the device specification are both described by relations. The device relation is true of its input and output arguments if they describe consistent input-output sequences of the device. Similarly, the user model relation is true if the inputs (observations) and outputs (actions) are consistent sequences for a rational user. The combined system is specified as the conjunction of the instantiated user model and the specification of the device. The task completion theorem we wish to prove thus has the form below. If a theorem of

this form can be proved, then even if a user is capable of making the rational errors considered, then that potential will not affect the completion of the task: the errors will never manifest themselves.

```
⊢ ∀ state traces .
    initial state ∧ device specification ∧ user model
      ⊃ ∃t. (invariant t) ∧ (goalachieved t)
```

For example, we proved the following theorem for our machines showing that they do not contain the user errors considered. Only the machine specification MACHINE_SPEC is different for the two theorems proved:

```
⊢ ∀s us ms COINVAL CHANGEVAL CHOCVAL.
  (COINVAL = CHANGEVAL + CHOCVAL) ∧
  (s 1 = RESET) ∧ INIT_USER us ∧ MACHINE_USER us ms ∧ MACHINE_SPEC s ms ⊃
    ∃t.(UserHasChoc state t) ∧
       (INVARIANT (CPOSSESSIONS CHOCVAL COINVAL CHANGEVAL) us t)
```

MACHINE_USER is the instantiated user model. Its only arguments are the user and machine state. All the other arguments have been instantiated. The theorem contains assumptions that the vending machine is in its initial state at the start of the interaction and that the user also starts the interaction in an appropriate state (specified by INIT_USER). In particular they must start with no mental triggers fired, they must not have terminated the interaction already, they must have at least one coin and the user model flag must not be set.

The arguments to the predicate CPOSSESSIONS give details of the values of the user's possessions. The correctness theorem we proved is generic in a very simple way: with respect to the value of coins, change and chocolate. They are represented by variables COINVAL, CHANGEVAL and CHOCVAL rather than by fixed integers. The correctness theorem contains an assumption that gives the restriction that the values concerned must satisfy: COINVAL = CHANGEVAL + CHOCVAL The correctness theorem holds for any triple of values that satisfy this relation. This demonstrates with a simple example how a single theorem can be proved about a range of interactive systems. The specification and theorems could be made generic in other ways such as in the range of selections and prices available.

We have proved the task completion theorem for both designs using symbolic simulation by proof within the HOL theorem prover [11]. Our verification is fully machine-checked. An induction principle concerning the stability of a signal is used repeatedly to step the simulation over periods of inactivity between a rule activating and the action actually happening. We prove a series of lemmas that, given assumptions about the machine and user state, assert a relation about the state after the next action is taken, either by the the user or the machine. These theorems are currently proved semi-automatically. A series of lemmas must be proved for each. However, many of the lemmas are proved automatically or are reusable. For the second of our two designs, the proof did not need to be changed but was run automatically. As the lemmas are of a very standard form it should be straightforward to automate their formulation as well as proof.

5 Summary and Further Work

We have presented an extension to our methodology for verifying interactive systems using HOL that is based on the use of a generic user model with explicit description of a user interface. We demonstrated how devices with both hard and soft-key interfaces can be modelled and thus verified using this approach. New classes of design problems that result in user errors related to soft-key interfaces can be detected. We illustrated our approach by considering a simple vending machine design, with two different interfaces. The example is simplified to illustrate the general approach – for example we abstract multiple selection to a single button and consider interactions starting from a specific reset state. In future work we will investigate such issues.

The user model describes a user who is capable of making a range of systematic errors when behaving in a simple rational way: but only if interacting with poorly designed interfaces. The model does not imply that mistakes will always be made, just that the potential is there. If a device design is such that users can make systematic errors whilst behaving in the rational way defined then it will be impossible to prove that the task can always be completed. It should be noted that we define classes of errors not by their effects but by their cognitive cause. We do not claim that in proving the absence of a particular error that a similar effect might not happen due to some other cause such as a alarm ringing.

We demonstrated previously [9] that our approach could be used to detect the possibility of three classes of error: post-completion errors, communication goal based errors and delay errors. By explicitly modelling aspects of the interface as described here we can also detect errors that arise due to the interface not providing device-dependent information to the user at a point where it is needed. It would not be possible to prove that the user takes a necessary action to complete the task if the label guard was not true at the appropriate point. With soft-key interfaces, a less obvious class of error arises where the person has triggered an action but the effect of the action changes before completion.

Whilst this work demonstrates the usefulness of modelling aspects of the user interface within our approach, our treatment of the interface is relatively simplistic. In future work we will investigate more detailed modelling of the interface and issues that arise from more complex soft-key interfaces such as window-based GUIs. We also intend to improve the accuracy of the user model, increasing the range of design-induced user errors it can detect. We are currently investigating the verification of an Air Traffic Control system. The proofs are currently only partially automated. However, much of the proof work is routine and due to the use of a common user model contains much similarity. We intend to more fully automate the process.

It is not possible to completely eradicate user errors from an interactive system with any degree of useful functionality, since people do sometimes behave randomly for accidental, irrational etc reasons. However, by providing a methodology for detecting designs that allow users behaving in a simple rational way to make systematic errors, the usability of interactive systems can be improved.

Acknowledgements This work is funded by EPSRC grant GR/M45221.

References

1. R. Back, A. Mikhajlova and J. von Wright. *Modeling Component Environments and Interactive Programs Using Iterative Choice*. Turku Centre for Computer Science Technical Report No 200, September 1998.
2. A. Blandford and R. Young. The role of communication goals in interaction. In *Adjunct Proceedings of HCI'98*, 1998.
3. P. Bumbulis, P.S.C. Alencar, D.D. Cowan and C.J.P. Lucena. Validating Properties of Component-based Graphical User Interfaces. In F. Bodart and J. van der Donckt, editors, *Design, Specification and Verification of Interactive Systems '96*, pages 347–365. Wien : Springer, 1996.
4. R. Butterworth, A. Blandford, and D. Duke. Demonstrating the cognitive plausibility of interactive system specifications. To appear in *Formal Aspects of Computing*.
5. R. Butterworth, A. Blandford, and D Duke. Using formal models to explore display based usability issues. *J. of Visual Languages and Computing*, 10:455–479, 1999.
6. M. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.
7. J. Campos and M. Harrison. Formally verifying interactive systems: a review. In M. D. Harrison and J. C. Torres, editors, *Design, Specification and Verification of Interactive Systems '97*, pages 109–124. Wien : Springer, 1997.
8. P. Curzon and A. Blandford. Reasoning about order errors in interaction. Supplementary Proceedings of TPHOLs2000.
9. P. Curzon and A. Blandford. Detecting multiple classes of user errors. To appear in *Proc. of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction*, LNCS. Springer-Verlag, 2001.
10. D. Duke, P. Barnard, D. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–394, 1998.
11. M. Gordon and T. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. CUP, 1993.
12. T. Moher and V. Dirda. Revising mental models to accommodate expectation failures in human-computer dialogues. In *Design, Specification and Verification of Interactive Systems '95*, pages 76–92. Springer, 1995.
13. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
14. F. Paterno' and M. Mezzanotte. Formal analysis of user and system interactions in the CERD case study. In *Proc. of IFIP Working Conference on Engineering for Human-Computer Interaction*, pages 213–226. Chapman and Hall, 1995.
15. S. Buckingham Shum *et al.* Multidisciplinary modelling for user-centred design: An air-traffic control case study. In M.A. Sasse, R.J. Cunningham, and R.L. Winder, editors, *Proc. of HCI'96*, pages 201–220. Springer-Verlag, 1996.