ISSN 1470-5559

Justifying Usability Design Rules Based on a Formal Cognitive Model

Paul Curzon and Ann Blandford

RR-05-07

December 2005

Department of Computer Science

Justifying Usability Design Rules Based on a Formal Cognitive Model

Paul Curzon[1] and Ann Blandford[2]

[1] Department of Computer Science, Queen Mary, University of London,
Email: pc@dcs.qmul.ac.uk,
[2]University College London Interaction Centre,
Email: a.blandford@ucl.ac.uk

Abstract

Interactive systems combine a human operator and computer. Either may be a source of error. Verification processes used must ensure the correctness of the computer, and minimize the possibility of human error. In human-centered design allowance is made for human frailty. One such approach is to follow design rules. Design rules, however, are often ad hoc. We examine how a formal cognitive architecture, encapsulating results from the cognitive sciences, can be used to justify design rules both semi-formally and formally. The cognitive architecture was developed by formalising cognitively plausible behaviour, based on results from cognitive psychology.

Keywords: Design rules, Formal verification, Human error, Formal cognitive architecture, Human-centered design, HUM.

1 Introduction

Interactive computer systems are systems that combine a human operator with a computer system. Such systems need to be both correct and usable. With the increasing ubiquity of interactive computer systems, usability becomes increasingly important because minor usability problems can scale to having major economic and social consequences. Usability has many aspects. In this paper, we focus 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. Changes to a design can also eradicate or at least reduce the likelihood of systematic human error occuring. It is therefore potentially within the scope of verification methodologies.

Human-centred design aims to reduce user error and other usability problems by designing systems in a way that makes allowance for human frailty. Techniques are needed for the evaluation of systems to determine whether they are so designed. A popular approach is to adhere to informal lists of design rules (also referred to as design principles and usability principles) that can then also be used as the basis for review based evaluation. Well known examples of general design / evaluation rules include Nielsen's Heuristics [32], Shneiderman's Golden Rules [42] and the Principles of Dix *et al* [16]. Such lists normally either give a small number of high level heuristics, leaving much to the craft skill of the designer or evaluator, or they consist of many more specific rules, for example tied to a particular graphical user interface style.

Some sets of design rules focus on particular aspects of design. For example, those of Leveson *et al* [27] focus specifically on avoiding mode related human error. They developed approximately 50 design criteria known to reduce the likelihood of human error; for example, indirect mode transitions (transitions that are not the result of operator action) should be eliminated where possible.

There is much overlap in the different rule sets. For example one of Nielsen's Heuristics [32], related to rules we consider, concerns the visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time. Similarly Shneiderman has a golden rule on offering informative feedback: For every operator action, there should be some system feedback. For frequent and minor actions the response can be modest, while for infrequent and major actions the response should be more substantial.

Nielsen's Heuristics are based on a combination of expert knowledge and an empirical approach to refine the rules. Factor analysis of 249 usability problems was used to improve the explanatory power of an early version of the rules. Similarly Shneiderman's rules are obtained reflectively from experience. This contrasts with Dix's Principles for usability [16], where the authors highlight the fact that the rules are theoretically derived from psychological, computational and sociological knowledge. Our approach in this paper follows the tradition of Dix, working from formal models.

Design rules, when used in the evaluation/verification stage, are generally applied informally, relying on the expert evaluator's expertise. Formal verification techniques have also been suggested as a way to verify that individual systems meet properties corresponding to design rules (see for example [7]). However, formal verification, even using automated systems, can be time consuming and requires mathematical skill that usability analysts may not possess. It therefore may not always be practical within the design cycle. Formal techniques can, however, be used in lightweight ways outside the design cycle to address problems with design rules. For example, formal specification of them and/or the contexts in which they are applicable may be useful. Leveson *et al* [27] note, for example, that whilst most of their rules were based on experience, some were derived from completeness aspects of a formal state machine model of existing criteria.

One potential problem with lists of design rules, especially where based on the experience of HCI experts, is that they can be ad hoc and are sometimes apparently contradictory. The contexts in which they apply may not always be clear. Formal specifications have the potential to unravel such contradictions [17]. A further use of formalized design rules is that they can be verified against claims about their purpose. That is: their use can be justified. These are the concerns of this paper.

1.1 Contribution

The main contribution of this paper is to show how (well-known) usability design rules, designed to prevent systematic user error, can be justified in terms of a formal specification of aspects of human cognition (or "cognitive architecture"). More specifically:

- We show how principles of cognition based on results from the cognitive sciences can form the basis of a formal cognitive architecture that is amenable to formal proof.
- We show how a variety of realistic potential erroneous actions emerge from the behaviour specified by a very simple cognitive architecture using poorly designed systems but do not occur with other alternatively designed systems.
- We derive design rules from the architecture. Our contribution is not the rules themselves, but rather the demonstration that they can be justified from a formalisation of a small set

of principles. We use the fact that the rules are grounded in a formal model of cognition to informally explore the scope of application of the rules. This part of the approach is semi-formal: we use informal high-level argument about a fully formal model, as opposed to the explicit application of inference rules of the underlying logic as in the next point.

- We then go on to show how design rule correctness theorems can be stated and fully formally proved with respect to a cognitive architecture, using as an illustrative example a design rule for eliminating one particular class of error.

1.2 Overview of Paper

We first outline related work in Section 2 to give the context of this work with respect to other formal approaches to HCI. We then define, in Section 3, 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 demonstrate the approach. They cover a variety of classes of cognitive behaviour of interest related to 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.

After outlining the higher-order notation used in the paper in Section 4, we describe a generic **formal cognitive model** of these principles in higher-order logic in Section 5. By "generic" we mean that it can be targeted to different tasks and interactive systems. Strictly this makes it a cognitive architecture [21]. In this 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. Combining the principles of cognition into a single architecture rather than formalising them separately allows reasoning about their interaction, and about how multiple minor errors might interact.

In Section 6, we show how cognitively plausible behaviour, whilst often also leading to appropriate user action, can in specific circumstances be considered as resulting in *erroneous actions.* We discuss the circumstances in which such erroneous actions can result by reasoning from the formal cognitive architecture. In particular, we relate them to Hollnagel's error phenotypes [23] as presented by Fields [19]: a classification scheme for erroneous action. In doing so, we identify cognitively plausible ways in which the erroneous actions could arise. We show that a wide variety are covered from even a minimal formal definition of cognitively plausible behaviour, demonstrating the generality of the approach.

In Section 7, we show how we can semi-formally derive **design rules** from the formal cognitive architecture that, if followed, ensure that the erroneous actions identified will not be made for the specific cognitive reasons embodied by the principles of cognition. Even though the cognitive architecture is capable of making the errors, the design rules ensure that the environments in which they would emerge do not occur. Similar erroneous actions with other cognitive causes are, of course, still possible.

In Section 8, we describe an approach to formally verifying usability design rules based on the formal cognitive architecture. To illustrate the approach, we consider 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 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. We reformulate it in a machine-centred way, based on machine rather than user concepts and actions. This is of more direct use to a designer. We ultimately prove that a machine-centred version of the design rule implies the absence of the class of error considered.

2 Formal Reasoning about Usability

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 [11] [29]. Most commonly such a specification is used with model-checking-based verification; investigations include whether a given event can occur or whether properties hold of all states. Another formal approach is exemplified by the work of Bumbulis *et al* [6], who verified properties of interfaces based on a guarded command language embedded in the HOL theorem proving system. Back *et al* [1] illustrate how properties can be proved and data refinement performed on 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 specific to an individual device, 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. For example, Duke *et al* [18] 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 [30] 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 [34] 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 [9]. Butterworth *et al* [8] do take this into account, using TLA to reason about reachability conditions within an interaction.

Rushby *et al* [40, 12, 37, 38, 39], like us, focus on user errors, though in their case just on one important aspect of human limitation. In particular they formalise plausible mental models of systems, looking for discrepancies between these and actual system behaviour using Murϕ. They are specifically concerned with the problem of mode errors and the ability of pilots to track mode changes. The mental models are simplified system models rather than cognitive models. Various ways of developing them are suggested including an empirical approach based on questioning pilots about their beliefs about the system [12]. Like interface-oriented approaches, each model is individually hand-crafted for each new device in this work. It is concerned with the knowledge level and is complementary to our work.

Bredereke and Lankenau [5] [25] extended Rushby *et al*'s basic approach [39], using CSP. Whereas in Rushby's original work the system model and mental model are intertwined in a single set of definitions, in CSP they can be described as separate processes. Furthermore, Bredereke and Lankenau include a relation from environment events to mental events that could in principle be lossy, corresponding to physical or psychological reasons for an operator not observing all interface events of a system. However they note that in practice in their work the relation does no more than renaming of events and so is not lossy. This contrasts with our work where we explicitly consider the operator's imperfections.

An approach to interactive system verification that focuses directly on a range of errors is exemplified by Fields [19]. 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.

Approaches that are based on a cognitive architecture (e.g. [24][22][35]) model underlying cognitive causes of errors. However, the modeling exemplified by these approaches is too detailed to be amenable to formal proof. Our previous work [14] followed this line but at a coarser level of detail, making formal proof tractable. General mechanisms of cognition are modeled 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.

Our previous work explored how the cognitive architecture considered here could be used to analyse interactive system designs by treating it as a component of that system with a fixed design [14, 15], proving that a specific task will always be completed eventually. This was done using the interactive proof system, HOL [20]. In the work we describe here, we use the cognitive architecture as the basis of reasoning about interactive systems in general. The process of doing so also acts, in part, to validate the architecture for formal verification. Our approach is similar to that of [4] on the PUMA project in that we are working from a (albeit different and more formal) model of user behaviour to high level guidance. There the emphasis is on a semi-formal basis underpinning the craft skill in spotting when a design has usability problems. In contrast, we are concerned 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.

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 is therefore to just formalise the design rules. This idea dates to some of the earliest work on formal methods in Human-Computer Interaction. Thimbleby [44] and Dix and Runciman [17], for example, give formal statements of design principles, discussing their use and showing how one can explore how they interact or conflict. Dix and Runciman developed an abstract model of a computer system around which the definitions were based. More recent work in this vein includes that of Blandford *at al* [3] and Roast [36]. If rules are formalised, it becomes possible to formally check individual systems against them using automated verification tools. For example, Butler *et al* [7] formalise some of the rules of Leveson *et al* [27] in PVS and show how the results of failed proof attempts identify potential system problems that could lead to user error. Lüttgen and Carreno [28] compare the strengths and weaknesses of a range of model checking tools with theorem provers for detecting mode confusion problems in this way. Our work by contrast is concerned with *justifying* the formalisation of design rules based on underlying principles about cognition embodied in a formal cognitive architecture. We consider how design rules can be proved to be correct, up to the assumptions of the formalisation of the principles of cognition. This gives extra assurance to those applying the design rules over just formalisation.

The work described in the present paper is part of the EPSRC funded HUM project which aims to develop empirically validated formally-based verification tools for interactive systems with a particular focus on human error. The University of Queensland's safeHCI project [26] 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.

3 Principles of Cognition

Our principles of cognition, and more formally the cognitive architecture, specify **cognitively plausible behaviour** (see [9]). That is, they specify possible traces of user actions that can be justified in terms of specific results from the cognitive sciences. Of course users might also act outside this behaviour, about which situations the architecture 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 principles give a *knowledge level* description in the terms of Newell [31]. We do not attempt to model underlying neural architecture nor the higher level cognitive architecture such as working memory units. Instead our model is that of an abstract specification, intended for ease of reasoning. The focus of the principles is in terms of 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.

In this section we describe the principles of the current cognitive architecture informally. We discuss them in greater detail in Section 5 when we describe the formalisation. We use railway ticket vending machines to illustrate the points. They are ubiquitous and are increasingly replacing manned ticket offices. However, existing designs continue to exhibit design problems that encourage user error [46].

We now outline the principles currently embodied in our cognitive architecture:

Timing of Actions: The time taken to take an action after it becomes plausible is not predictable. For example, once an instruction to make a choice of destination becomes available on a ticket machine, it is not predictable exactly how long a person will take to press the button corresponding to their destination choice.

Non-determinism: In any situation, any one of several behaviours that are cognitively plausible at that point might be taken. The separate behaviours form a series of options, any of which could be taken. It cannot be assumed in general that any specific behaviour that is plausible at a given time will be the one that a person will follow. For example, faced with an apparent choice of selecting a destination and inserting money, both of which are needed to achieve the goal of buying a ticket, either could be taken by the person.

No-option-based termination behaviour: If there is no apparent action that a person can take that will help her complete her task then the person may terminate the interaction. 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. We do not concern ourselves with precisely what behaviour constitutes terminating an interaction. It could mean giving up completely, or attempting to restart the interaction from the beginnning, for example. The scope of concern here is only up to the point where the interaction is terminated. Note that a possible action that a person could take is to wait. However, they will only do so given some explicit reason according to one of the other principles. For example, a ticket machine might display a message "Please Wait". If they see it, the person would have a cognitively plausible option: to wait. If there were no such message and no other reason to suggest that waiting indefinitely would lead to the person achieving the task, they could terminate the interaction.

Task-based termination behaviour: When the task the user set out to complete is completed we take the interaction to be terminated. This is how we define successful completion of the interaction. We assume the person enters an interaction with a goal to achieve. Task completion could be more than just goal completion, however. 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. Examples of such tasks with respect to a ticket machine include taking back a credit card or taking change. Other actions could be taken after the task is completed, but for the purposes of our analysis we assume that they form part of a subsequent interaction. In doing this we rule out of consideration for our current cognitive architecture situations where a device confuses a user into believing the task is not completed. This could be considered in future extensions to the work described here.

Goal-based termination behaviour: Cognitive psychology studies have shown that users intermittently, but persistently, terminate interactions as soon as their goal has been achieved [10]. This could occur even if the task, in the sense above, is not fully achieved. With a ticket machine this may correspond to the person walking away, starting a new interaction (perhaps by hitting a reset button), *etc.* This principle does not have to lead to erroneous actions: that depends on the environment.

Reactive behaviour: A user may react to an external stimulus or message, doing the action suggested by the stimulus. For example, if a flashing light comes on next to the coin slot of a ticket vending machine, a user might, if the light is noticed, react by inserting coins if it appears to help the user achieve their goal.

Communication goal behaviour: A user enters an interaction with knowledge of the task and in particular task dependent sub-goals that must be discharged – in particular, information that must be communicated to the device or items (such as coins) that must be inserted into the device. Given the opportunity, they may attempt to discharge any such *communication goals* [2]. The precise nature of the action associated with the communication goal may not be known in advance. A communication goal specification is a task level partial plan. It is a pre-determined plan that has arisen from knowledge of the task in hand independent of the environment in which that task will be accomplished. It is not a fully specified plan, in that no order of the corresponding actions may be specified. In the sense of [2] a communication goal is purely about information communication. Here we use the idea more generally to include other actions that are known to be necessary to complete a task. For example, when purchasing a ticket, in some way the destination and ticket type must be specified as well as payment made. The way that these must be done and their order may not be known in advance. However, a person enters an interaction with the aim of purchasing a ticket primed for these communication goals. 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. For example, if a "return ticket" button is visible then the person may press that first if that is what they see first. If a button with their destination is visible then they may press it first. Communication goals are a reason why people do not just follow instructions.

Mental versus Physical Actions: A user commits to taking an action in a way that cannot be revoked after a certain point. Once a signal has been sent from the brain to the motor system to take an action, the signal cannot be stopped even if the person becomes aware that it is wrong before the action is taken. For example, on deciding to press a button labelled with the destination "Liverpool", at the point when the decision is made the mental trigger action takes place and after a very short delay, the actual action takes place.

Relevance: A user will only take an action if there is something to suggest it corresponds to the desired effect. We do not currently model this explicitly: however, it is implicit in the way other behaviour is modelled.

4 Higher-order logic

We use higher-order logic to formalise the principles of cognition. Whilst higher-order rather than a first order logic is not essential for this, its use makes the formal specifications simpler than the use of a first-order logic would. An aim of formalisation is to make principles inspectable. Formalising them in as natural way as possible is therefore important.

We have used the HOL interactive proof system [20], a theorem prover for higher-order logic, so theorems are machine-checked. Given the relative simplicity of the theorems presented here, 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 [13]. 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.

The notation used in this paper is summarized in Table 1.

5 Formalising Cognitively Plausible Behaviour

The principles of cognition are formalised as an abstract cognitive architecture. It is specified by a higher-order logic relation `USER`, the top levels of which are given in Figure 1. The full specification is given in Appendix A. It takes as arguments information such as the user's goal, `goalachieved`, a tuple of actions that the user may take, `actions`, *etc.* The arguments of the architecture will be examined in more detail as needed in the explanation of the architecture below.

The final two arguments of the architecture, `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. Their concrete type is supplied when the architecture is instantiated for a given interaction. As in this paper we are performing general reasoning about the architecture rather than about specific interactive systems, they remain type variables. The user and machine states record over time the series of mental and physical actions made by the user, together with a record of the user's possessions and knowledge. They are instantiated to a tuple of *history functions.* A history function is 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

a $\wedge$ b	both a and b are true
a $\vee$ b	either a is true or b is true
a $\supset$ b	a is true implies b is true
$\forall$n. P(n)	for all n, property P is true of n
$\exists$n. P(n)	there exists an n for which property P is true of n
f n	the result of applying function f to argument n
a = b	a equals b
IF a THEN b ELSE c	if a is true then b is true, otherwise c
:num	the type of natural numbers (also used to represent time)
:bool	the type of booleans
:a$\rightarrow$ b	the type of functions from type a to type b
:a#b	the type pairing elements of type a with elements of type b
:'m	has type, the polymorphic type variable
(a, b)	the pair with first element a and second element b
FST p	the first element of pair p
SND p	the second element of pair p
[ ]	the empty list
x :: l	cons: the list consisting of first element x and remaining list l
l1 APPEND l2	the list consisting of list l1 appended onto list l2
EL n l	the nth element of list l
MAP f l	apply f to each element of list l
$\vdash$*thm* P	P is a theorem proved in HOL
$\vdash$*def* P	P is a definition

Table 1: Higher-order Logic notation

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

⊢def USER_CHOICE flag actions commitments commgoals stimulus_actions
            finished goalachieved invariant ustate mstate =
     (∀t.
     ¬(flag t) ∨
     (IF (finished ustate (t-1))
      THEN (NEXT flag actions FINISHED 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 FINISHED t)
      ELSE USER_RULES flag actions commgoals stimulus_actions
            goalachieved mstate ustate t))

⊢def USER_RULES flag actions commgoals stimulus_actions goalachieved mstate ustate t =
     COMPLETION flag actions goalachieved ustate t ∨
     REACTS flag actions stimulus_actions t ∨
     COMMGOALER flag actions commgoals goalachieved ustate mstate t ∨
     ABORTION flag actions goalachieved commgoals stimulus_actions ustate mstate t
```

Figure 1: The USER relation (slightly simplified)

that for each time instance indicates whether the cognitive architecture has terminated the interaction.

For the purpose of exposition, in Figure 1 and the immediate discussion, we simplify the definition, omitting the idea of 'probes'. As discussed below, probes do not model cognition as such, but give a window into the architecture's working. The extension to the cognitive architecture to include them is described in Section 5.9 and the full architecture is given in Appendix A.

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 cognitive architecture 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
```

5.1 Non-determinism

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`. This is where the principle of non-determinism is modelled. Note the use of disjunction in definition `USER_RULES` in Figure 1. Each rule describes an action that a user *could* plausibly make. The rules are grouped corresponding to a user performing actions for specific cognitively related reasons. For example, `REACTS` in Figure 1 groups all reactive behaviour. 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. 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.

5.2 Timing of Actions

The architecture is based on a temporal primitive, `NEXT`, that specifies the next user action taken *after* a given time. For example,

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 architecture. The first line of `USER_CHOICE` in Figure 1, `¬(flag t)`, 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`.

NEXT flag actions FINISHED t

The `action` argument of `NEXT` is instantiated to `FINISHED`. 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.

A detail about the formalisation that is needed to understand the definitions is that the action argument of `NEXT` is actually represented as a position into the action list. For example `FINISHED` gives the position in the list `actions` of the history function, `finished`, that records over time whether the interaction is terminated. Whilst other actions in the cognitive architecture are defined at instantiation time, a finished signal is required for all cognitive models. FINISHED is therefore defined as a constant, position 0, in the list. The positions of other signals are defined when the cognitive architecture is instantiated rather than in the cognitive architecture.

5.3 Mental versus physical actions

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`, whether any mental action `maction`, from the list supplied was taken in the previous time instance (`t-1`):

$\vdash_{def}$ (CommitmentMade [ ] t = FALSE) $\wedge$
    (CommitmentMade (maction :: rest) t =
        (maction(t-1) = TRUE) $\vee$ (CommitmentMade rest t))

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

$\vdash_{def}$ COMMIT flag actions maction paction t =
    (maction (t-1) = TRUE) $\wedge$ NEXT flag actions paction t

$\vdash_{def}$ (COMMITS flag actions [ ] t = FALSE) $\wedge$
  (COMMITS flag actions (ca :: commits_actions) t =
    ((COMMITS flag actions commits_actions t) $\vee$
     (COMMIT flag actions (CommitmentGuard ca) (CommitmentAction ca) t)))

In `COMMITS`, `CommitmentGuard` extracts the guard of a commitment pair, `ca`: that is the mental action. `CommitmentAction` returns the physical action that corresponds to that mental action.

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 over-riding priority as modelled by being in the if-then-else staircase:

    ELSE IF (CommitmentMade (CommitmentGuards commitments) t)
        THEN (COMMITS flag actions commitments t)

5.4 Task-based termination behaviour

The third if statement of definition `USER_CHOICE` specifies that a user will terminate an interaction when their whole task is achieved.

    ELSE IF TASK_DONE (goalachieved ustate) (invariant ustate) t
        THEN (NEXT flag actions finishedpos t)

`TASK_DONE` asserts whether the task is completed at a given time or not. It requires arguments about the goal and an invariant. We discuss these in turn below.

We are concerned with goal-based interactions. The user enters the interaction with 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 this goal is achieved or not. This history function defines what the goal is for the interaction under consideration. With a ticket machine, for example, this may correspond to the person's possessions including the ticket. Like `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 argument and can as such be instantiated with an arbitrarily complex condition. It might, for example, be that the user has a particular object such as a ticket, 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 completion of these subsidiary tasks could be modelled in several ways including modelling the tasks explicitly. We use a general approach based on the concept of an *interaction invariant* [14]. The underlying reason for these subsidiary 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: the interaction invariant that held at the start must be restored. 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 ticket machine might be true when the total value of the user's possessions (coins and ticket) have been restored to their original value, the user having exchanged coins for tickets of the same value. Task completion involves not only completing the user's goal, but also restoring the invariant. The invariant is a higher-order argument to the cognitive architecture. It is modelled as a history function, indicating over time when the invariant is and is not achieved.

We can thus define task completion formally in terms of goal completion and invariant restoration. The task is completed at a time when both the goal and invariant history functions are true.

$\vdash$*def* TASK_DONE goal inv t = (goal t $\wedge$ inv t)

Note that `TASK_DONE goalachieved invariant` is also a history function - given a time it indicates the times when the task is and is not completed in terms of the goal and invariant's history functions.

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. It is not treated as being a non-deterministically chosen action.

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

5.5 Goal-based Completion

As already noted, achieving the goal alone is often the trigger that leads to a person terminating an interaction [10]. This behaviour is formalised as a `NEXT` rule. If the goal is achieved at a time then the next action of the cognitive architecture can be to terminate the interaction:

$\vdash$*def* COMPLETION flag actions finished goalachieved ustate t =
    (goalachieved ustate t) $\wedge$ NEXT flag actions finished t

As this is combined disjunctively with other rules, its presence does not assert that it will be the action taken, just that according to the cognitive architecture there are cognitively plausible traces where it occurs.

5.6 Reactive Behaviour

We model a user reacting to a stimulus from a device, doing the action suggested by it, generically. In a given interaction there may be many different stimuli to react to. Relation `REACT` gives the general rule defining what it means to react to a given stimulus. If at time `t`, the stimulus `stimulus` is active, the next action taken by the user out of possible actions, `actions`, at an unspecified later time, may be the associated `action`. As with the goal and invariant already discussed, the stimulus argument is a higher order function that can be instantiated with an arbitrarily complex condition over time.

$\vdash$*def* REACT flag actions stimulus action t =
    stimulus t $\wedge$ NEXT flag actions action t

As there may be many reactive signals, the cognitive architecture 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. `Stimulus` and `Action` extract a pair's components.

$\vdash_{def}$ (REACTS flag actions [ ] t = FALSE) $\wedge$
(REACTS flag actions (s :: st) t =
((REACTS flag actions st t) $\vee$ (REACT flag actions (Stimulus s) (Action s) t)))

5.7 Communication Goals

We model communication goals (task dependent sub-goals a person has knowledge of) as a list of (guard, action) pairs, one for each communication goal, in a similar way to 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.

5.8 No-option-based termination behaviour

The cognitive architecture includes a final default non-deterministic rule, `ABORTION`, that models the case where a person can not see any plausible action to help them towards their goal. It just forms the negation of the guards of all the other rules, based on the same arguments.

5.9 Probes

The features of the cognitive architecture discussed above concern aspects of cognition. One recent extension of the architecture as compared to previous work [14] 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 in the current model 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:

(Goalcompletion probes t) $\wedge$ goalachieved t $\wedge$ 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 is actually:

$\vdash_{def}$ REACT flag actions stimulus action probes t =
stimulus t $\wedge$ (Goalcompletion probes t = FALSE) $\wedge$ 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. In future work we intend to investigate formally other design rules. This will require filling out the cognitive architecture with probes for each rule. It is not done here for simplicity whilst demonstrating the ideas.

6 The Erroneous Actions that Emerge

In the previous sections we have described a formal model of cognitive behaviour. It does not explicitly describe erroneous behaviour. Any of the rules could correspond to both correct or erroneous behaviour. Error emerges from the interaction between the architecture and a particular context.

Erroneous actions are the proximate cause of failure attributed to human error in the sense that it was a particular action (or inaction) that immediately caused the problem: a user pressing a button at the wrong time, for example. However, to understand the problem, and so ensure it does not happen again, approaches that consider the proximate 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 sensible. Hollnagel [23] distinguishes between human error `phenotypes` (classes of erroneous actions) and `genotypes` (the underlying, for example psychological, cause). He identifies a range of simple phenotypes. These are single deviations from required behaviour. Fields [19] enumerates and gives formal definitions of those as repetition of an action, reversing the order of actions, omission of actions, late actions, early actions, replacement of one action by another, insertion of an additional action from elsewhere in the task, and intrusion of an additional action unrelated to the task.

In practical designs it is generally infeasible to make all erroneous actions impossible. A more appropriate aim is therefore to ensure that *cognitively plausible* erroneous actions are not made. To ensure this, it is necessary to consider the genotypes of the possible erroneous actions. We examine how our simple cognitive architecture can exhibit behaviour corresponding to these phenotype errors, thus linking them to underlying causes. We thus show, based on reasoning about the formal cognitive architecture, that, from the minimal principles we started with, a wide range of classes of erroneous actions in the form of phenotypes occur.

We now look at each simple phenotype and at the situations where they are cognitively plausible according to our architecture. We show that even with the very simple model of cognitively plausible behaviour a wide range of error classes is possible. We do not claim to model *all* cognitively plausible phenotypical erroneous actions. There are other ways each could occur for reasons we do not consider. However, not all errors that result from the architecture were explicitly considered when the principles were defined. The scope of the cognitive architecture in terms of erroneous actions is wider than those it was originally expected to encompass.

6.1 Repetition of actions

The first class of erroneous action considered is the repetition of an action already performed. Our cognitive architecture could repeat actions if apparently guided to do so by the device.

If the guards of a reactive rule are true due to the stimulus being present, then the rule will be active and so could lead to the action being taken, as long as the task is not completed. If the action had been taken earlier as a result of the stimulus being present then this would be a repetition. On the other hand, if the stimulus does not become true again, the action will not be repeated.

The current cognitive architecture would thus do such a repetition if reactive signals guided the action and continued to do so after the action had been completed. For example, with a ticket machine, if a light next to a coin slot continued to flash for a period after the correct money had been inserted a person might assume they had not inserted enough and insert more. Alternatively, if an initial screen included buttons to select ticket type, select destination, select departure station etc, which it returned to after each was completed without removing the completed options, then a person might try to do completed actions again thinking they had not achieved them.

Similarly an action originally performed as a communication goal could be repeated by the architecture if a reactive prompt to do so later appeared. For example, suppose a person entered an interaction knowing they needed to communicate the destination “Liverpool” and consequently pressed the corresponding button. If they were later presented with a screen apparently asking them to select a destination as an option they might do so again, being unsure that they had been successful the first time. This would not occur the other way round since once performed reactively, for whatever reason, the action is removed as a communication goal in the architecture.

Situations where it is desirable to offer the user a choice of redoing a previously completed operation might be a situation where this kind of potential confusion was introduced into a device design for perfectly good reasons. If so the design would need to take into account this problem as we will discuss later.

6.2 Reversing the order of actions

A second class of error is to reverse the order of two actions. This pattern of behaviour can arise from our architecture as a result of the way communication goals are modelled. In particular, communication goals can be discharged by the cognitive architecture in any order. Therefore, if an interactive system requires a particular sequence, then the order may be erroneously reversed by the architecture if the actions correspond to communication goals and the opportunity is presented. A person might insert money and then press the destination button when a particular ticket machine requires the money to be inserted second. This does not apply to non-communication goal actions, however. For example, two actions that are device dependent (pressing a confirmation button and one to release change, for example) will not be reversed by the cognitive architecture as they will be done in a reactive way.

6.3 Omission of actions

The cognitive architecture may omit actions at the end of a sequence. In particular, it may terminate the interaction at any point once the goal has been achieved. For example, once the person is holding the ticket they intended to buy, they may walk away from the machine, leaving their change, credit card or even return portion of their ticket. Whatever other rules are active, once the goal is achieved, the completion rule is active, so could be fired.

The cognitive architecture may also omit trailing actions if there is no apparent action possible. If at any time instance the guard of no other rule is active, then the guard of the termination rule becomes active and so the cognitive architecture terminates. There must always be some action apparently possible. This action could be to pause but only if given reactive guidance to do so. For example, if there is a period when the ticket machine prints the ticket, where the person must do nothing, then with no feedback to wait they may abort. In this respect the cognitive architecture does not quite reflect the way people behave. If there is no action possible the architecture is guaranteed to terminate, whereas in reality a person might pause before giving up. However, as our concern is user error, this is not critical as either way termination is possible so task completion is not guaranteed.

If a system throws away previously entered data, perhaps as a result of a later choice not being valid, then the cognitive architecture could omit repeating the earlier input if it is not prompted to do so. This is the converse problem to that in Section 6.1 where repetition of an action is an error. It arises for example in situations such as online flight booking systems where a destination is selected followed by a departure airport for which no route exists, leading to the system discarding both airports input.

If the cognitive architecture took an action early due to it corresponding to a communication goal (e.g. selecting a destination first instead of ticket type) then the cognitive architecture would assume that the action had had the desired effect. The action (selecting a destination) would be removed from the communication goal list: the cognitive architecture "believes" it has been performed. It then would not be done at the appropriate point in the interaction; i.e. a second (omission) error would occur. In this situation the correct action would be a *repetition* of the earlier action – repetition is not an error in this situation but only because it is required to recover from the previous action.

6.4 Late actions

The cognitive architecture does not put any time bounds on actions. All rules simply assert that once an action is selected then it will eventually occur. If any action must be done in a time critical manner, then the cognitive architecture will be capable of failing to do so. In practice this is too restrictive – it means the current cognitive architecture will always be able to fail with a device that resets after some time interval, for example, as would be normal for a ticket machine. Where such time criticality is inherent in a design, extra assumptions that deadlines are met would need to be added explicitly.

6.5 Early actions

If there are periods when an action can apparently be performed, but if performed is ignored by the computer system, then in some circumstances the cognitive architecture would take the next action early. In particular, if the user has outstanding communication goals then the corresponding actions may be taken early. This will potentially occur even if the device gives explicit guidance that the user must wait. This corresponds to the situation where a person does not notice the guidance but takes the action because they know they have to and have seen the opportunity. Similarly, if the device is presenting an apparent opportunity for reactive behaviour before it is ready to accept that action then the cognitive architecture could react to it.

6.6 Replacement of one action by another

Fields formally defines replacement [19] in a way that admits the substitution of any related action from the task for the correct one. Examples might include placing butter in the oven instead of the fridge, or switching on the headlights when meaning to switch on the windscreen wipers of a car. Our model cannot make such general replacements which could have a wide variety of cognitive causes. However, the cognitive architecture is, under appropriate circumstances, able to make one specific kind of replacement error due to indirect interface changes. In the context of mode errors, indirect mode changes have been specifically implicated as a problem that leads to operator error in a range of situations including flight decks. A particular problem occurs when the operator fails to mentally track the mode changes. (See for example [41] for a discussion).

The cognitive architecture exposes a more subtle form of problem that remains even in modeless systems or in a moded system where the pilot is aware of the mode. In particular, if an indirect change of state in the computer system can occur (that is not in response to a user action), then if the cognitive architecture has already committed to some action (such as pressing a button), but its effect changes between the commitment being made and the physical action actually being taken, then the wrong action with respect to the device will occur. This can lead to a person doing something they know is wrong. The change could occur due to a machine time-out or an environmental change

For example, suppose a ticket machine consisted of a touch screen showing virtual buttons for different ticket types such as off-peak travel card, single and return. If when the time changed to peak travel time, the off-peak option automatically disappeared and was replaced by a peak-travel card option, a person intending to select the off-peak travelcard, could instead without intending it select a peak travelcard. This may or may not be what they would have done had they realised the option was disappearing but either way the selection action they intended would have been replaced by one they did not. This problem has been identified in systems including Cash Machines, Email systems and an Air Traffic Control System [43].

6.7 Insertion of actions from elsewhere in the task

Insertion of an action can occur with communication goals. They can be attempted by the cognitive architecture at any point in the interaction where the opportunity to discharge them apparently presents itself. With reactive tasks, insertion will occur only if the device gives a reactive signal to suggest it can be done when it cannot.

6.8 Intrusion of actions unrelated to the task

Actions unrelated to the task can intrude with the cognitive architecture as a result of reactive signals on the device. If a device supports multiple tasks and uses reactive signals that signal an action to be performed that is not part of the task, such an action may be taken if there is nothing to suggest whether it is relevant or not.

6.9 Summary

In summary, the principles of cognition implemented in the architecture generate behaviours that account for a range of phenotypes. It is also apparent from the above discussion that the same underlying cause can account for erroneous actions corresponding to several different phenotypes. Furthermore a single phenotype can be a result of several different underlying causes. The behaviour specified in the architecture is neither correct nor incorrect behaviour. However, from it we have seen a wide variety of erroneous actions are possible.

7 Design Rules

We now examine some usability design rules. In particular, the design rules considered are those that emerge from the discussion in Section 6. They are derived from our informal analysis of the behaviour of the cognitive architecture. They are based on the underlying causes: the genotypes rather than the phenotypes. Here we present 9 rules:

- Allow the task to be finished no later than the goal.
- Provide information about what to do.
- Providing information is not enough.
- Use forcing functions.
- Make the interface permissive.

- Controls should make visible their use.
- Give immediate feedback.
- Do not change the interface under the user's feet.
- Where possible, determine the user's task early.

7.1 Task finished no later than the goal

The cognitive architecture contains a rule to terminate if the goal is achieved. Whatever other rules are active, this one could be activated due to the non-deterministic nature of the rules. The cognitive architecture can therefore terminate the moment its goal is achieved. Furthermore, no output from the device can prevent this as it would just result in additional rules being active which cannot preclude this action being taken. For the cognitive architecture to guarantee to not terminate early for this reason, the task must be completed no later than the goal. Tasks to restore the pertubations caused by an interaction must be completed either before the final action that achieves the goal or at the same time, or be designed out of the interaction completely. Any design that requires the cognitive architecture to perform extra completion tasks must ensure they are done before the goal is achieved. The rule will then only be active precisely when the task termination rule will be active, so that termination does not occur before the task rule is achieved.

For a ticket machine, taking the ticket must be the last action of the user. They must by then have taken change or hold their credit card, or these must be returned in a way that means they must be taken together or not at all. Multiple ticket parts (e.g. the return ticket) must also be dispensed together.

In practice (e.g. when termination involves logging out from a system) it may not always be possible to satisfy this design rule; in such situations, another means of restoring the invariant needs to be found. An attempted verification, based on the cognitive architecture, of a design that did not follow this design rule would fail because there would be an interaction path where the goal was achieved and so termination would occur on that path, when the task was not achieved. In particular, as noted above, providing extra information to tell the user to do the termination action is not sufficient.

We return to this design rule in Section 8. We use it as a simple case study in how a formal proof verifying a design rule against the cognitive architecture can be performed.

7.2 Provide information about what to do

Actions that are not communication goals can only be triggered in the architecture if they are a response to reactive signals – information indicating that the given rule is the next to be performed to achieve the given task. Therefore, if an action must be performed that does not correspond to a communication goal then information in the form of clear reactive guidance needs to be provided to tell the architecture to take the action.

In the case of a ticket machine, if a button must be pressed to confirm the ticket selected is the one required, then highly salient instructions to do this must be provided. There must be no other distracting options apparently possible at that point. Even then the instructions could be missed.

For communication goal actions, reactive information is not needed, though information linking the communication goal to the specific action is needed: something (such as the presence of a visible coin slot for inserting money) must make it clear that the communication goal can be discharged.

7.3 Providing information is not enough

The above design rules concern always providing information. A simple specific case might be to clearly indicate the order that actions should be taken in. This approach is often used where, for example, a panel gives instructions or lights flash to indicate the next button to press. The next design rule is that that is not good enough – so might appear to be contradictory. However, it depends on the situation.

Is providing information ever enough? According to the cognitive architecture – yes. It is sufficient if the user has nothing else to do and the action clearly takes them towards their goal. Thus (for our principles) if all communication goals are discharged (the ticket has been specified and money inserted) and the goal is not yet achieved (no ticket is held) then providing clear information is both useful and necessary. However, the cognitive architecture is non-deterministic. There may be several active rules and therefore several possible actions that could be taken. Reactive signals are not modelled as having higher priority than any other signal. Other possible actions are, for example, to terminate the interaction (if the goal is achieved), or discharge a communication goal. If the guards of such rules are active then they are possible actions. Making other signals true cannot make such a guard false; it can only make false guards true, so increasing the range of possible actions. Therefore, just providing flashing lights or beeps or other reactive signals is not enough to ensure correct operation if other actions are also possible. An attempted verification of such a design would fail because it would not be possible to prove that the correct action was taken. Some other action would be possible which could ultimately lead to the user aborting the interaction. If any possible path leads to abortion before the goal is achieved then a task completion correctness statement that states that the goal is achieved on all paths would be unprovable.

Thus, whilst providing appropriate information is important, on its own it should not be relied on - it needs to be combined with careful design of the interaction.

7.4 Forcing functions

The fact that the cognitive architecture is capable of taking several different options, and that giving reactive signals and messages is not enough, means that some other way is needed to ensure the options are narrowed down to only the correct ones. As Norman [33] suggests, in good design, only correct actions for the range of tasks supported at a point should be possible. Somehow, the design must ensure that the only cognitively plausible actions are correct ones. This suggests the use of forcing functions. This does not mean there must only be one button to press at any time, but that only buttons that it is valid to press at the current time can possibly be of use for a given task. Within the limits of the architecture, this means that if communication goals are not yet discharged, and should not yet be discharged, then there should be no apparent opportunity to discharge them.

For example, a soft screen (ie with online labelled buttons) might be used so that the only buttons pressable correspond to ones that can now correctly be pressed. If money cannot be inserted then coin slots should be closed.

Similarly, the solution to post-completion errors is to not allow the goal to be achieved until the task is completed – forcing the user to complete other completion tasks first (where possible), as discussed above, by structurally redesigning the interaction.

7.5 Permissiveness

Forcing functions follow the design principle that the options available to the user should be reduced. An alternative way of solving the same problem is to do the opposite and make the design permissive [45]: that is, the design does not force a particular ordering of events at all. In this case, the design should be such that each of the actions that can be taken by the cognitive architecture are accepted by the design and lead to the task being achieved. With our cognitive architecture, permissiveness cannot be used universally, however. For example, it is not sufficient with completion tasks to allow them to be done in any order. As we have seen, if the goal is achieved before the task is completed then the cognitive architecture leaves open the possibility of termination. There is no way the design can recover – once the cognitive architecture terminates it does not re-start the task. Therefore, in this situation, being permissive does not work. Fixing one error situation would introduce another. The ticket of a ticket machine must be released last. That action corresponds to the goal so cannot be permissive. At times in an interaction when communication goals are outstanding, the cognitive architecture could discharge them if the opportunity is present. Thus permissiveness is a useful design rule to apply to communication goals. In particular, permissiveness should be applied if forcing functions are not used when communication goals are active. A communication goal that appears dischargable should be dischargable. For example, a ticket machine could allow destination and ticket type to be chosen in any order.

7.6 Controls should make visible their correct use

The cognitive architecture provides for both reactive behaviour and directly goal-based behaviour. The guards of all cognitive architecture actions should include a signal indicating the presence of information suggesting they are appropriate actions. If a control is not labelled then the cognitive architecture will not take the action. Thus all controls must be labelled if the cognitive architecture is to use them. This does not mean that labels must be written. Their natural affordances [33] could suffice. That is, the physical form of a control may be considered sufficient to warrant the signal being asserted. For example, a coin slot advertises by its form that it is for the insertion of coins. This would need to be decided by a usability expert using complementary techniques to those considered here. Also, the control only needs to be visible at the point in the specific interaction where the cognitive architecture must take the action. Thus visibility need not be universal.

Conversely, care must be taken that affordances do not advertise an incorrect use. For example, slots for credit cards and rechargeable travel cards need to be obviously distinguished unless a single slot works for both.

7.7 Give immediate feedback

If there is no possible action apparent to the cognitive architecture then it will abort. That will occur if the guards of no rules are true. If a user must wait, then feedback to wait should appear immediately with nothing else apparently possible (e.g. no other buttons visible). One possible reactive action can always be to pause provided it is guarded by the existence of a "please wait" message.

For example if the user must wait while a ticket is printed then this needs to be explicitly clear to the user with no other distracting actions apparently possible.

7.8 Do not change the interface under the user's feet

The distinction between mental actions and physical actions, leads to a design rule that the interface should not change except in response to user action. There is a, possibly very short, delay between the cognitive architecture making a decision, after which it cannot stop the motor system, and the action actually occurring. This means that if the effect of an intended action changes after the decision has been made, but before the physical action is actually taken, then the cognitive architecture could do the wrong thing.

More specifically, therefore, a possible design rule is that no input to the computer system should change its effect spontaneously. This is quite restrictive, however. Less restrictive design possibilities are available to overcome the problems. This is linked to the design rule of Leveson *et al* [27] on avoiding indirect mode changes. Their concern was with mode changes and the person's inability to track them. Here the issue is much wider than just with mode awareness. Any changes to the interface that can indirectly change the behaviour of an action are problematic. There is still a problem even if the person is tracking the current state successfully.

One situation where this can be a problem is with timeouts – if no action is made in some period then the machine resets to some initial state. The cognitive architecture does not strictly support such behaviour at present. However, one possibility with the current limited cognitive architecture, and as used by some ticket machines, is to ask the user if they want more time after some delay. However, this could mean the buttons change their meanings. What did mean "I want to go to Liverpool" suddenly means "I do not want more time", for example. Such problems can be overcome, provided the old buttons all mean "I want more time", and the one that means "no more time" was not linked previously to any action – or with a soft-button interface the old button did not exist at all. Such a design would only work with the cognitive architecture if reactive signals were being used, as if the action were taken as a result of a communication goal, then that communication goal would have been discharged. The cognitive architecture would only take the action again if prompted reactively to do so.

7.9 Where possible, determine the user's task early

The cognitive architecture can take reactive actions intended for other tasks. This can be overcome if multiple-task devices determine the task to be performed at the first point of divergence between the tasks. For example, a ticket machine that can also be used as a cash point may have a common initial sequence inserting a credit card. However, once the tasks diverge, the next device action should be to determine the task the user is engaged in, in a way that makes no other actions (specifically communication goals for any other tasks) apparently possible. From then on actions from other tasks will not need to intrude in the design. This is important since a communication goal can be discharged at any point where apparently possible. In complex situations this will be difficult to achieve.

7.10 Summary

Having shown in Section 6 that the cognitive architecture can make many forms of error, we have argued in this section that design rules can be derived that fix the problems. Appropriate design can eradicate specific classes of error from the cognitive architecture. To the extent that the cognitive architecture does describe cognitively plausible behaviour, this means that the design rule will help prevent *systematic* user error. If design changes can eradicate systematic human error then they fall within the scope of formal verification techniques, as investigated in our parallel work [14]. In the next section, we continue to focus on design rules and look at how fully formal reasoning about the cognitive architecture, rather than the informal argument used here, can be used to justify design rules.

8 Verifying a User Error Design Rule

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 1.2. 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. Our contribution is not specifically in the theorem proved, which is quite simple, but rather in the demonstration of the general approach.

8.1 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 the error has not occurred. Thus whilst the error occurring is a direct consequence of the existence of this rule in the cognitive architecture, the rule is not directly modelling erroneous actions, just cognitively plausible behaviour that leads to an erroneous action in some situations.

We define formally the occurence of a post-completion error in definition `PCE_OCCURS`. It occurs if there is a time, `t`, before the end time of the interaction `te`, such that the probe `Goalcompletion` is true at that time but the invariant has not been re-established.

$\vdash_{def}$ PCE_OCCURS probes invariant te =
    $(\exists t.\ t \leq te \wedge \text{Goalcompletion probes } t \wedge \neg(\text{invariant } t))$

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.

8.2 Formalising a Design Rule

We next formalise one of the well-known user-centred design rules outlined in Section 7 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.

$\vdash_{def}$ PCE_DR goalachieved invariant te =
    $(\forall t.\ t \leq te \supset \neg(\text{goalachieved } t \wedge \neg(\text{TASK\_DONE goalachieved invariant } t)))$

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.

8.3 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 the HOL proof system, the following theorem stating this:

$\vdash$*thm* USER ... goalachieved invariant probes ustate mstate $\wedge$
    PCE_DR (goalachieved ustate) (invariant ustate) te $\supset$
        $\neg$(PCE_OCCURS probes (invariant ustate) te)

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. We are in effect taking a much wider definition of the "system" to include the user as well as the computer component. 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.

8.4 Machine-Centred Rules

The above design rule is in terms of user concerns – an invariant of the form suitable for the cognitive architecture 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 such as below, for that.

$\vdash$*def* MACHINE_PCE_DR goalevent minvariant te =
    $(\forall$t. goalevent t $\supset$ $(\forall$t1. t $\leq$ t1 $\wedge$ t1 $\leq$ te $\supset$ 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 ticket machine, the goal as specified in a user-centred way is that the user has a ticket. Once this first becomes true it will continue to hold

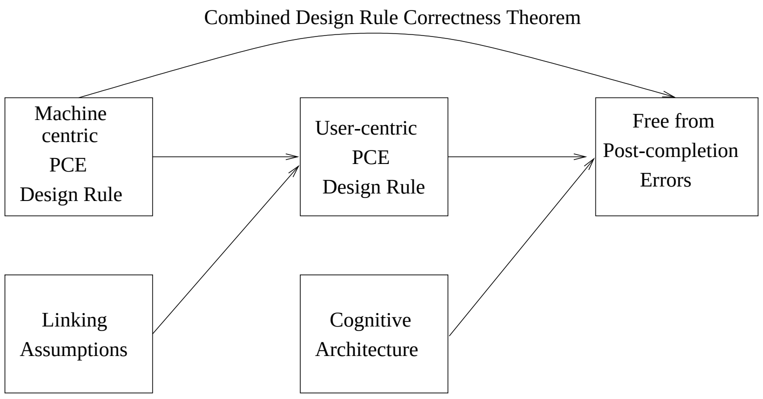
Combined Design Rule Correctness Theorem
Machine centric PCE Design Rule
User-centric PCE Design Rule
Free from Post-completion Errors
Linking Assumptions
Cognitive Architecture

Figure 2: Verifying the Design Rule in Stages

until the end of the interaction, since for the purposes of analysis we assume that the user does not give up the ticket again until after the interaction is over. The machine event however, is that the machine fires a signal that releases the ticket. This is a relation on the machine state rather than on the user state. 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, rather than the user's, are the same as at the start of the interaction – it having exchanged a ticket for an equal value 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 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 have proved that this new design rule implies the original, provided assumptions are met about the relationship between the two forms of goal statements and invariants. These assumptions form the basis of the integration between the user and machine-centred worlds.

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

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.

8.5 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:

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

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.

9 Discussion

We have outlined a formal description of a very simple cognitive architecture. The cognitive architecture describes fallible behaviour. However, rather than explicitly describing *erroneous* behaviour, it is based on *cognitively plausible* behaviour. Despite this we show that a wide variety of erroneous actions can occur from the behaviour described in appropriate circumstances. We have considered how devices (software, hardware or even everyday objects) must be designed if a person acting as specified by the cognitive architecture is to be able to successfully use the device. We have shown how well-known design rules, if followed, would allow this to occur. Each of these rules removes potential sources of user error that would prevent the verification of a design against the cognitive architecture using the techniques described in [14]. We thus provide a theoretically based set of design rules, built upon a formal model. This model has very precise semantics that are open to inspection. Of course our reasoning is about what the cognitive architecture might do rather than about any real person. As such, the results should be treated with care. However, errors that the cognitive architecture could make can be argued to be cognitively plausible and so worth attention.

Ad-hoc lists of design rules can easily appear to be contradictory or only apply in certain situations. By basing them on cognitively plausible principles, we can reason about their scope and make this scope more precise. For example, should systems always be permissive [45], allowing any action to be taken, or only under certain circumstances? At first sight, permissiveness appears to contradict forcing functions [33] when only certain actions are made possible. By reasoning from cognitive principles we can be precise about these surface contradictions.

One of our aims has been to demonstrate a lightweight use of formal methods. As such, we have started with a formal description of user behaviour and used it as the basis for *semi-formal* reasoning about what erroneous behaviours emerge, and design rules that prevent behaviours emerging. Such semi-formal reasoning could be erroneous. We have also explored the formal, machine-checked verification of a design rule. Using HOL (the proof system the cognitive architecture is defined within), this involves giving formal descriptions of design rules and proving that – under the assumptions of the cognitive architecture – particular erroneous situations do not occur. We demonstrated this by considering a design rule intended to prevent one class of systematic user error – post-completion errors – occurring.

We specified two versions of a design rule intended 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 also 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. Such an integration to hardware verification is described in [13].

The work presented here builds on our previous work on fully formal proofs that an interactive system completes a task [14]. 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.

10 Further Work

We have only fully formally 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. Further work is needed to formally model and verify these other error classes and design rules. 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 test 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, it will be possible to model more erroneous behaviour. 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 cognitive architecture 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. This work is funded by EPSRC grants GR/S67494/01 and GR/S67500/01.

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, September 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 craft skill in usability evaluation. In *Proceedings of Interact*, pages 577–584, 2001.

[5] J. Bredereke and A. Lankenau. A rigorous view of mode confusion. In *Proc. of Safecomp 2002, 21st International Conf. on Computer Safety, Reliability and Security*, volume 2434 of *Lecture Notes in Computer Science*, page 1931. Springer-Verlag, 2002.

[6] 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. Vanderdonckt, editors, *Proc. Design, Specification and Verification of Interactive Systems '96*, pages 347–365. Springer, 1996.

[7] Ricky W. Butler, Steven P. Miller, James N. Potts, and Victor A. Carreño. A formal methods approach to the analysis of mode confusion. In *17th AIAA/IEEE Digital Avionics Systems Conference*, Bellevue, WA, October 1998.

[8] 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.

[9] R. Butterworth, A.E. Blandford, and D. Duke. Demonstrating the cognitive plausibility of interactive systems. *Formal Aspects of Computing*, 12:237–259, 2000.

[10] M. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.

[11] 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.

[12] Judith Crow, Denis Javaux, and John Rushby. Models and mechanized methods that integrate human factors into automation design. In *International Conference on Human-Computer Interaction in Aeronautics: HCI-Aero*, September 2000.

[13] P. Curzon and A. Blandford. Formally justifying user-centred design rules: a case study on post-completion errors. In E.A. Boiten, J. Derrick, and G. Smith, editors, *Proc. of the 4th International Conference on Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, pages 461–480. Springer, 2004.

[14] 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.

[15] P. Curzon and A.E. Blandford. A user model for avoiding design induced errors in soft-key interactive systems. In R.J. Bolton and P.B. Jackson, editors, *TPHOLS 2001 Supplementary Proceedings*, number ED-INF-RR-0046 in Informatics Research Report, pages 33–48, 2001.

[16] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction.* Prentice Hall, 3rd edition, 2004.

[17] A. J. Dix and C. Runciman. Abstract models of interactive systems. In P. J. Cook and S. Cook, editors, *People and Computers: Designing the Interface*, pages 13–22. Cambridge University Press., 1985.

[18] D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–394, 1998.

[19] 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.

[20] M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[21] 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.

[22] W.D. Gray. The nature and processing of errors in interactive behavior. *Cognitive Science*, 24(2):205–248, 2000.

[23] E. Hollnagel. *Cognitive Reliability and Error Analysis Method.* Elsevier, 1998.

[24] 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.

[25] A. Lankenau. Avoiding mode confusion in service-robots. In M. Mokhtari, editor, *Integration of Assistive Technology in the Information Age, Proc. of the 7th Int. Conf. on Rehabilitation Robotics*, page 162167. IOS Press, 2001.

[26] D. Leadbetter, P. Lindsay, 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.

[27] N.G. Leveson, L.D. Pinnel, S.D. Sandys, S. Koga, and J.D. Reese. Analyzing software specifications for mode confusion potential. In C.W. Johnson, editor, *Proceedings of the Workshop on Human Error and System Development*, pages 132–146, March 1997. Glasgow Accident Analysis Group Technical Report GAAG-TR-97-2 Available at www.cs.washington.edu/research/projects/safety/www/papers/glascow.ps.

[28] Gerald Lüttgen and Victor Carreño. Analyzing mode confusion via model checking. In D. Dams, R. Gerth, S. Leue, and M.Massink, editors, *SPIN'99*, number 1680 in Lecture Notes in Computer Science, pages 120–135. Springer-Verlag, 1999.

[29] P. Markopoulos, P. Johnson, and J. Rowson. Formal architectural abstractions for interactive software. *International Journal of Human Computer Studies*, 49:679–715, 1998.

[30] 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.

[31] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[32] J. Nielsen. Heuristic evaluation. In J. Nielsen and R.L. Mack, editors, *Usability Inspection Methods*. John Wiley and Sons, 1994.

[33] Donald Norman. *The Design of Everyday Things*. Currency-Doubleday, 1988.

[34] 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.

[35] 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. *International Journal of Human-Computer Studies*, 55:1–14, 2001.

[36] 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.

[37] John Rushby. Analyzing cockpit interfaces using formal methods. *Electronic Notes in Theoretical Computer Science*, 43, 2001.

[38] John Rushby. Modeling the human in human factors. In U. Vogues, editor, *SAFECOMP 2001*, volume 2187 of *Lecture Notes in Computer Science*, pages 86–91. Springer-Verlag, 2001.

[39] John Rushby. Using model checking to help discover mode confusions and other automation suprises. *Reliability Engineering and System Safety*, 75(2):167–177, 2002. Originally presented at the 3rd Workshop on Human Error, Safety and System Development, 1999.

[40] John Rushby, Judith Crow, and Everett Palmer. An automated method to detect potential mode confusions. In *18th AIAA/IEEE Digital Avionics Systems Conference (DASC)*, October 1999.

[41] N.B. Sartar, D.D. Woods, and C.E. Billings. Automation suprises. In G. Salvendy, editor, *Handbook of Human Factors and Ergonomics*. Wiley, 2nd edition, 1997.

[42] Ben Shneiderman. *Designing the User Interface*. Addison Wesley, 3rd edition, 1998.

[43] S. Buckingham Shum, A. Blandford, D. Duke, J. Good, J. May, F. Paterno, and R. Young. Multidisciplinary modelling for user-centred design: An air-traffic control case study. In M.A. Sasse, R.J. Cunningham, and R.L. Winder, editors, *Proceedings of Human-Computer Interactoin '96, People and Computers XI*, BCS Conference Series, pages 201–220. Springer-Verlag, 1996.

[44] H. Thimbleby. Generative user-engineering principles for user interface design. In B. Shackel, editor, *Proceedings First IFIP Conference on Human Computer Interaction, INTERACT'84*, pages 102–107. North Holland, 1984.

[45] H. Thimbleby. Permissive user interfaces. *International Journal of Human-Computer Studies*, 54(3):333–350, 2001.

[46] H. Thimbleby, A. Blandford, P. Cairns, P. Curzon, and M. Jones. User interface design as systems design. In Xristine Faulkner, Janet Finlay, and Francoise Detienne, editors, *People and Computers XVI Memorable Yet Invisible, Proceedings of the 16th British HCI Conference*, volume 1, pages 281–302. Springer, 2002.

A The formal cognitive architecture

In this appendix we give the formal HOL definitions of the cognitive architecture. A version that is loadable in HOL can be obtained from the HUM website at
http://www.dcs.qmul.ac.uk/research/imc/hum/

A.1 Basics

In this section the definitions underpinning the timing of actions are given. The key definition is `NEXT`. It is then used throughout the remainder of the cognitive architecture to specify when actions occur. It is based on local definitions `STABLE`, `LSTABLE` and `LF`.

`STABLE` asserts that the signal, `P`, has the same value `v` between times `t1` and `t2`.

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

`LSTABLE` asserts that `STABLE` holds for a list of signals.

```
⊢def (LSTABLE [ ]t1 t2 v = TRUE) ∧
     (LSTABLE (a :: l) t1 t2 v =
        STABLE a t1 t2 v ∧
        LSTABLE l t1 t2 v)
```

Given an argument 0 for `n`, `LF` states that all actions in a list except for a specified one given by position, `ppos`, are inactive (false) at the given time.

```
⊢def (LF n [ ]ppos t = TRUE) ∧
     (LF n (a :: l) ppos t = (((n = ppos) ∨ ¬(a t)) ∧
                              LF (SUC n) l ppos t))
```

`NEXT` asserts that the next action from a list of actions to become true after time `t1` is `action`.

```
⊢def NEXT flag actions action t1 =
        ∃t2. t1 <= t2 ∧
             LSTABLE actions t1 t2 F ∧
             (LF 0 actions action t2) ∧
             EL action actions t2 ∧
             (flag (t2+1)) ∧
             STABLE flag (t1+1) (t2+1) F
```

A.2 Possessions

In this subsection we define some physical restrictions on possessions and their values.

`HAS_POSSESSION` is used to define the physical restrictions on possessions of an individual, linking for a single kind of possession the events of taking and giving up a possession, having a possession, the number of an object possessed and its value. Properties include that a person has a possession if its count is greater than 0; if in a time instance a person takes a possession and does not give it up, then the count goes up by 1; etc.

```
⊢def HAS_POSSESSION haspossession takepossession givepossession
          valueofpossession countpossession (ustate:'u) (mstate:'m) =
      (∀t. haspossession ustate t = (countpossession ustate t > 0)) ∧
      (∀t. (givepossession mstate t ∧ ¬(takepossession mstate t)) =
             ((countpossession ustate t > 0) ∧
             (countpossession ustate (t+1) = countpossession ustate t - 1 ))) ∧
      (∀t. (takepossession mstate t ∧ ¬(givepossession mstate t)) =
         (countpossession ustate (t+1) = countpossession ustate t + 1 )) ∧
      (∀t.      ((¬(givepossession mstate t) ∧ ¬(takepossession mstate t)) ∨
         ((givepossession mstate t) ∧ (takepossession mstate t))) =
             (countpossession ustate (t+1) = countpossession ustate t))
```

Possessions are recorded as a new state tuple type recording having a possession, taking one, giving one up, the value and the count of the possession. Corresponding functions that access the tuples' fields are defined.

```
pstate_type = :('u → num → bool) # ('m → num → bool) # ('m → num → bool) #
             num # ('u → num → num)
```

⊢*def* HasPossession (pstate: pstate_type) = FST pstate

⊢*def* TakePossession (pstate: pstate_type) = FST (SND pstate)

⊢*def* GivePossession (pstate: pstate_type) = FST (SND (SND pstate))

⊢*def* ValueOfPossession (pstate: pstate_type) = FST (SND (SND (SND pstate)))

⊢*def* CountPossession (pstate: pstate_type) = SND (SND (SND (SND pstate)))

A collection of possessions is recorded as a list of possession tuples. `POSSESSIONS` asserts that `HAS_POSSESSION` holds of each.

```
⊢def (POSSESSIONS [ ] (ustate:'u) (mstate:'m) = TRUE) ∧
      (POSSESSIONS ((p: pstate_type) :: possessions) ustate mstate =
         ((POSSESSIONS possessions ustate mstate) ∧
         (HAS_POSSESSION (HasPossession p) (TakePossession p)
                         (GivePossession p) (ValueOfPossession p)
                         (CountPossession p)
                         ustate mstate)))
```

The value of a list of possessions is the total value of the possessions of each type.

$\vdash_{def}$ (POSSESSIONS_VAL [ ] (ustate:'u) t = 0) $\wedge$
    (POSSESSIONS_VAL ((p: pstate_type) :: ps) (ustate:'u) t =
        ((POSSESSIONS_VAL ps (ustate:'u) t) +
            ((CountPossession p ustate t) * (ValueOfPossession p))))

A.3 Probes

In this subsection we define probes. They are signals that record the firing of other behavioural rules at each time instance. In this version of the cognitive architecture there is only a single probe related to a goal completion rule (given later) firing.

$\vdash_{def}$ Goalcompletion probes t = probes t

A.4 Reactive Signals

In this subsection we define rules about how a user might react to external stimulus from a device such as lights flashing.

Reactive signals are provided to the cognitive architecture as a list of pairs consisting of a stimulus and a resulting action. We first define accessor functions for the tuples.

$\vdash_{def}$ Stimulus stimulus_actions = FST stimulus_actions

$\vdash_{def}$ Action stimulus_actions = SND stimulus_actions

`REACT` is the rule for reacting to a stimulus. The relation is true at a time, `t` if the stimulus is true and the next action is the given action. As this is not the goal completion rule, the goal completion probe is false.

$\vdash_{def}$ REACT flag actions stimulus action probes t =
        (stimulus t = TRUE) $\wedge$
        (Goalcompletion probes t = FALSE) $\wedge$
        NEXT flag actions action t

`REACTS` states that given a list of reactive stimulus rules, any can fire (their relation can be true) at a time instance.

$\vdash_{def}$ (REACTS flag actions [ ] probes t = FALSE) $\wedge$
    (REACTS flag actions (s :: stimulus_actions) probes t =
        ((REACTS flag actions stimulus_actions probes t) $\vee$
        (REACT flag actions (Stimulus s) (Action s) probes t)))

A.5 Mental Commitments

In this subsection we define what it means to have made a mental commitment to take a physical action.

Mental commitments are pairs, consisting of a guard and an action that can occur if the guard is true. Given a list of such pairs, `CommitmentGuards` extracts all the guards.

$\vdash_{def}$ CommitmentGuard commitments = FST commitments

$\vdash_{def}$ CommitmentAction commitments = SND commitments

$\vdash_{def}$ CommitmentGuards commitments = MAP CommitmentGuard commitments

The rule for turning a guard mental action (irrevokable decision) into a physical one, is that if the mental action was true on the previous cycle then the next action is a corresponding physical action as given by `COMMITS`.

```
⊢def COMMIT flag actions maction paction t =
          (maction (t-1) = TRUE) ∧ NEXT flag actions paction t
```

Given a list of mental commitments the rule corresponding to any one of them can fire.

```
⊢def (COMMITS flag actions [ ] t = FALSE) ∧
      (COMMITS flag actions (ca :: commits_actions) t =
          ((COMMITS flag actions commits_actions t) ∨
          (COMMIT flag actions (CommitmentGuard ca) (CommitmentAction ca) t)))
```

Given a list of commitments, a commitment is currently made if the guard of any one of them was true on the previous cycle.

```
⊢def (CommitmentMade [ ] t = FALSE) ∧
      (CommitmentMade (maction :: rest) t =
              (maction(t-1) = TRUE) ∨ (CommitmentMade rest t))
```

A.6 Communication Goals

In this subsection we define rules related to one form of task related knowledge: communication goals.

`FILTER` takes two lists. The first is a list of all signals. The second a list of positions in that first list. For each position, it checks if that signal is true at the current time `t` and removes it from the list of positions if so. This is used to take a communication goal list and remove all those for which the action was performed.

```
⊢def (FILTER actions [ ] t = [ ] ) ∧
      (FILTER actions (a :: act) t =
          IF (EL (FST a) actions t
          THEN (FILTER actions act t)
          ELSE (a :: (FILTER actions act t))))
```

A history list (ie a list of actions) is filtered if at the next time instance all those entries which were active currently are removed from the list.

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

A communication goal is a pair consisting of a an action and a guard. We define corresponding accessor functions.

$\vdash$*def* ActionOfComGoal cg = FST cg

$\vdash$*def* GuardOfComGoal cg = SND cg

A communication goal rule fires if the guard of the communication goal is true, the goal has not been achieved and the next action is the given action. This is not the goal completion rule so its probe is false.

```
⊢def COMMGOAL flag actions action guard goal probes (ustate:'u) (mstate:'m) t =
        ¬(goal ustate t) ∧
        (guard t) ∧
        (Goalcompletion probes t = FALSE) ∧
        NEXT flag actions action t
```

`COMMGOALER` asserts using the subsidiary definition `COMMGOALS` that given a list of communication goals, any one of them can fire at a given time.

```
⊢def (COMMGOALS flag actions [ ] goal probes (ustate:'u) (mstate:'m) t = FALSE) ∧
     (COMMGOALS flag actions (a :: acts) goal probes ustate mstate t =
        ((COMMGOALS flag actions acts goal probes ustate mstate t) ∨
         (COMMGOAL flag actions (ActionOfComGoal a) (GuardOfComGoal a)
               goal probes ustate mstate t)))
```

```
⊢def COMMGOALER flag actions acts goal probes (ustate:'u) (mstate:'m) t =
        COMMGOALS flag actions (acts t) goal probes ustate mstate t
```

A.7 Goal-based Termination

In this subsection we define termination behaviour based on a person's goals.

The first action, always at position 0 of the list of all actions is the one that indicates when an interaction is terminated.

$\vdash$*def* FINISHED = 0

The goal based completion rule fires when the goal is achieved and the next action is to finish the interaction. This is the goal completion rule so the goal completion probe fires.

```
⊢def COMPLETION flag actions goalachieved probes (ustate:'u) (t:num) =
        (Goalcompletion probes t = TRUE) ∧
        (goalachieved ustate t = TRUE) ∧
        NEXT flag actions FINISHED t
```

A.8 Termination due to no options

In this subsection we define termination behaviour that results from there being no useful steps that can be taken.

We first define some simple list operators. They are used to manipulate the lists of guards for the other rules to create a guard that fires when no other guard fires. `NOT_CONJL` takes a list of booleans and creates a new list with all its entries negated. `CONJ1L` takes a list of booleans and ands a boolean to each element of the list. `APPLYL` applies a function to each element of a list.

```
⊢def (NOT_CONJL [ ] = TRUE) ∧
     (NOT_CONJL (P :: l) = ¬P ∧ (NOT_CONJL l))

⊢def (CONJ1L P [ ] = [ ]) ∧
     (CONJ1L P (Q :: l) = ((P ∧ Q) :: (CONJ1L Q l)))

⊢def (APPLYL [ ] a = [ ]) ∧
     (APPLYL (f :: l) a = ((f a) :: (APPLYL l a)))
```

The basic rule for no-option based termination, `ABORT` fires if its guard is true, leading to the next action being to finish the interaction. `ABORTION` constructs the guard for `ABORT` so that it holds if no other rule's guard holds.

```
⊢def ABORT flag actions guards probes (ustate:'u) (t:num) =
         (Goalcompletion probes t = FALSE) ∧
         (guards = TRUE) ∧
         NEXT flag actions FINISHED t

⊢def ABORTION flag actions goalachieved commgoals stims_actions probes
             (ustate:'u) (mstate:'m) (t:num) =
         ABORT flag actions
             (NOT_CONJL
                 ((goalachieved ustate t) ::
                 (APPEND
                  (CONJ1L (¬(goalachieved ustate t))
                              (APPLYL (MAP GuardOfComGoal (commgoals t)) t))
                  (APPLYL (MAP FST stims_actions) t) ) ) )
             probes ustate t
```

A.9 Top Level Definitions of the Architecture

In this subsection we give the top level definitions of the architecture, pulling the separate behaviours together into a single architecture.

The initial communication goals at time 1 are set to those supplied as an argument to the cognitive architecture as a whole.

```
⊢def USER_INIT commgoals init_commgoals = (commgoals 1 = init_commgoals)
```

The task is completed when the goal is achieved and the interaction invariant is restored.

```
⊢def TASK_DONE goal inv t = (goal t ∧ inv t)
```

The rules encoding different reasons for taking an action are combined by disjunction, so as to be non-deterministic. In the current version of the architecture they consist of goal completion, reacting to a stimulus, executing a communication goal and finishing due to no options.

```
⊢def USER_RULES flag actions commgoals stimulus_actions goalachieved mstate ustate t =
        COMPLETION flag actions goalachieved probes ustate t ∨
        REACTS flag actions stimulus_actions probes t ∨
        COMMGOALER flag actions commgoals goalachieved probes ustate mstate t ∨
        ABORTION flag actions goalachieved commgoals stimulus_actions probes ustate mstate t
```

The non-deterministic rules are wrapped in an if-then else structure. If the interaction has already been terminated it stays terminated. If a mental commitment has been made then it is carried out. If the task is completed then the interaction terminates. Only if none of the above hold do the non-deterministic options come into play. The above is only a consideration if the flag is true as it indicates the time is such that a new decision needs to be made by the architecture. It is set by each rule to cover the time period over which that rule has committed to a decision.

```
⊢def USER_CHOICE flag actions commitments commgoals stimulus_actions
            finished goalachieved invariant probes (ustate:'u) (mstate:'m) =
    (∀t.
    ¬(flag t) ∨
    (IF (finished ustate (t-1))
     THEN (NEXT flag actions FINISHED t ∧ (Goalcompletion probes t = FALSE))
     ELSE IF (CommitmentMade (CommitmentGuards commitments) t)
     THEN (COMMITS flag actions commitments t ∧ (Goalcompletion probes t = FALSE))
     ELSE IF TASK_DONE (goalachieved ustate) (invariant ustate) t
     THEN (NEXT flag actions FINISHED t ∧ (Goalcompletion probes t = FALSE))
     ELSE USER_RULES flag actions commgoals stimulus_actions
            goalachieved probes mstate ustate t))
```

To make the above rules work, in the background various other process need to take place at every time instance. If the interaction is terminated it stays terminated. The rules about possessions must always hold. The person's internal communication goal list is maintained from time instance to time instance with only completed actions removed. Where rules are not driving the probe value directly its signal remains false.

```
GENERAL_USER_UNIVERSAL actions commgoals possessions finished flag
            probes (ustate:'u) (mstate:'m) =
    (∀t. finished ustate t ⊃ finished ustate (t+1)) ∧
    (POSSESSIONS possessions ustate mstate) ∧
    (FILTER_HLIST actions commgoals) ∧
    (∀t. ¬(flag t) ⊃ (Goalcompletion probes t = FALSE))
```

The above properties of each time instance are combined with the rule about initialising the cognitive architecture.

$\vdash_{def}$ USER_UNIVERSAL flag actions commgoals init_commgoals possessions
        finished probes (ustate:'u) (mstate:'m) =
    (USER_INIT commgoals init_commgoals) $\wedge$
    (GENERAL_USER_UNIVERSAL actions commgoals possessions
        finished flag probes ustate mstate)

Finally the full cognitive architecture is the combination of `USER_UNIVERSAL` about the background processes and `USER_CHOICE` about the actual rules that drive the actions.

$\vdash_{def}$ USER flag actions commitments commgoals init_commgoals stimulus_actions
        possessions finished goalachieved invariant probes (ustate:'u) (mstate:'m) =
    (USER_UNIVERSAL flag actions commgoals init_commgoals possessions finished
        probes ustate mstate) $\wedge$
    (USER_CHOICE flag actions commitments commgoals stimulus_actions
        finished goalachieved invariant probes ustate mstate)