Formally Justifying User-centred Design Rules: a Case Study on Post-completion Errors

Paul Curzon[1] and Ann Blandford[2]

[1]Middlesex University, Interaction Design Centre, Bramley Road, London N14 4YZ
[2]University College London Interaction Centre, Remax House, 31-32 Alfred Place, London WC1E 7DP
p.curzon@mdx.ac.uk, a.blandford@ucl.ac.uk

Abstract. Interactive systems combine a human operator with a computer. Either may be a source of error. The verification processes used must ensure both the correctness of the computer component, and also minimize the risk of human error. Human-centred design aims to do this by designing systems in a way that make allowance for human frailty. One approach to such design is to adhere to design rules. Design rules, however, are often ad hoc. We examine how a formal cognitive model, encapsulating results from the cognitive sciences, can be used to justify such design rules in a way that integrates their use with existing formal hardware verification techniques. We consider here the verification of a design rule intended to prevent a commonly occurring class of human error know as the post-completion error.

Keywords

Cognitive architecture, user error, design rules, formal verification.

1 Introduction

Interactive computer systems are systems that combine a human operator with a computer system. Such a system needs to be both correct and usable. With the increasing ubiquity of interactive computer systems, usability becomes increasingly important. Minor usability problems can scale to having major economic and social consequences. Usability has many aspects. We concentrate on one aspect: user error. Humans are naturally prone to error. Such error is not predictable in the way the behaviour of a faulty computer may be. However, much human error is systematic and as such can be modelled and reasoned about.

Design approaches to prevent usability problems often tend to be ad hoc: following lists of design rules, sometimes apparently contradictory, that are based on the experience of HCI experts. Furthermore the considerations of usability experts are often far removed from those of hardware verification approaches, where the emphasis is on correctness of the system against a functional specification. In this paper we consider how the two worlds of formal hardware verification and human-centred usability verification can be integrated. We propose a way in which usability design rules can be both formalised and derived from formalised principles of cognition within the same framework as hardware verification. We illustrate the approach by considering one well-studied and widely occurring class of systematic human error: the post-completion error. A post-completion error occurs when a user achieves their main goal but omits 'clean up' actions; examples include making copies on a photocopier but forgetting to retrieve the original and forgetting to take change from a vending machine.

We first define simple **principles of cognition**. These are principles that generalise the way humans act in terms of the mental attributes of knowledge, tasks and goals. The principles are not intended to be exhaustive, but to cover a variety of classes of cognitive behaviour of interest, based on the motor system, simple knowledge-based cognition, goal-based cognition, *etc.* They do not describe a particular individual, but generalise across people as a class. They are each backed up by evidence from HCI and/or psychology studies. Those presented are not intended to be complete but to demonstrate the approach. We have developed a generic **formal cognitive model** of these principles in higher-order logic. By "generic" we mean that it can be targeted to different tasks and interactive systems. Strictly this makes it a cognitive architecture [16]. In the remainder of the paper we will refer to the generic model as a *cognitive architecture* and use the term *cognitive model* for a version of it instantiated for a given task and system. The underlying principles of cognition are formalised once in the architecture, rather than having to be re-formalised for each new task or system of interest. Whilst higher-order logic is not essential for this, its use makes the formal specifications simpler than the use of a first-order logic would.

The principles, and more formally the cognitive architecture, specify **cognitively plausible behaviour** (see [7]). That is, they specify possible traces of user actions that can be justified in terms of the specific principles. Of course users might also act outside this behaviour, about which situations the model says nothing. Its predictive power is bounded by the situations where people act according to the principles specified. All theorems in this paper are thus bounded by that assumption. That does not preclude useful results from being obtained, provided their scope is remembered. The architecture allows us to investigate what happens if a person does act in such plausible ways. The behaviour defined is neither "correct" nor "incorrect". It could be either depending on the environment and task in question. It is, rather, "likely" behaviour. We do not model erroneous behaviour explicitly. It emerges from the description of cognitively plausible behaviour. The focus of the description is on the internal goals and knowledge of a user. This contrasts with a description of a user's actions as, say, a finite state machine that makes no mention of such cognitive attributes.

After describing the architecture, we next formalise a particular class of systematic user error, that is made in a wide range of situations, in terms of the cognitive architecture. We also formalise a simple and well known usability design rule that, if followed, eliminates this class of error. We prove a theorem that states that if the design rule is followed, then the erroneous behaviour cannot occur *due to the specified cause* as a result of a person behaving according to the principles of cognition formalised.

The design rule is initially formalised in user-centred terms. To enable the integration with machine-centred verification, we next reformulate it in a machine-centred way, ultimately proving that a machine-centred version of the design rule implies the absence of the class of error considered. Even though the cognitive architecture is capable of making the error, the design rule ensures that the user environments (as provided by the computer part of the system) in which it would emerge do not occur. Other errors are, of course, still possible. The main contribution of this paper is to demonstrate a way that formal reasoning about design rules can be achieved based on a cognitive architecture but within the same framework as verification of other aspects.

We have used the HOL interactive proof system [15] so theorems are machine-checked. Given the relative simplicity of the theorems this is not essential in that hand proofs alone would have been possible. Machine-checked proof does give an extra level of assurance over that of the informal proofs upon which they are based. Furthermore our work sets out a framework in which these theorems can be combined with complex machine-checked hardware verification. Machine-checking of the design rule proofs maintains a consistent treatment. Finally, this work aims to demonstrate a general approach. For more complex design rules, the proofs may be harder so machine-checking may be more directly useful.

2 Related Work

There are several approaches to formal reasoning about the usability of interactive systems. One approach is to focus on a formal specification of the user interface [9]. Most commonly it is used with model-checking-based verification; investigations include whether a given event can occur or whether properties hold of all states. In contrast, Bumbulis *et al* [5] verified properties of interfaces based on a guarded command language embedded in the HOL system. Back *et al* [1] illustrate how properties can be proved and data refinement performed of a specification of an interactive system. However, techniques that focus on the interface do not directly support reasoning about design problems that lead to users making systematic errors; also, the usability properties checked are necessarily device-specific and have to be reformulated for each system verified.

An alternative is formal user modelling of the underlying system. It involves writing both a formal specification of the computer system and one of the user, to support reasoning about their conjoint behaviour. Both system and user are considered as central components of the system and modelled as part of the analysis. Doing so provides a conceptually clean method of bringing usability concerns into the domain of traditional verification in a consistent way. Duke *et al* [13] express constraints on the channels and resources within an interactive system; this approach is particularly well suited to reasoning about interaction that, for example, combines the use of speech and gesture. Moher and Dirda [21] use Petri net modelling to reason about users' mental models and their changing expectations over the course of an interaction; this approach supports reasoning about learning to use a new computer system but focuses on changes in user belief states rather than proof of desirable properties. Paterno' and Mezzanotte [22] use LOTOS and ACTL to specify intended user behaviours and hence reason about interactive behaviour.

Our work complements these uses of formal user modelling. None of the above focus on reasoning about user errors. Models typically describe how users are intended to behave: they do not address human fallibility. If verification is to detect user errors, a formal specification of the user, unlike one of a computer system, is not a specification of the way a user should be; rather, it is a description of the way they are [7]. Butterworth *et al* [6] do take this into account, using TLA to reason about reachability conditions within an interaction. Rushby [25] formalised plausible mental models of systems, looking for discrepancies between these and actual system behaviour. However, like interface-oriented approaches, each model is individually hand-crafted for each new device in this work.

An approach to interactive system verification that focuses directly on errors is exemplified by Fields [14]. He models erroneous actions explicitly, analysing the consequences of each possible action. He thus models the effect of errors rather than their underlying causes. A problem of this approach is the lack of discrimination about which errors are the most important to consider. It does not discriminate random errors from systematic errors which are likely to re-occur and so be most problematic. It also implicitly assumes there is a "correct" plan, from which deviations are errors.

The University of Queensland's safeHCI project [20] has similar aims and approach to our overall project, combining the areas of cognitive psychology, human-computer interaction and system safety engineering. The details differ, however. SafeHCI has had a focus on hazard analysis and system-specific modelling, whereas our work has an emphasis on generic cognitive models.

Approaches that are based on a cognitive architecture (e.g. [19][17][23]) model underlying cognitive causes of errors. However, the modelling exemplified by these approaches is too detailed to be amenable to formal proof. Our previous work [11] followed this approach but at a coarser level of detail, making formal proof tractable. In this approach general mechanisms of cognition are modelled and so need be specified only once, independent of any given interactive system. Furthermore, by explicitly doing the verification at the level of underlying cause, on failed verification, a much greater understanding of the problem is obtained. Rather than just knowing the manifestation of the error – the actions that lead to the problem – the failed proof provides understanding of the underlying causes.

Blandford *et al* [4] have used a formal model of user behaviour to derive high level guidance. There the emphasis is on a semi-formal basis underpinning the craft skill in spotting when a design has usability problems. We are concerned here with guidance for a designer rather than for a usability analyst. We focus on the verification of general purpose design rules rather than the interactive systems themselves.

```
USER flag actions commitments commgoals init_commgoals stimulus_actions
        possessions finished finishedpos goalachieved invariant
        (ustate:'u) (mstate:'m) =
  (USER_CHOICE flag actions commitments commgoals stimulus_actions
        finished finishedpos goalachieved invariant ustate mstate) ∧
  (USER_UNIVERSAL actions commgoals possessions finished flag ustate mstate)

USER_CHOICE flag actions commitments commgoals stimulus_actions
        finished finishedpos goalachieved invariant ustate mstate =
(∀t.
  ¬(flag t) ∨
  (IF (finished ustate (t-1))
   THEN (NEXT flag actions finishedpos t)
   ELSE IF ((CommitmentMade (CommitmentGuards commitments) t)
   THEN (COMMITS flag actions commitments t)
   ELSE IF TASK_DONE (goalachieved ustate) (invariant ustate) t
   THEN (NEXT flag actions finishedpos t)
   ELSE USER_RULES flag actions commgoals stimulus_actions finishedpos
        goalachieved mstate ustate t))))

USER_RULES flag actions commgoals stimulus_actions finishedpos goalachieved
        (mstate:'m) (ustate:'u) t =
  COMPLETION flag actions finishedpos goalachieved ustate t ∨
  REACTS flag actions stimulus_actions t ∨
  COMMGOALER flag actions commgoals goalachieved ustate mstate t ∨
  ABORTION flag actions finishedpos goalachieved commgoals stimulus_actions
        ustate mstate t
```

Fig. 1. The USER relation

Providing precision to ensure different people have the same understanding of a concept has been suggested as the major benefit of formal models in interaction design [3]. One approach would therefore be to just formalise the design rules (see [3], [24]). In our approach, we not only formalise design rules, we also prove theorems justifying them based on underlying principles about cognition embodied in a formal cognitive architecture. In this way the design rules are formally demonstrated to be correct, up to the assumptions of the principles of cognition. This gives extra assurance to those applying the design rules. This approach builds on our previous work where informal argument only was used to justify the effectiveness of design rules [12]. We show here how this can be formalised in the same framework as other forms of verification.

3 Formalising Cognitively Plausible Behaviour

We first describe our cognitive architecture. It is specified by a higher-order logic relation `USER`, the top levels of which are given in Figure 1. It takes as arguments information such as the user's goal, `goalachieved`, a tuple of actions that the user may take, `actions`, *etc.* The final two arguments, `ustate` and `mstate`, each of polymorphic type as specified by the type variables `'u` and `'m`, represent the user state and the machine state over time. The specific type is only given when the architecture is instantiated for a given interaction. These states record over time the series of mental and physical actions made by the user, together with a record of the user's possessions. They are instantiated to a tuple of history functions: functions of type `time` $\rightarrow$ `bool`, from time instances to a boolean indicating whether that signal is true at that time (i.e. the action is taken, the goal is achieved, etc). The other arguments to `USER` specify accessor functions to one of these states. For example, `finished` is of type `'u` $\rightarrow$ `time` $\rightarrow$ `bool`. Given the user state it returns a history function that for each time instance indicates whether the user model has terminated the interaction. The other arguments of the model will be examined in more detail as needed in the explanation of the model below.

The `USER` relation is split into two parts. The first, `USER_CHOICE`, models the user making a choice of actions. It formalises the action of the user at a given time as a series of rules, one of which is followed at each time instance. `USER_UNIVERSAL` specifies properties that are true at all time instances, whatever the user does. For example, it specifies properties of possessions such that if an item is not given up then the user still has it. We focus here on the choice part of the model as it is most relevant to the concerns of this paper. `USER_CHOICE` is therefore described in detail below. In outline, it states that the next user action taken is determined as follows:

if the interaction is finished
then it should remain finished
else if a physical action was previously decided on
then the physical action should be taken
else if the whole task is completed
then the interaction should finish
else an appropriate action should be chosen non-deterministically

The cognitive architecture is ultimately, in the final else case above, based on a series of non-deterministic temporally guarded action rules, formalised in relation `USER_RULES`. Each describes an action that a user *could* plausibly make. The rules are grouped (e.g. in definition `REACTS` in Figure 1) corresponding to a user performing actions for specific cognitively related reasons. Each such group then has a single generic description. Each rule combines a pre-condition such as a particular message being displayed, with an action, such as a decision made to press a given button at some later time.

rule 1 fires asserting its action is taken $\vee$
rule 2 fires asserting its action is taken $\vee$
...
rule n fires asserting its action is taken

Apart from those included in the if-then-else staircase of `USER_CHOICE`, no further priority ordering between rules is modelled. We are interested in whether an action is cognitively plausible at all (so could be systematically taken), not whether one is more likely than another. We are concerned with design rules that prevent any systematic erroneous action being taken even if in a situation some other action is more likely anyway. The architecture is a relation. It does not assert that a rule will be followed, just that it may be followed. It asserts that the behaviour of any rule whose guards are true at a point in time is cognitively plausible at that time. It cannot be deduced that any specific rule will be the one that the person will follow if several are cognitively plausible.

The architecture is based on a temporal primitive, `NEXT` that specifies the next user action taken *after* a given time. `NEXT flag actions action t` states that the `NEXT` action performed *after* time `t` from a list of all possible user actions, `actions`, is `action`. It asserts that the given action's history function is true at some first point in the future, and that the history function of all other actions is false up to that point. The action argument is of type integer and specifies the position of the action history function in the list `actions`. The `flag` argument to `NEXT` and `USER` is a specification artifact used to ensure that the time periods that each firing rule specifies do not overlap. It is true at times when a new decision must be made by the model. The first line of `USER_CHOICE` in Figure 1 thus ensures, based on the truth of the flag, that we do not re-specify contradictory behaviour in future time instances to that already specified. Consider the first if-then-else statement of `USER_CHOICE` in Figure 1 as an example of the use of `NEXT`. The `action` argument of `NEXT` is instantiated to `finishedpos`. It states that if the interaction was finished then the next action remains finished: once the interaction has terminated the user takes no other action.

We model both **physical and mental actions**. A person decides (making a mental action) to take a physical action before it is actually taken. Once a signal has been sent from the brain to the motor system to take the physical action, the signal cannot be revoked even if the person becomes aware that it is wrong before the action is taken. Each physical action modelled is thus associated with an internal mental action that commits to taking it. The argument `commitments` to the relation `USER` is a list of pairs that links the mental and physical actions. `CommitmentGuards` extracts a list of all the mental actions (the first elements of the pairs). The recursively defined, `CommitmentMade` checks, for a given time instance, `t`, if any mental action was taken in the previous time instance (`cmt(t-1)`):

$$\begin{array}{l}(\text{CommitmentMade } [\,]\ t = \text{FALSE}) \wedge \\ (\text{CommitmentMade } (\text{cmt} :: \text{rest})\ t = (\text{cmt}(t\text{-}1)) \vee (\text{CommitmentMade rest } t))\end{array}$$

If a mental action, `mact`, made a commitment to a physical action `pact` on the previous cycle (time, `t-1`) then that will be the next action taken. Definition `COMMITS` asserts this disjunctively for the whole list of commitments:

$$\text{COMMIT flag actions mact pact } t = (\text{mact } (t\text{-}1)) \wedge \text{NEXT flag actions pact } t$$

(COMMITS flag actions [] t = FALSE) $\wedge$
(COMMITS flag actions (l :: commits_actions) t =
  ((COMMITS flag actions commits_actions t) $\vee$
  (COMMIT flag actions (CommitmentGuard l) (CommitmentAction l) t)))

Based on these definitions the second if statement of `USER_CHOICE` in Figure 1 states that if a mental action is taken on a cycle then the next action will be the externally visible action it committed to. The physical action already committed to by a mental action is thus given high priority as modelled by being in the if-then-else staircase.

Task-based termination behaviour: In the third if statement, `USER_CHOICE` specifies that a user will terminate an interaction when their whole task is achieved. The user has a goal and the task is not completed until that goal is achieved. We must therefore supply a relation argument `goalachieved` to the cognitive architecture that indicates over time whether the goal is achieved or not. With a vending machine, for example, this may correspond to the person's possessions including chocolate. Similar to `finished`, `goalachieved` extracts from the state a history function that, given a time, returns a boolean value indicating whether the goal is achieved at that time. Note that `goalachieved` is a higher-order function and can as such represent an arbitrarily complex condition. It might, for example, be that the user has a particular object as above, that the count of some series of objects is greater than some number or a combination of such atomic conditions.

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 subsidiary tasks. The underlying reason for these tasks being 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. Examples of such tasks with respect to a vending machine include taking change. 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 [11]: a higher-order argument to the cognitive architecture. The interaction invariant is an invariant at the level of abstraction of whole interactions in a similar sense to a loop invariant in program verification. For example, the invariant for a simple vending machine might be true when the total value of the user's possessions (coins and chocolate) have been restored to their original value, the user having exchanged coins for chocolate of the same value. Task completion involves not only completing the user's goal, but also restoring the invariant.

TASK_DONE goalachieved invariant t = (goalachieved t $\wedge$ invariant t)

We assume that on completing the task in this sense, the interaction will be considered terminated by the user unless there are physical actions already committed to. It is therefore modelled in the if-then-else staircase of `USER_CHOICE` to give it priority over other rules apart from committed actions.

We next examine the non-deterministic rules in the final else case of `USER_CHOICE` that form the core of the model and are defined in `USER_RULES`.

COMPLETION: Cognitive psychology studies have shown that users intermittently, but persistently, terminate interactions as soon as their goal has been achieved [8]. This behaviour is formalised as a guarded rule. If the goal is achieved at a time then the next action of the cognitive architecture can be to terminate the interaction:

COMPLETION flag actions finished goalachieved (ustate:'u) t =
  (goalachieved ustate t) ∧ NEXT flag actions finished t

REACTS: A user may react to a stimulus from a device, doing the action suggested by it. For example, if a flashing light comes on next to the coin slot of a vending machine, a user might, if the light is noticed, react by inserting coins. In a given interaction there may be many different stimuli to react to. Rather than specify this behaviour for each, we define it generically. Relation `REACT` gives the rule defining what it means to react to a given stimulus. If at time `t`, the stimulus `stim` is active, the next action taken by the user out of possible actions, `actions`, at an unspecified later time, may be the associated `action`.

REACT flag actions stim action t = stim t ∧ NEXT flag actions action t

As there may be many reactive signals, the user model is supplied with a list of stimulus-action pairs: [(s1, a1); . . . (sn, an)]. `REACTS`, given a list of such pairs, recursively extracts the components and asserts the above rule about them. The clauses are combined using disjunction, so are non-deterministic choices, and this definition is combined with other non-deterministic rules. `Grd` and `Act` extract a pair's components.

(REACTS flag actions [] t = FALSE) ∧
(REACTS flag actions (s :: st) t =
    ((REACTS flag actions st t) ∨ (REACT flag actions (Grd s) (Act s) t)))

COMMGOALER: A user often enters an interaction with knowledge of the task, if not the device used to achieve it. They may, as a result, start with sub-goals that they know must be discharged to achieve their main goal. This kind of preconceived sub-goal is known as a *communication goal* [2]. For example, when the user has the goal of purchasing a ticket, they are likely to know that in some way the destination and ticket type must be specified as well as payment made. Communication goals are distinct from device dependent sub-goals that result from the person reacting to stimulus from the device or "tidying" sub-goals that restore a perturbation made to the device from the initial state. The precise nature of the action associated with a communication goal may not be known in advance. A communication goal specification is not a fully specified plan, in that no order of the corresponding actions may be specified. The way that these must be done and their order may not be known in advance. If the person sees an apparent opportunity to discharge a communication goal they *may* do so. Once they have done so they will not expect to need to do so again. No fixed order is assumed over how communication goals will be discharged if their discharge is apparently possible. Communication goals are a reason why people do not just follow instructions.

We model communication goals as guard-action pairs as for reactive signals. The guard describes the situation under which the discharge of the communication goal appears possible, such as when a virtual button actually is on the screen. As for reactive behaviour, the architecture is supplied with a list of (guard, action) pairs one for each communication goal. Unlike the reactive signal list that does not change through an interaction, communication goals are discharged. This corresponds to them disappearing from the user's mental list of intentions. We model this by removing them from the communication goal list when done. We do not go into detail of the formalisation of communication goals here as it is not directly relevant. The interested reader should see [11].

ABORTION: A user may terminate an interaction when there is no apparent action they can take that would help complete the task. For example, if on a touch screen ticket machine, the user wishes to buy a weekly season ticket, but the options presented include nothing about season tickets, then the person might give up, assuming their goal is not achievable. The model includes a final default non-deterministic rule, `ABORTION`, that models this case by just forming the negation of the guards of all other rules.

The features of the cognitive architecture discussed above concern aspects of cognition. An extension of the architecture for this paper over that of our previous work [11] as given in Figure 1 involves the addition of **probes**. Probes are extra signals that do not alter the cognitive behaviour of the architecture, but instead make internal aspects of its action visible. This allows specifications to be written in terms of hidden internal cognitive behaviour, rather than just externally visible behaviour. This is important for this work as our aim is to formally reason about whether design rules address underlying cognitive causes of errors not just their physical manifestation. The form of probe we consider here records for each time instance whether a particular rule fires at that instance. We require a single probe that fires when the goal-based termination rule described above fires. We formalise this using a function, `Goalcompletion` that extracts the goal completion probe from the collection of probes passed as an additional argument to the cognitive architecture. To make the probe record goal completion rule events, we add a clause specifying the probe is true to the rule concerning goal completion, `COMPLETION` given above:

$$\text{(Goalcompletion probes t)} \land \text{goalachieved t} \land \text{NEXT flag actions finished t}$$

Each other rule in the architecture has a clause added asserting the probe is false at the time it fires. For example the `REACT` rule becomes:

$$\text{(Goalcompletion probes t = FALSE)} \land \text{stim t} \land \text{NEXT flag actions action t}$$

A similar clause is also added to the part of the architecture that describes the behaviour when no rule is firing.

4 Verifying a User Error Design Rule

Erroneous actions are the immediate, obvious cause of failure attributed to human error, as it was a particular action (or inaction) that caused the problem: users pressing a button at the wrong time, for example. However, to understand the problem, and so minimize re-occurrence, approaches that consider the immediate causes alone are insufficient. It is important to consider *why* the person took that action. The ultimate causes can have many sources. Here we consider situations where the ultimate causes of an error are that limitations of human cognition have not been addressed in the design. An example might be that the person pressed the button at that moment because their knowledge of the task suggested it would be sensible. Hollnagel [18] distinguishes between human error `phenotypes` (classes of erroneous actions) and `genotypes` (the underlying psychological cause). He identifies a range of simple phenotypes such as repetition of an action, omission of actions, etc. In this paper, to demonstrate the feasibility of formally reasoning about design rules based on cognitively plausible behaviour, we consider one particular error genotype: the class of errors known as *post-completion errors* introduced in Section 1. A similar effect (i.e. phenotype) to a post completion error can occur for other reasons. However that would be considered a different class of error (genotype). Other design rules might be required to prevent it.

4.1 Formalising Post-completion Error Occurrence

In our cognitive architecture post completion error behaviour is modelled by the goal termination rule firing. Probe signal `Goalcompletion` records whether that particular rule has fired at any given time. Note that the rule can fire when the goal is achieved but does not have to. Note also that it firing is necessary but not sufficient for the cognitive architecture to make a post-completion error. In some situations it is perfectly correct for the rule to fire. In particular if the interaction invariant has been re-established at the point when it fires then an error has not occurred. Thus whilst the error occurring is a direct consequence of the existence of this rule in the model, the rule is not directly modelling erroneous actions, just cognitively plausible behaviour that leads to an erroneous action in some situations.

Definition `PCE_OCCURS` specifies that a post-completion error occurs if there is a time, `t`, before the end time of the interaction `te`, such that the `Goalcompletion` probe is true at that time but the invariant has not been re-established.

$$\begin{array}{l}\text{PCE\_OCCURS probes invariant te} = \\ \quad (\exists \text{t. t} \leq \text{te} \wedge \text{Goalcompletion probes t} \wedge \neg(\text{invariant t}))\end{array}$$

This takes two higher order arguments, representing the collection of probes indicating which rules fire and the relation indicating when the interaction invariant is established. A final argument indicates the end time of interest. It bounds the interaction under consideration corresponding to the point when the user has left and the machine has reset. The start time of the interaction is assumed to be time zero.

4.2 Formalising a Design Rule

We next formalise a well-known user-centred design rule intended to prevent a user having the opportunity to make a post-completion error. It is based on the observation that the error occurs because it is possible for the goal to be achieved before the task as a whole has been completed. If the design is altered so that all user actions have been completed before the goal then a post-completion error will not be possible. In particular any tidying up actions associated with restoring the interaction invariant must be either done by the user before the goal can possibly be achieved, or done automatically by the system. This is the design approach taken for British cash machines where, unlike in the original versions, cards are always returned before cash is dispensed. This prevents the post-completion error where the person takes the cash (achieving their goal) but departs without the card (a tidying task).

The formal version of the design rule states that for all times less than the end time, `te`, it is not the case that both the goal is achieved at that time and the task is not done. Here, `goalachieved` and `invariant` are the same as in the cognitive architecture.

$$\begin{array}{l} \text{PCE\_DR goalachieved invariant te} = \\ \quad (\forall \text{t. t} \leq \text{te} \supset \neg(\text{goalachieved t} \wedge \neg(\text{TASK\_DONE goalachieved invariant t}))) \end{array}$$

Thus when following this design approach, the designer must ensure that at all times prior to the end of the interaction it is not the case that the goal is achieved when the task as a whole is incomplete. The design rule was formulated in this way to match a natural way to think about it informally according to the above observation.

4.3 Justifying the Design Rule

We now prove a theorem that justifies the correctness of this design rule (up to assumptions in the cognitive architecture). If the design rule works, at least for users obeying the principles of cognition, then the cognitive architecture's behaviour when interacting with a machine satisfying the design rule should never lead to a post-completion error occurring. We have proved using HOL the following theorem stating this:

$$\begin{array}{l} \vdash \text{USER} \ldots \text{ goalachieved invariant probes ustate mstate} \wedge \\ \quad \text{PCE\_DR (goalachieved ustate) (invariant ustate) te} \supset \\ \qquad \neg(\text{PCE\_OCCURS probes (invariant ustate) te}) \end{array}$$

We have simplified, for the purposes of presentation the list of arguments to the relation `USER` which is the specification of the cognitive architecture, omitting those arguments that are not directly relevant to the discussion. One way to interpret this theorem is as a traditional correctness specification against a requirement. The requirement (conclusion of the theorem) is that a post-completion error does not occur. The conjunction of the user and design rule is a system implementation. The system is implemented by placing an operator (as specified by the cognitive architecture `USER`) with the machine (as minimally specified by the design rule). The definitions and theorem proved are generic. They do not specify any particular interaction or even task. A general, task independent design rule has thus been verified.

The proof of the above theorem is simple. It involves case splits on the goal being achieved and the invariant being established. The only case that does not follow immediately is when the goal is not achieved and the invariant does not hold. However, this is inconsistent with the goal completion rule having fired so still follows fairly easily.

4.4 Machine-centred rules

The above design rule is in terms of user concerns – an invariant of the form suitable for the cognitive model and a user-centred goal. Machine designers are not directly concerned with the user and this design rule is not in a form that is directly of use. The designer cannot manipulate the user directly, only machine events. Thus whilst the above rule and theorem are in a form of convenience to a usability specialist, they are less convenient to a machine designer. We need a more machine-centred design rule as below.

MACHINE_PCE_DR goalevent minvariant te =
  $(\forall t.\ \text{goalevent}\ t \supset (\forall t1.\ t \leq t1 \land t1 \leq te \supset \text{minvariant}\ t1))$

This design rule is similar to the user-centred version, but differs in several key ways. Firstly, the arguments no longer represent user based relations. The `goalevent` signal represents a machine event. Furthermore this is potentially an instantaneous event, rather than a predicate that holds from that point on. Similarly, the machine invariant concerns machine events rather than user events. Thus, for example with a vending machine, the goal as specified in a user-centred way is that the user has chocolate. Once this first becomes true it will continue to hold until the end of the interaction, since for the purposes of analysis we assume that the user does not give up the chocolate again until after the interaction is over. The machine event however, is that the machine fires a signal that releases chocolate. This is a relation on the machine state rather than on the user state: `GiveChoc mstate`. It is also an event that occurs at a single time instance (up to the granularity of the time abstraction modelled). The machine invariant is also similar to the user one but specifying that the value of the *machine's* possessions are the same as at the start of the interaction – it having exchanged chocolate for an equal amount of money. It is also a relation on the machine's state rather than on the user's state.

The ramification of the goal now being an instantaneous event is that we need to assert more than that the invariant holds whenever the goal achieved

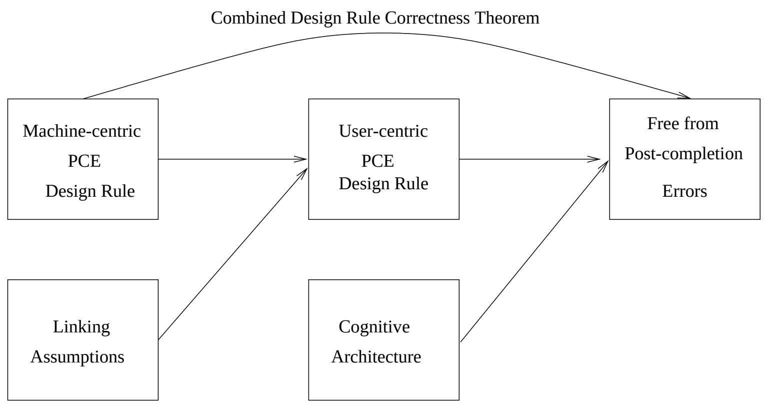
Combined Design Rule Correctness Theorem
Machine-centric PCE Design Rule
User-centric PCE Design Rule
Free from Post-completion Errors
Linking Assumptions
Cognitive Architecture

Fig. 2. Verifying the Design Rule in Stages

event holds. The invariant must hold from that point up to the end of the interaction. That is the reason a new universally quantified variable `t1` appears in the definition, constrained between the time the goal event occurs and the end of the interaction.

We prove that this new design rule implies the original, provided assumptions are met about the relationship between the two forms of goal statements and invariants. It is these assumptions that form the basis of the integration between the user and machine-centred worlds.

$$\begin{array}{l}\vdash (\forall t.\ \text{minvariant } t \supset \text{invariant } t) \wedge \\ \quad (\forall t.\ (\text{goalachieved } t) \supset \exists t2.\ t2 \leq t \wedge (\text{goalevent } t2)) \supset \\ \qquad \text{MACHINE\_PCE\_DR goalevent minvariant te} \supset \\ \qquad\quad \text{PCE\_DR goalachieved invariant te}\end{array}$$

This asserts that the machine based design rule `MACHINE_PCE_DR` does indeed imply the user-centred one `PCE_DR`, under two assumptions. The first assumption is that at all times the machine invariant being true implies that the user invariant is true at that time. The second assumption asserts a connection between the two forms of goal statement. If the user has achieved their goal at some time `t` then there must have existed an earlier time `t2` at which the machine goal event occurred. The user cannot achieve the goal without the machine enabling it.

4.5 Combining the Theorems

At this point we have proved two theorems. Firstly we have proved that a machine-centred statement of a design rule implies a user-centred one, and secondly that the user-centred design rule implies that post-completion errors are not made by the cognitive architecture. These two theorems can be combined giving us a theorem that justifies the correctness of the machine-centred design rule with respect to the occurrence of post-completion errors as illustrated in Figure 2. The theorem proved in HOL is:

$$
\begin{array}{l}
\vdash (\forall t.\ \text{minvariant t} \supset \text{invariant ustate t}) \land \\
\quad (\forall t.\ (\text{goalachieved t}) \supset \exists t2.\ t2 \leq t \land (\text{goalevent t2})) \supset \\
\qquad \text{MACHINE\_PCE\_DR goalevent minvariant te} \land \\
\qquad \text{USER} \ldots \text{ goalachieved invariant probes ustate mstate} \supset \\
\qquad\quad \neg(\text{PCE\_OCCURS probes (invariant ustate) te})
\end{array}
$$

This is a generic correctness theorem that is independent of the task or any particular machine. It states that under the assumptions that link the machine invariant to the user interaction invariant and the user goal to the machine goal action, the machine specific design rule is "correct". By correct in this context we mean that if any device whose behaviour satisfies the device specification is used as part of an interactive system with a user behaving according to the principles of cognition as formalised, then no post-completion errors will be made. This is despite the fact that the principles of cognition themselves do not exclude the possibility of post-completion errors.

5 Integration with full system verification

Our aim has been to verify a usability design rule in a way that integrates with formal hardware verification. The verification of the design rule needs to consider user behaviour. However, hardware designers and verifiers do not want to be concerned with cognitive models. Our aim has been therefore to separate these distinct interests so that they can be dealt with independently, but within a common framework.

There are several ways the design rule correctness theorem could be used. The most lightweight is to treat the verification of the design rule as a justification of its use in a variety of situations with no further formal reasoning, just an informal argument that any particular device design does match the design rule as specified. Its formal statement then would give a precise statement, including assumptions in the theorem, of what was meant by the design rule. Slightly more formally, the formal statement of the design rule could be instantiated with the details of a particular device. This would give a precise statement about that device. The instantiated design rule correctness theorem then is a specific statement about the absence of user error.

Instantiation involves specifying a user and machine state with entries for each action, the user's goal, interaction invariant, etc. For example, for a vending machine, the goal might simply be specified as `UserHasChoc`, an accessor to the first entry in the user state, say. The goal event from the machine perspective would be a machine state accessor `GiveChoc`. A further part of the instantiation would be to specify that the invariant was that the value of the user's possessions (money and chocolate) was at least as high as at the start. The number, and value of each possession is recorded in the user state. A relation `POSS_VAL` calculates the total value. If `possessions` is an accessor function into ustate, the invariant for a vending machine is then

$$(\text{POSS\_VAL possessions ustate t} \geq \text{POSS\_VAL possessions ustate 1})$$

Taking this approach, the final instantiated design rule theorem refers to the specific goals, actions and invariant of the case in point.

A more heavyweight use of the design rule correctness theorem would be to formally verify that the device specification of interest implies such an instantiated design rule. Suppose the device specification for a given vending machine is `VENDING_SPEC mstate`, the goal is given by `GiveChoc` and machine-based invariant by `VND_MINV` then we would prove a theorem of the form:

$$\text{VENDING\_SPEC mstate} \supset$$
$$\qquad \text{MACHINE\_PCE\_DR (GiveChoc mstate) (VND\_MINV mstate) te}$$

This theorem and its proof only needs to refer to the device specification not the user specification precisely because of the use of a machine-centred version of the design rule. It is independent of the user model and user state.

This theorem can be trivially combined with the design rule correctness statement. This gives a formal result not just that the specification meets the design rule but that in interacting with it a user would not make post-completion errors. For example, if `VND_INV` is the user-centred version of the invariant, `HasChoc` the user-centred version of the goal and `Prbs` accesses the probes from the user state we get an instantiated theorem:

$$\begin{array}{l}
(\forall \text{t. VND\_MINV mstate t} \supset \text{VND\_INV ustate t}) \land \\
(\forall \text{t. (HasChoc ustate t)} \supset \exists \text{t2. t2} \leq \text{t} \land \text{(GiveChoc mstate t2))} \land \\
\text{USER} \ldots \text{ (HasChoc ustate) (VND\_INV ustate) (Prbs ustate) ustate mstate} \land \\
\text{VENDING\_SPEC mstate} \supset \\
\qquad\qquad \neg\text{(PCE\_OCCURS (Prbs ustate) (VND\_INV ustate) te)}
\end{array}$$

Ideally the two assumptions linking the two formalisations of the invariant and the two formalisations of the goal would be discharged. This is the only part of the proof that requires reasoning about the user model. We have isolated it from the verification of the specification meeting its requirements. We obtain a theorem that the user model, using a vending machine that meets the specification, will not make post-completion errors.

$$\begin{array}{l}
\text{USER} \ldots \text{ (HasChoc ustate) (VND\_INV ustate) (Prbs ustate) ustate mstate} \land \\
\text{VENDING\_SPEC mstate} \supset \\
\qquad\qquad \neg\text{(PCE\_OCCURS (Prbs ustate) (VND\_INV ustate) te)}
\end{array}$$

As the verification framework we have used was originally developed for hardware verification, it would then be simple to combine this result with a hardware verification result stating that the implementation of the device implied its behavioural specification. Suppose we had proved the hardware verification result:

$$\forall \text{mstate. VENDING\_IMPL mstate} \supset \text{VENDING\_SPEC mstate}$$

where `VENDING_IMPL` is a structural specification giving an implementation of the vending machine. We obtain immediately a theorem stating that the *implementation* of the vending machine does not lead to post-completion errors occurring:

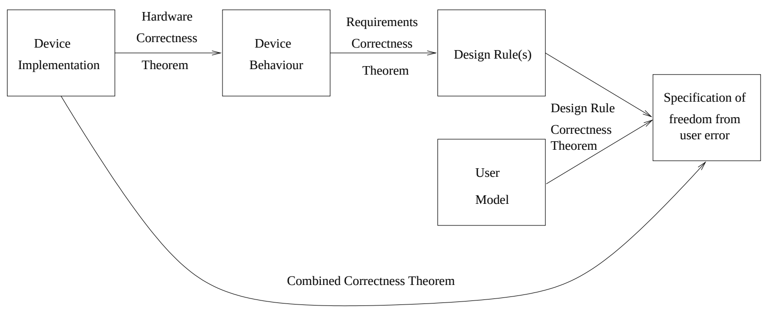
Device Implementation
Hardware Correctness Theorem
Device Behaviour
Requirements Correctness Theorem
Design Rule(s)
Design Rule Correctness Theorem
Specification of freedom from user error
User Model
Combined Correctness Theorem

Fig. 3. Combining separate system correctness statements

USER ... (HasChoc ustate) (VND_INV ustate) (Prbs ustate) ustate mstate ∧
VENDING_IMPL mstate ⊃
    ¬(PCE_OCCURS (Prbs ustate) (VND_INV ustate) te)

The design rule correctness theorem can thus be combined with a result that a particular device specification meets the design rule. By further combining it with a result that a particular implementation of the device meets the specification we obtain a theorem that the *implementation* does not result in post-completion errors occurring as is illustrated in Figure 3.

The hardware verification is done independently of the cognitive model and explicit usability concerns but then combined with theorems that use them. In previous work [10] [26] we demonstrated how hardware verification correctness theorems could be similarly chained with a full usability task completion correctness theorem stating that when the cognitive model was placed with the behavioural specification of the device, the combined behaviour of the resulting system was such that the task was guaranteed to be completed. The difference here is that the end usability statement being chained to is about the absence of a class of errors rather than task completion; however, the general approach is similar.

6 Conclusions

We have shown how a usability design rule can be verified and the result combined with analysis of other aspects of a design. We started by outlining a set of principles of cognition specifying cognitively plausible behaviour. These principles are based on results from the cognitive science and human-computer interaction literature. From these principles we developed a formal cognitive architecture. This architecture does not directly model erroneous behaviour. Erroneous behaviour emerges if it is placed in an environment (i.e. with a computer system) that allows it.

We then formally specified a class of errors known as post-completion errors. We also specified two versions of a design rule claimed to prevent post-completion errors. The first is specified in terms of user goals and invariant. The second is in terms of machine events, and so of more direct use to a designer. We proved a theorem that the user-centred design rule is sufficient to prevent the cognitive architecture from committing post-completion errors. This theorem is used to derive a theorem that the machine-based formulation is also sufficient. The resulting theorem is a correctness theorem justifying the design rule. It says that users behaving according to the principles of cognition will not make post-completion errors interacting with a device that satisfies the design rule.

The definitions and theorems are generic and do not commit to any specific task or machine. They are a justification of the design rule in general rather than in any specific case. They can be instantiated to obtain theorems about specific scenarios and then further with specific computer systems.

This work demonstrates an approach that integrates machine-centred verification (hardware verification) with user-centred verification (that user errors are eliminated). The higher-order logic framework adopted is that developed for hardware verification. Specifications, whether of implementations, behaviours or design rules, are higher-order logic relations over signals specifying input or output traces. The theorems developed therefore integrate directly with hardware verification theorems about the computer component of the system. The user based parts of the proof have been isolated from the machine based parts. The theorem developed here, once instantiated for a particular device can be combined with correctness theorems about that device to obtain a theorem stating that the machine implementation implies that no post-completion errors can occur. This requires a proof of a linking theorem that the device specification satisfied the machine-centred design rule.

The work presented here builds on our previous work on fully formal proofs that an interactive system completes a task [11]. A problem with that approach is that with complex systems, guarantees of task completion may be unobtainable. The current approach allows the most important errors for a given application to be focussed on.

7 Further Work

We have only considered one class of error and a simple design rule that prevents it occurring. In doing so we have shown the feasibility of the approach. There are many other classes of error. Others that are potential consequences of our principles of cognition are discussed in [12]. Further work is needed to formally model those error classes and design rules, and verify them formally following the approach developed here. This will also allow us to reason about the scope of different design rules especially those that apparently contradict.

In this paper we have been concerned with the verification of design rules in general, rather than their use in specific cases. We have argued, however that, since the framework used is that developed for hardware verification, integration of instantiated versions of the design rule correctness theorem is straightforward. Major case studies are needed to demonstrate the utility of this approach.

Our architecture is intended to demonstrate the principles of the approach and covers only a small subset of cognitively plausible behaviour. As we develop it, it will give a more accurate description of what is cognitively plausible. We intend to extend it in a variety of ways. As this is done, more erroneous behaviour will be possible. We have essentially made predictions about the effects of following design rules. In broad scope these are well known and based on usability experiments. However, one of our arguments is that more detailed predictions can be made about the scope of the design rules. The predictions resulting from the model could be used as the basis for designing further experiments to validate the model and the correctness theorems proved, or further refine it. We also suggested there are tasks where it might be impossible to produce a design that satisfies all the underlying principles, so that some may need to be sacrificed in particular situations. We intend to explore this issue further.

Acknowledgements We are grateful to Kirsten Winter and the anonymous referees whose comments have helped greatly improve this paper.

References

1. R. Back, A. Mikhajlova, and J. von Wright. Modeling component environments and interactive programs using iterative choice. Technical Report 200, Turku Centre for Computer Science, sep 1998.
2. A. E. Blandford and R.M. Young. The role of communication goals in interaction. In *Adjunct Proceedings of HCI'98*, pages 14–15, 1998.
3. A.E. Blandford, P.J. Barnard, and M.D. Harrison. Using interaction framework to guide the design of interactive systems. *International Journal of Human Computer Studies*, 43:101–130, 1995.
4. A.E. Blandford, R. Butterworth, and P. Curzon. PUMA footprints: linking theory and craftskill in usability evaluation. In *Proceedings of Interact*, pages 577–584, 2001.
5. P. Bumbulis, P.S.C. Alencar, D.D. Cowen, and C.J.P. Lucena. Validating properties of component-based graphical user interfaces. In F. Bodart and J. van der Donckt, editors, *Proc. Design, Specification and Verification of Interactive Systems '96*, pages 347–365. Springer, 1996.
6. R. Butterworth, A.E. Blandford, and D. Duke. Using formal models to explore display based usability issues. *Journal of Visual Languages and Computing*, 10:455–479, 1999.
7. R. Butterworth, A.E. Blandford, and D. Duke. Demonstrating the cognitive plausibility of interactive systems. *Formal Aspects of Computing*, 12:237–259, 2000.
8. M. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.
9. J.C. Campos and M.D. 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.
10. P. Curzon and A.E. Blandford. Using a verification system to reason about post-completion errors. Presented at Design, Specification and Verification of Interactive Systems 2000. Available from http://www.cs.mdx.ac.uk/puma/ as working paper WP31.

11. P. Curzon and A.E. Blandford. Detecting multiple classes of user errors. In Reed Little and Laurence Nigay, editors, *Proceedings of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)*, volume 2254 of *Lecture Notes in Computer Science*, pages 57–71. Springer-Verlag, 2001.
12. P. Curzon and A.E. Blandford. From a formal user model to design rules. In P. Forbrig, B. Urban, J. Vanderdonckt, and Q. Limbourg, editors, *Interactive Systems. Design, Specification and Verification, 9th International Workshop*, volume 2545 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2002.
13. D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–394, 1998.
14. R.E. Fields. Analysis of erroneous actions in the design of critical systems. Technical Report YCST 20001/09, University of York, Department of Computer Science, 2001. D.Phil Thesis.
15. M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.
16. W. Gray, R.M. Young, and S. Kirschenbaum. Introduction to this special issue on cognitive architectures and human-computer interaction. *Human-Computer Interaction*, 12:301–309, 1997.
17. W.D. Gray. The nature and processing of errors in interactive behavior. *Cognitive Science*, 24(2):205–248, 2000.
18. E. Hollnagel. *Cognitive Reliability and Error Analysis Method.* Elsevier, 1998.
19. D.E. Kieras, S.D. Wood, and D.E. Meyer. Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Trans. Computer-Human Interaction*, 4(3):230–275, 1997.
20. D. Leadbetter, P. Lindsey, A. Hussey, A. Neal, and M. Humphreys. Towards model based prediction of human error rates in interactive systems. In *Australian Comp. Sci. Communications: Australasian User Interface Conf.*, volume 23(5), pages 42–49, 2001.
21. T.G. 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. Wien : Springer, 1995.
22. F. Paterno' and M. Mezzanotte. Formal analysis of user and system interactions in the CERD case study. In *Proceedings of EHCI'95: IFIP Working Conference on Engineering for Human-Computer Interaction*, pages 213–226. Chapman and Hall Publisher, 1995.
23. F.E. Ritter and R.M. Young. Embodied models as simulated users: introduction to this special issue on using cognitive models to improve interface design. *Int. J. Human-Computer Studies*, 55:1–14, 2001.
24. C. R. Roast. Modelling unwarranted commitment in information artifacts. In S. Chatty and P. Dewan, editors, *Engineering for Human-Computer Interaction*, pages 77–90. Kluwer Academic Press, 1998.
25. J. Rushby. Using model checking to help discover mode confusions and other automation suprises. In *3rd Workshop on Human Error, Safety and System Development (HESSD'99)*, 1999.
26. H. Xiong, P. Curzon, S. Tahar, and A. Blandford. Formally linking MDG and HOL based on a verified MDG system. In M. Butler, L. Petre, and K. Sere, editors, *Proc. of the 3rd International Conference on Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 205–224, 2002.