Formal Modelling of Cognitive Interpretation

Rimvydas Rukšėnas,[1] Paul Curzon,[1] Jonathan Back[2] and Ann Blandford[2]

[1] Department of Computer Science, Queen Mary, University of London
{rimvydas,pc}@dcs.qmul.ac.uk
[2] University College London Interaction Centre
{j.back,a.blandford}@ucl.ac.uk

Abstract. We formally specify the interpretation stage in a dual state space human-computer interaction cycle. This is done by extending / reorganising our previous cognitive architecture. In particular, we focus on shape related aspects of the interpretation process associated with device input prompts. A cash-point example illustrates our approach. Using the SAL model checking environment, we show how the extended cognitive architecture facilitates detection of prompt-shape induced human error.

Key words: human error, cognitive architecture, model checking, SAL.

1 Introduction

Interactive systems combine human and computer actors. Their correctness depends on the behaviour of both. It is reasonable, and useful, to say that humans behave rationally: entering interactions with goals and domain knowledge likely to help them achieve their goals. Whole classes of persistent, systematic user errors may occur due to modelable cognitive causes [1]. Often opportunities for making such errors can be reduced with good design [2]. A methodology for detecting designs that allow users, when behaving in a rational way, to make systematic errors will improve such systems.

We previously [3] developed a generic formal cognitive model from abstract cognitive principles, such as entering an interaction with knowledge of the task's subsidiary goals, showing its utility for detecting some systematic user error. Here we describe a development of it. The cognitive architecture previously identified device signals with user perception of them: a gross simplification of the complex process which involves perception, interpretation and evaluation. Similarly, user actions were identified with their corresponding device input. Such simplifications can hide design flaws. We address this problem by separating the user and the device state spaces. We also structure the underlying state space of the cognitive architecture to distinguish input signals (originating from user perception), output signals (consequences of user actions) and internal state (user memory). The formal version of our generic user model, module `User`, is outlined in Sect. 2. Our restructuring introduces intermediate entities, *interpretation* and *effect*, relating the now distinct state spaces (see Fig. 1) described in detail in Sect. 3. The *effect* is an abstract view of how user actions are translated into device

¶ Preprint to appear in *Interactive Systems: Design, Specification and Verification (DSV-IS 2006)*, Lecture Notes in Computer Science, 2006.

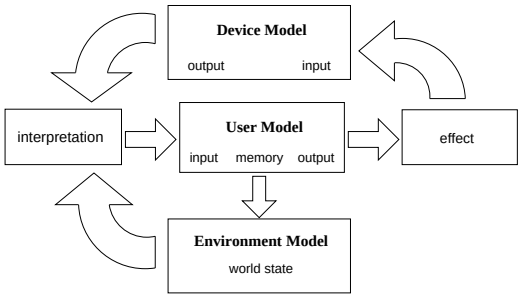

Fig. 1. The cycle of interaction.

commands. The importance of such translation is evident in interactive systems involving voice or gesture recognition of user inputs. More detailed modelling of this is left to future work. The *interpretation* is an abstract view of the pathways from device signals and environment objects to the user decision of what they could mean. Blandford and Young [4] also separate user and device descriptions. An important difference (apart from the degree of formality) is our explicit inclusion of the interpretation and effect stages within the cycle of interaction. To illustrate the utility of the changes, we focus here on one use: modelling how the shape (form, size, etc.) of a device prompt affects user (mis)understanding. The shape of a device prompt may restrict or structure the type of information that a user is required to provide.

Dillon [5] argued that the shape concept assumes both spatial and semantic characteristics. He found that, as well as using spatial cues, individuals recognise and respond to content and meaning when interpreting requirements. During an interaction there is an inter-coupling of spatial and semantic components of memory. It is therefore likely that the interpretation of a device prompt relies on both spatial and semantic cues. A device prompt may incorporate both of these cue types: e.g. a user may be required to use several different passwords depending on the service required. A fixed digit entry field can act as both a spatial and semantic cue. The location of the password entry field (spatial cue) may correspond to the specific service required, while the size of the entry field (both a spatial and semantic cue) may correspond to the required length of the password and hence inform the user of which password is required (if passwords are of different lengths). Semantic cues are hints based on meaning that help users understand requirements. Dillon and Shaap [6] found that experienced users could better process these semantics, while novices had to rely solely on spatial cues. However, not all semantic cues are based on knowledge of the system. For example, a user with no knowledge of the system can still use the size of the entry field as a semantic cue as it only requires knowledge of passwords.

Information appears to be processed automatically without conscious effort [7]. When cognitive operations are underspecified (e.g., when multiple inputs are possible), humans tend to default to high frequency responses. (frequency biasing). The largest single class of action slip errors are due to strong habit intru-

sions that have structural (spatial) and contextual (semantic) elements in common with planned actions [1]. Detecting cases where device prompt shape can be misunderstood may enable certain types of action slip errors to be avoided.

There are several approaches to formal reasoning about usability. One is to focus on a formal specification of the user interface [8, 9]. Most commonly this approach relies on model-checking tools; investigations include whether a given event can occur or whether properties hold of all states. An alternative is formal user modelling, as here. It involves writing formal specifications of both the computer system and the user, then reasoning about their conjoint behaviour. Both device and user are considered as equally central components of the system and modelled as part of the analysis. Duke and Duce [10] formally reason about HCI this way; their approach is well suited to reasoning about interaction that, for example, combines the use of speech and gesture. However, their framework lacks tool support, which would make it difficult to apply in practice. Bowman and Faconti [11] formally specify a cognitive architecture using the process calculus LOTOS, analysing its properties using temporal logic. These approaches are more detailed than ours, which abstracts above cognitive processes. Moher and Dirda [12] 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 analysis of desirable properties.

Rushby *et al* [13] focus specifically on mode errors and the ability of pilots to track mode changes. They formalise plausible mental models of systems and analyse them using the Murϕ state exploration tool. However, the mental models are essentially abstracted system models; they do not rely upon structure provided by cognitive principles. Neither do they model user interpretation.

Campos and Doherty [14] use perception mappings to specify mental models; no formal model of user behaviour is developed. Instead, they reason about the properties of representations of information in the interface. Also, perception in their approach seems to be always deterministic. Bredereke and Lankenau [15] reason about user perception of reality using a refinement based approach. Perception is expressed as 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, the authors note that in practice they use the relation to rename events and so it is not lossy. This contrasts with our work which explicitly considers imperfect user interpretation. Cerone *et al*'s [16] CSP model of an air traffic control system includes controller behaviour. A model checker was used to look for new behaviourial patterns, overlooked by the analysis of experimental data. The classification stage of the interaction cycle of their model is similar to our user interpretation.

2 The Cognitive Architecture in SAL

Our cognitive architecture is a higher-order logic formalisation of abstract principles of cognition and specifies cognitively plausible behaviour [17]. The archi-

Table 1. A fragment of the SAL language

`x:T`	`x` has type `T`
`x' = e`	the new value of `x` is that of the expression `e`
`{x:T | p}`	a subset of `T` such that the predicate `p` holds
`a[i]`	the `i`-th element of the array `a`
`r.x`	the field `x` of the array `r`
`r WITH .x := e`	the record `r` with the field `x` replaced by the value of `e`
`g` → upd	if `g` is true then update according to `upd`
`c [] d`	non-deterministic choice between `c` and `d`
`[](i:T): c`$_i$	non-deterministic choice between the `c`$_i$ where `i` is in range `T`

tecture specifies possible user behaviour (traces of actions) that can be justified in terms of specific results from the cognitive sciences. Real users can act outside this behaviour, about which the architecture says nothing. Its predictive power is bounded by the situations where people act according to the principles specified. The architecture allows one to investigate what happens if a person acts in such plausible ways. The behaviour defined is neither "correct" or "incorrect". It could be either depending on the environment and task in question. We do not attempt to model underlying neural architecture nor the higher level cognitive architecture such as information processing. Instead our model is an abstract specification, intended for ease of reasoning.

Our previous formalisation of the cognitive architecture was developed in a theorem prover. The new version is based on the SAL model checking environment [18]. It provides a higher-order specification language and tools for analysing state machines specified as parametrised modules and composed either synchronously or asynchronously. The SAL notation we use is given in Table 1.

We rely upon cognitive principles that give a *knowledge level* description in the terms of Newell [19]. Their focus is on the goals and knowledge of a user. In this section, we discuss the principles and the way they are modelled.

Non-determinism In any situation, any one of several cognitively plausible behaviours might be taken. It cannot be assumed that any specific plausible behaviour will be the one that a person will follow. The SAL specification is a transition system. Non-determinism is represented by the non-deterministic choice, `[]`, between the named guarded commands (i.e. transitions). Each describes an action that a user *could* plausibly make. For example, in the following, *ReactCommit* is the name of a family of transitions indexed by `i:ReactRange`.

```
TRANSITION
   ([](i:GoalRange): GoalCommit:..) [] ([](i:GoalRange): GoalTrans:..)
[] ([](i:ReactRange): ReactCommit:..) [] ([](i:ReactRange): ReactTrans:..)
[] Exit:..[] Abort:..[] Idle:..
```

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, it cannot be stopped even if the person becomes aware that it is wrong before the action is taken. Therefore,

we model both *physical* and *mental* actions. Each physical action modelled is associated with an internal mental action that commits to taking it. In the SAL specification, this is reflected by the pairings of guarded commands: *GoalCommit – GoalTrans* and *ReactCommit – ReactTrans*. The first of the pair models committing to an action, the second actually doing it (see below).

User goals A user enters an interaction with knowledge of the task and, in particular, task dependent sub-goals that must be discharged. These sub-goals might concern information that must be communicated to the device or items (such as credit cards) that must be inserted into the device. Given the opportunity, people may attempt to discharge any such goal, even when the device is prompting for a different action. We model such knowledge as user goals which represent a pre-determined partial plan that has arisen from knowledge of the task in hand, independent of the environment in which that task will be accomplished. No fixed order is assumed over how user goals will be discharged.

To see how this is modelled in SAL consider the following guarded command *GoalTrans* for doing a user action that has been committed to:

$$\texttt{gcommit[i]} = \texttt{committed} \quad \rightarrow \quad \begin{array}{l} \texttt{gcommit}'\texttt{[i]} = \texttt{done};\ \ \texttt{gcommitted}' = \texttt{FALSE}; \\ GoalTransition(i) \end{array}$$

The left-hand side of $\rightarrow$ is the guard of this command. It says that the rule will only activate if the associated action has already been committed to, as indicated by the `i`-th element of the local variable array `gcommit` holding value `committed`. If the rule is then non-deterministically chosen to fire, this value will be changed to `done` and the boolean variable `gcommitted` is set to false to indicate there are now no commitments to physical actions outstanding and the user model can select another goal. $GoalTransition(i)$ defines the state update transitions associated with this particular action i.

User goals are modelled as an array `goals` which is a parameter of the `User` module. The user model state space consists of three parts: input variable `in`, output variable `out`, local variable (memory) `mem`; environment is modelled by global variable `env`. All of these are specified using type variables and are instantiated for each concrete interactive system. Each goal is specified by a record with the fields `grd`, `tout`, `tmem` and `tenv`. The `grd` field is discussed below. The remaining fields are relations from old to new states that describe how two components of the user model state, outputs `out` and memory `mem`, and environment `env` are updated by discharging this goal. These relations, provided when the generic user model is instantiated, are used to specify $GoalTransition(i)$ as follows:

```
out' ∈ {x:Out | goals[i].tout(in,out,mem)(x)};
mem' ∈ {x:Memory | goals[i].tmem(in,mem)(x)};
env' ∈ {x:Env | goals[i].tenv(in,mem,env)(x) ∧ possessions}
```

The update of `env` must also satisfy a generic relation, *possessions*. It specifies universal physical constraints on possessions, linking the events of taking and giving up a possession with the corresponding increase or decrease in the number of objects possessed. This number is modelled as an environment component.

If the guarded command for *committing* to a user goal (given below) fires, it switches the commit flag for goal i to `committed` allowing the above rule to become

active. The predicate `grd`, extracted from the `goals` parameter, specifies when there are opportunities to discharge this user goal. Because we assign `done` to the corresponding element of the array `gcommit` in the *GoalTrans* command, once fired the command below will not execute again. If the user model discharges a goal, without some additional reason such as a prompt, it will not do so again.

$$\begin{array}{l} \texttt{gcommit[i]} = \texttt{ready} \ \wedge \ \texttt{NOT(gcommitted} \vee \texttt{rcommited)} \\ \wedge \ \texttt{finished} = \texttt{notf} \ \wedge \ \texttt{goals[i].grd(in, mem, env)} \end{array} \rightarrow \begin{array}{l} \texttt{gcommit}'\texttt{[i]} = \texttt{committed}; \\ \texttt{gcommitted}' = \texttt{TRUE} \end{array}$$

Reactive behaviour A user may react to an external stimulus, doing the action suggested by the stimulus. For example, if a flashing light comes on a user might, if the light is noticed, react by inserting coins in an adjacent slot. Reactive actions are modelled in the same way as user goals but on different variables, e.g. parameter `react` of the `User` module rather than `goals`. $ReactTransition(i)$ is specified in the same way as $GoalTransition(i)$. The array element `rcommit[i]` is reassigned `ready` rather than `done`, once the corresponding action has been executed, as reactive actions, if prompted, *may* be repeated.

Goal based task completion Users intermittently, but persistently, terminate interactions as soon as their perceived goal has been achieved [2], even if subsidiary tasks generated in achieving the main goal have not been completed. A cash-point example is a person walking away with the cash but leaving the card.

In the SAL specification, a condition that the user perceives as the main goal of the interaction is represented by a parameter `PerceivedGoal` of the `User` module. Goal based completion is then modelled as the guarded command *Exit*, which simply states that, once the predicate `PerceivedGoal` becomes true and there are no commitments to user goals and/or reactive actions, the user may complete the interaction. This action may still not be taken because the choice between enabled guarded commands is non-deterministic. Task completion is modelled by setting the local variable `finished` to `ok`:

$$\begin{array}{l} \texttt{PerceivedGoal(in, mem)} \ \wedge \ \texttt{finished} = \texttt{notf} \\ \wedge \ \texttt{NOT(gcommitted} \vee \texttt{rcommitted)} \end{array} \rightarrow \quad \texttt{finished}' = \texttt{ok}$$

No-option based task termination If there is no apparent action that a person can take that will help complete the task then the person may terminate the interaction. For example, if, on a 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 the goal is not achievable.

In the SAL specification, the no-option condition is expressed as the negation of predicates `EnabledGoals` and `EnabledReact`. Note that, in such a situation, a possible action that a person could take is to wait. However, they will only do so given some cognitively plausible reason such as a displayed "please wait". The waiting conditions are represented in the `User` module by predicate parameter `Wait`. If `Wait` is false, `finished` is set to `abort` in the guarded command *Abort*.

3 Formal Specification of User Interpretation for an ATM

The separation of user and device state spaces means connectors are required to compose the user and device models (recall Fig. 1). If the state spaces of

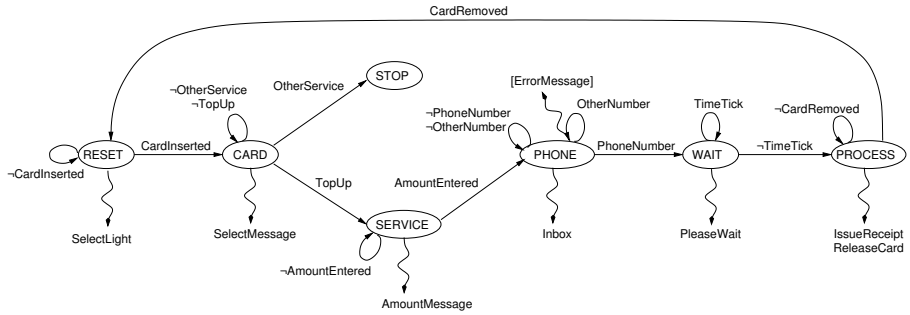

Fig. 2. ATM specification as finite state machine

both models precisely match, these connectors are simply identity mappings. This would yield essentially the same situation as with the shared state space. However, the separated state spaces open up new possibilities for specifying more complex connectors. These allow the formal modelling of the interpretation processes that are occurring in the interaction between the user and the device.

In this section, we consider the user interpretation part of a specific interactive system. Considering such concrete examples will help us to develop in the future an abstract model (as with the user model itself) of user interpretation. We start by specifying the machine and user models of this system.

We use here the task of topping-up a mobile phone based on a real ATM. A finite state machine specification of this device is given in Fig. 2. False machine outputs are omitted. The actual SAL specification, module `ATM`, is a straightforward translation of this diagram. Since our focus is the top-up task, we omit the specification of other services provided (this corresponds to the diagram's `STOP` state). Also, as we are illustrating the modelling of user interpretation, in this paper we abstract the authentication process by assuming that PIN entering/verification is a part of the card insertion step.

According to our specification, the ATM initially prompts users to insert a card. Once this is done, the machine provides several touch screen menu options. We assume (and specify in the user model later on) that the user chooses the top-up option. The machine then displays a new menu with several options to select the top-up value; the user can choose any. In response, the machine displays an input box and asks for a phone number. The user interpretation of this request is discussed in detail below. For now it suffices to know that the interpretation can result in two actions: entering a phone number, or entering some other number (we assume the machine can distinguish these two alternatives). In the former case, a receipt is issued and the card is released; in the latter, the machine displays an error message and again prompts for a phone number. The transactions related to the actual top-up process take time. Thus a "please wait" message is displayed during processing. Finally, the machine returns to the initial state once the released card is removed.

The input and output components of the device state space are relevant to the discussion of user interpretation. The input variables of our specification are `CardInserted`, `TopUp`, `OtherService`, `AmountSelected`, `PhoneNumber`, `OtherNumber`, and `CardRemoved` (Fig. 2). The output variables are `CardMessage`, `SelectMessage`, `AmountMessage`, `Inbox`, `ErrorMessage`, `PleaseWait`, `IssueReceipt`, and `ReleaseCard`. These variables are booleans, except `Inbox`. It is a record consisting of two fields: `option` and `size`. The former specifies whether the request to enter a phone number is displayed, the latter is the size of the input box.

The generic user model `User` was described in Sect. 2. To analyse our interactive system, we now instantiate that model for the concrete task of topping-up a mobile phone. We start by specifying the state space of our user model.

In general, it is plausible to assume that the specific details of an ATM specification might be unavailable at the time the concrete user model is developed. Even if they are, it could be preferable to specify the user state space in more cognitive terms, not constraining oneself by the existing device specification. First, we consider user perceptions which are represented in the `User` module by the input variable `in`. We assume that the user is able to perceive the following signals from the machine: `InsertCard`, `SelectService`, `SelectAmount`, `RemoveCard`, `WaitMessage`, and `ErrorMessage` (their names should be self-explanatory). The user can also perceive the shape of the input box, `InboxShape`. People usually know their phone numbers, however, they might also have another (different) number on their top-up cards. It is cognitively plausible that the user may be uncertain which number is requested. This confusion is represented in the user model by two distinct components, `EnterPhoneNumber` and `EnterCardNumber`. Finally, the user evaluates the state of the machine deciding whether the requested service has been received, modelled by `ServiceReceived`. These components form a record, `In`, which is used to instantiate the corresponding type variable in `User`.

Next, we specify state space components related to the actions users might take. These correspond to the ATM inputs in Fig. 2 and are: `CardInserted`, `TopUp`, `OtherService`, `AmountSelected`, `PhoneNumber`, `OtherNumber`, and `CardRemoved`. Like the user inputs above, these components form a record, `Out`. For this paper, the memory component of the `User` module, `Mem`, is kept simple. We assume the user remembers only the actions taken in the previous step. `Mem` is therefore the same record type as `Out`. Finally, user actions can both affect and be restricted by the environment of our system; we thus have a record type, `Env`. It includes counters, `BankCards` and `PhoneCards`, for the user possessions (cards); values (the balances of the corresponding accounts) of these possessions, `BankBalance` and `PhoneBalance`; and the sizes, `SizePhone` and `SizeCard`, of the card numbers.

We assume that user knowledge of ATM transactions includes the need to (1) provide a payment card, (2) communicate that the top-up option is required and (3) communicate the top-up value. This knowledge is specified as user goals (elements of array `goals`) instantiated by giving the action guard and the update to the output component. For the insert-card goal, the guard is that the person perceives an `InsertCard` signal and has a bank card. The output action is to set `CardInserted` to true (`Default` is a record with all its fields set to false so asserts that nothing else is done). We omit the memory and environment updates:

```
grd  := λ(in,mem,env): in.InsertCard ∧ env.BankCards > 0
tout := λ(in,out0,mem): λ(out): out = Default WITH .CardInserted := TRUE
```

Choosing to top-up and communicating the top-up value are modelled similarly.

We assume that the user can *reactively* respond to device prompts by attending to either spatial or semantic cues (or both) and enter the phone number. This may happen only when the machine state is interpreted as signalling to enter the number by `in.EnterPhoneNumber`. The number must also not have been entered, as indicated by the memory, in the previous step, unless the person sees an error message requesting that repetition. Formally, the action is specified as follows:

```
grd := λ(in,mem,env): in.EnterPhoneNumber ∧
                      (NOT(mem.PhoneNumber) ∨ ErrorMessage)
tout :=λ(in,out0,mem): λ(out): out = Default WITH .NumberEntered := TRUE
```

However, as discussed earlier, it is plausible that a prompt for the phone number can be misinterpreted as that for the number on the top-up card instead (a semantic cue). The corresponding reactive action is analogous to the one above. Finally, the user can respond to the prompt for taking back their card:

```
grd  := λ(in,mem,env): in.RemoveCard ∧ NOT(mem.CardRemoved)
tout := λ(in,out0,mem): λ(out): out = Default WITH .CardRemoved := TRUE
```

Goal and wait predicates are the last parameters used to instantiate the `User` module. We assume that the user considers receiving the requested service as the main goal of the interaction. We also assume that seeing a "please wait" message is the only condition when the user, perceiving no options to act, does not terminate the interaction. The two predicates are specified in SAL as follows:

```
PerceivedGoal  =  λ(in,mem): in.ServiceReceived
Wait           =  λ(in,mem): in.WaitMessage
```

Finally, the ATM user model, `ATMuser`, is defined by instantiating the generic user model with the parameters (goals, reactive actions, perceived goal and wait conditon) just defined.

So far we have specified an ATM and have developed a formal model of its user. The state spaces of the two specifications are distinct. This closely corresponds to reality, since the state of an ATM and the user interpretation of it are not necessarily identical. The changing machine state is first attended to then interpreted by the user. Next we formally specify this interpretation, thus providing a connector for separate state spaces. The specification is written as a new SAL module, `interpretation`. The module, being a connector, has input variables that are the output variables of `ATM`, and output variable that is the record `in`, the input (perception) component of the `User` module.

We model user interpretation (below) by the SAL definition construct which allows one to describe system invariants. Intuitively, this means that the left-hand side of an equation is updated whenever the value of the right-hand side changes. Here, we assume that the user model directly perceives some of the ATM signals such as prompts for inserting a card, selecting amount, a wait message, etc. Consequently, the first seven conjuncts in the definition are simple renamings of the appropriate fields in the record `in` to the corresponding variables in `ATM`. Below we discuss in more detail the omitted parts of the final three conjuncts.

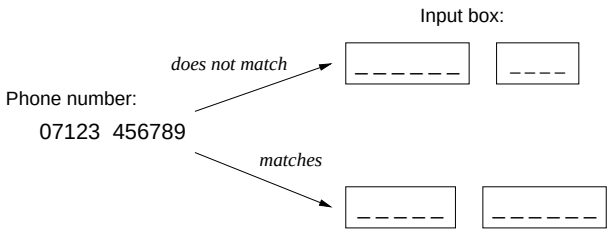

Fig. 3. User interpretation of input boxes.

```
DEFINITION in ∈ {x:In |
   x.WaitMessage = PleaseWait ∧ x.ErrorMessage = ErrorMessage ∧
   x.InsertCard = CardMessage ∧ x.SelectService = SelectMessage ∧
   x.SelectAmount = AmountMessage ∧ x.RemoveCard = ReleaseCard ∧
   x.ServiceReceived = IssueReceipt ∧ x.InboxShape = ... ∧
   definition of x.EnterPhoneNumber, x.EnterCardNumber }
```

As explained earlier, the field `ServiceReceived` corresponds to the active goal in our user model. The definition above identifies it with the machine action issuing a receipt. Of course, getting a receipt could not plausibly be the actual user goal, a better candidate for which is to have the mobile phone account increased by the desired amount. The latter, however, is impossible to observe directly, so, with this machine, getting a receipt is the best available approximation.

In this paper, we consider what influence the shape of a machine prompt can have on user interpretation of it. For this, we use input boxes displayed by ATMs (see Fig. 3). There could be many aspects of the shape to investigate; for simplicity, the shape of an input box is modelled as its size in our case studies. In general, however, it could represent any relevant aspect of shape. The definition below identifies shape with size; the condition `Inbox.option` ensures that this identification occurs only when an input box is displayed, otherwise, the user model does not perceive the box at all, as represented by the shape value `0`:

```
x.InboxShape = IF Inbox.option THEN Inbox.size ELSE 0 ENDIF
```

The penultimate conjunct in the definition illustrates how shape can affect user interpretation of machine prompts. We present it in four parts below. The first part (conjunct) specifies the situation when there is nothing in the machine state that could be interpreted by the user model as a prompt, or spatial cue, for entering a phone number or the number on a top-up card. This happens when the shape of the input box (possibly not displayed at all) matches neither of the numbers the user could consider as relevant to the prompt:

```
x.InboxShape ≠ env.SizePhone ∧ Inbox.size ≠ env.SizeCard
⇒ NOT(x.EnterPhoneNumber) ∧ NOT(x.EnterCardNumber)
```

When the shape (size) of the displayed input box matches the phone number and is different from the number on the card, we assume that the user model interprets this as a prompt, or semantic cue, for entering the phone number:

```
x.InboxShape = env.SizePhone ∧ Inbox.size ≠ env.SizeCard
⇒ x.EnterPhoneNumber ∧ NOT(x.EnterCardNumber)
```

Analogously, the user model can interpret the machine state as a prompt for the number on the top-up card:

```
x.InboxShape ≠ env.SizePhone ∧ Inbox.size = env.SizeCard
⇒ NOT(x.EnterPhoneNumber) ∧ x.EnterCardNumber
```

Finally, the user can be confused as to which of the two numbers is requested. This may happen when the shape of the displayed box matches both numbers. We assume that the result of user interpretation in this case is non-deterministic, but only one interpretation is allowed (here `XOR` is exclusive-or):

```
x.InboxShape = env.SizePhone ∧ Inbox.size = env.SizeCard
⇒ x.EnterPhoneNumber XOR x.EnterCardNumber
```

Now we have specified all the components of our interactive system. The whole system, `system`, is modelled in SAL as their parallel composition:

```
(ATMuser [] ATM [] environment) || (interpretation || effect)
```

Here, `[]` denotes asynchronous (interleaving) composition, whereas `||` denotes synchronous composition. For brevity, we have not presented the specifications of the `effect` and `environment` modules. Informally, the `effect` module specifies how user actions from `ATMuser` are translated into the machine commands; in other words, how the output component `out` is connected to the `ATM` inputs. In our case study, the translation is simple renaming, analogous to those given in the definition of `in` above. The `environment` module contains no transitions; it simply defines constants such as the size of the phone and top-up card numbers.

4 Verification of Interactive Systems

We now present verification examples, focussing on the system aspects influencing user interpretation. We first introduce system properties to verify. Our approach is concerned with two kinds of correctness properties. Firstly, we want to be sure that, in any possible system behaviour, the user's main goal of interaction is eventually achieved. Given our model's state space, this is written in SAL as the assertion (where `F` means 'eventually'): `F(PerceivedGoal(in,mem))`. Second, in achieving a goal, subsidiary tasks are often generated that the user must complete to complete the task associated with their goal. If the completion of these subsidiary tasks is represented as a predicate, `SecondaryGoal`, the required condition is specified as: `G(PerceivedGoal(in,mem) ⇒ F(SecondaryGoal(in,mem,env)))` (where `G` means 'always'). This states that the secondary goal is always eventually achieved once the perceived goal has been achieved. Often secondary goals can be expressed as interaction invariants [3] which state that some property of the model state, that was perturbed to achieve the main goal, is restored.

In the first example, the ATM design's displayed input box has shape (size) larger than strictly needed and it (incorrectly) matches the top-up card number but not the correct but shorter phone number. Our first attempt is to verify that the user model eventually achieves the perceived goal of getting a receipt. Unfortunately, the desired property is not true, and the SAL model checker produces

a counterexample which shows both the trace of system states and the actions taken by the user model and the machine. The analysis of the counterexample indicates that the user model loops on entering the top-up card number. Further analysis reveals that this looping is due to the user (mis)interpreting the ATM prompt for entering the phone number as that for the card number. This misinterpretation is caused by the shape of the input box, which matches the card number. Of course, this does not mean that every real ATM user is prone to such misinterpretation or would loop forever. However, the assumptions on which our specification of user interpretation is based are cognitively plausible and this is a systematic consequence of them. Therefore, some users are liable to make this error and changes in the ATM design are advisable.

Next, we consider a modified ATM design in which the shape of the displayed box matches the phone number. We assume here that the shape of the card number is different. Now the first correctness property, the user eventually achieving the perceived goal, is satisfied by the interactive system. We thus proceed with the verification of the second property that the user eventually achieves the secondary goal. This is expressed as an interaction invariant, which states that the total value of the user possessions (the balance of the account associated with the payment card plus the top-up card balance) is eventually restored to the initial value. Unfortunately, the verification of this property fails. The counterexample produced indicates that the failure is caused by the user model finishing the transaction as soon as a receipt is issued. Detecting this type of user error, a post-completion error, with its underlying causes and possible remedies, has been discussed in our earlier papers [3]. Here, we just note that such errors could be eliminated by modifying the ATM from Fig. 2 so that a receipt is issued only when the user has removed the card.

Finally, consider the case when the phone and card number both match the shape of the displayed box. The verification of the first correctness property fails. The counterexample produced is as in the first example (when only the card number matched the displayed box). Further analysis reveals that, unlike in that example, the user model can now achieve the perceived goal. Within the SAL environment, this is verified using the assertion `EF(PerceivedGoal(in,mem))`, where the operator `EF` states that there is a path such that the corresponding formula is eventually true. This indicates that both user interpretations of the machine prompt are possible, which can lead to the confusion of real ATM users.

5 Summary and Further Work

We have presented a refined version of our cognitive architecture. The state space of the formal user model has been separated from that of the device. This both required and facilitated the abstract modelling of user interpretation of device outputs. We presented a simple case study. It illustrates how such abstract models can be used within our verification approach to detect problems in interactive systems related to shape induced confusion over device signal meaning. Our abstract model is a simplification of cognitive interpretation, and clearly not every

user of such a device will experience the problems our approach reveals. However, since the abstraction is cognitively plausible, a strong potential for user confusion is there, and a substantial number of users might experience it.

As SAL provides support for higher-order specifications, the cognitive architecture remains generic, and is instantiated to verify any specific interactive system. Besides the major restructuring described here, the treatment of the underlying state space is simplified in the SAL version where simple variables are used instead of the history functions of the original. Since theorem provers are better for reasoning about abstract properties than concrete ones as here, the ideal is to have an integrated theorem prover/model checker framework. Being developed as a framework for combining various verification tools, including the PVS theorem prover, SAL is a very promising environment for the future development of our verification methodology.

For simple systems as considered here, mechanical verification is unnecessary. The problems detected could be identified by examining the specification of user interpretation. Still, writing a formal specification helps to identify problems, and our framework provides structure to the specification process. Moreover, a combination of several user interpretation pathways would lead to complex specifications, requiring mechanical verification. Finally, the verification of specific systems is only a part of a larger verification framework where the formal specification of user interpretation could be combined with, say, design rules to reason about general properties of interactive systems using a theorem prover.

Our cognitive model was not developed directly from a complete psychological theory. Rather an exploratory approach was taken, starting with simple principles of cognition such as non-determinism, goal-based termination and reactive behaviour. However, even the small number of principles is rich enough for plausible erroneous behaviour to emerge that was not directly expected [3].

Other aspects of user interpretation remain to be investigated. An ability to combine information in-the-world with knowledge in-the-head allows individuals to make interpretations. Mandler [20], amongst others, argues that knowledge can be accessed directly or indirectly. When interaction relies on novel associations it is likely to demand more direct attention. However, frequent and familiar interactions use indirect knowledge that involves interpretation. Further work needs to identify when user interpretations are made, what types of spatial and semantic cues are used (see Dillon [5]), and if these can be modelled.

The user interpretation errors detected occur for systematic reasons. A generic model of user interpretation can capture these systematic reasons, thus helping to eliminate errors they cause. While the cognitive architecture is generic, the specification of user interpretation currently is not. It must be written for each specific system from scratch. Considering other aspects of user interpretation will facilitate the development of a generic interpretation model. Finally, we will also investigate the formal modelling of *effect*, the counterpart of user interpretation. It is especially relevant for multimedia based interactive systems. We expect that our changes to and reorganisation of the model will facilitate such modelling, as evidenced by the aspect of user interpretation considered here.

Acknowledgments This research is funded by EPSRC grants GR/S67494/01 and GR/S67500/01.

References

1. Reason, J.: Human Error. Cambridge University Press (1990)
2. Byrne, M.D., Bovair, S.: A working memory model of a common procedural error. Cognitive Science **21**(1) (1997) 31–61
3. Curzon, P., Blandford, A.E.: Detecting multiple classes of user errors. In Little, R., Nigay, L., eds.: Proc. of the 8th IFIP Work. Conf. on Engineering for Human-Computer Interaction. Volume 2254 of LNCS, Springer-Verlag (2001) 57–71
4. Blandford, A.E., Young, R.M.: Separating user and device descriptions for modelling interactive problem solving. In Nordby, K., Helmersen, P., Gilmore, D., Arnesen, S., eds.: INTERACT'95, Chapman and Hall (1995) 91–96
5. Dillon, A.: Spatial-semantics: how users derive shape from information space. Journal of the American Society for Information Science **51** (2000) 521–528
6. Dillon, A., Schaap, D.: Expertise and the perception of shape in information. Journal of the American Society for Information Science **47** (1996) 786–788
7. Tulving, E.: Elements of Episodic Memory. Oxford University Press (1983)
8. Campos, J.C., Harrison, M.D.: Formally verifying interactive systems: a review. In Harrison, M., Torres, J., eds.: Design, Specification and Verification of Interactive Systems '97, Springer-Verlag (1997) 109–124
9. Markopoulos, P., Johnson, P., Rowson, J.: Formal architectural abstractions for interactive software. Int. Journal of Human Computer Studies **49** (1998) 679–715
10. Duke, D.J., Duce, D.A.: The formalization of a cognitive architecture and its application to reasoning about human computer interaction. Formal Aspects of Computing **11** (1999) 665–689
11. Bowman, H., Faconti, G.: Analysing cognitive behaviour using LOTOS and Mexitl. Formal Aspects of Computing **11** (1999) 132–159
12. Moher, T.G., Dirda, V.: Revising mental models to accommodate expectation failures in human-computer dialogues. In: Design, Specification and Verification of Interactive Systems '95, Springer-Verlag (1995) 76–92
13. Rushby, J.: Analyzing cockpit interfaces using formal methods. Electronic Notes in Theoretical Computer Science **43** (2001)
14. Campos, J.C., Doherty, G.J.: Reasoning about dynamic information displays. In Jorge, J.A., Nunes, N.J., e Cunha, J.F., eds.: Interactive Systems. Design, Specification, and Verification, Volume 2844 of LNCS. Springer-Verlag (2003) 288–302
15. Bredereke, J., Lankenau, A.: A rigorous view of mode confusion. In: Proc. of SAFECOMP 2002. Volume 2434 of LNCS, Springer-Verlag (2002) 19–31
16. Cerone, A., Lindsay, P.A., Connelly, S.: Formal analysis of human-computer interaction using model-checking. In: Proc. of the Third IEEE Int. Conference on Software Engineering and Formal Methods (SEFM05), IEEE Press (2005) 352–362
17. Butterworth, R.J., Blandford, A.E., Duke, D.J.: Demonstrating the cognitive plausibility of interactive systems. Formal Aspects of Computing **12** (2000) 237–259
18. de Moura, L., Owre, S., Ruess, H., Rushby, J., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In Alur, R., Peled, D.A., eds.: Computer Aided Verification: CAV 2004. Volume 3114 of LNCS, Springer-Verlag (2004) 496–500
19. Newell, A.: Unified Theories of Cognition. Harvard University Press (1990)
20. Mandler, G.: Recognizing: the judgment of previous occurrence. Psychological Review **87** (1980) 252–271