Modelling Rational User Behaviour as Games between an Angel and a Demon

Rimvydas Rukšėnas, Paul Curzon
Queen Mary, University of London
Department of Computer Science
Mile End Road, London E1 4NS, UK
rimvydas@dcs.qmul.ac.uk, pc@dcs.qmul.ac.uk

Ann Blandford
University College London
UCL Interaction Centre
Gower Street, London WC1E 6BT, UK
A.Blandford@ucl.ac.uk

Abstract

Formal models of rational user behaviour are essential for user-centred reasoning about interactive systems. At an abstract level, planned behaviour and reactive behaviour are two important aspects of the rational behaviour of users for which existing cognitive modelling approaches are too detailed. In this paper, we propose a novel treatment of these aspects within our formal framework of cognitively plausible behaviour. We develop an abstract, formal model of rational behaviour as a game between two opponents. Intuitively, an Angel abstractly represents the planning aspects, whereas a Demon represents the reactive aspects of user behaviour. The formalisation is carried out within the MOCHA *framework and is illustrated by simple examples of interactive tasks.*

1. Introduction

We use interactive computer systems for online shopping, banking, or interacting on social networking sites. Since such systems have become ubiquitous in modern society, they must be usable, secure and provide intended services. Interactivity however implies that these systems' properties do not solely depend on their software components, but also on the actions of their human users. One way to address this when verifying interactive systems is to combine the formal models of both the system (device) and aspects of its user behaviour [9, 10].

Our approach based on this idea has been successfully used to detect a range of systematic user errors related to, e.g., functional correctness [8] and security [20]. The approach relies on a generic user model (abstract cognitive architecture) [7, 19] that formalises abstract cognitive principles, such as a user entering an interaction with knowledge of the task and its subsidiary goals, and choosing non-deterministically between appropriate actions. While successful, the approach is limited by the assumption that the non-determinism in our current user model might only be resolved *demonically*, i.e., in the worst possible way. This is a plausible and useful assumption in some situations (e.g., reactive behaviour of novice users), however it does not easily capture more advanced aspects of user behaviour (e.g., aspects of expert behaviour).

More generally, the rational behaviour of humans includes both planned behaviour and reactive behaviour [6]. Similar views are also held in various agent-based formalisms [23]. Both planned and reactive behaviour aim to reduce the distance between the user goals and the current state of an interactive system. With planned behaviour, this is achieved by moving the goals towards the current state, whereas, with reactive behaviour, the current state is moved towards the goal state by taking immediately doable actions.

In this paper, we argue that *angelically* (i.e., in the best possible way) resolved non-determinism provides an alternative and more abstract way of formally modelling the planning aspects of rational user behaviour. It works for both immediate, and thus relatively simple, choices between available alternatives and more advanced aspects such as ordering sequences of necessary actions. The simplicity of this approach is achieved by explicitly modelling only task-specific user actions, whereas the construction and analysis of the "planning space" is implicitly carried out by the verification tool as part of the model checking process. This allows us to focus on the area of interest of our work, systematic errors induced by a design, rather than problems caused by the planning behaviour of individuals.

As a result, and in contrast to cognitive modelling, e.g. [15, 12], our approach does not deal with the planning process *per se*. Instead, we can focus on the analysis of whether the user behaviour, including planning aspects, satisfies properties such as avoiding systematic errors when performing an interactive task. We also show that care is needed in relying on angelic non-determinism so that the user modelled does not become an omniscient deity capable of overcoming all the flaws, imperfections and assumptions in the designs of user interfaces, thereby making the

978-0-7695-3437-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SEFM.2008.43

verification results meaningless and potentially misleading.

We formalise these ideas as a *generic* cognitive architecture. It is built as a collection of agents where each agent represents a different aspect of user behaviour, e.g., a reactive agent, a planning agent, etc. We then interpret this cognitive architecture as a *game* between cooperating and/or competing agents.

For verification purposes, all agents are divided into two coalitions: *our* agents represent the angelic aspects of user behaviour, whereas the *other* agents represent the demonic aspects. Thus we associate, e.g., the planning agent with our coalition and the reactive agent with the other coalition. The membership of some agents might depend on the user type and/or particular scenarios. For example, in the case of well-trained expert users, it is cognitively plausible to assume that the "mode" agent making the choice between the reactive and the planning modes of behaviour is angelic, since such users are capable of planning their actions so that the main task goal is achieved. On the other hand, novice users might rely on the reactive behaviour based on interface cues. Thus, their behaviour is more precisely captured by viewing the mode agent as demonic.

Summarising, the main contribution of this work is an abstract model of rational user behaviour based on an appropriate combination of angelic and demonic non-determinism. This cognitive architecture extends our earlier work that relied on a solely demonic interpretation of non-determinism in the user model. The extension (a) makes the analysis based on a new model and its predictions more precise and reliable, and (b) widens the range of interactive scenarios that can be formally captured. At the same time, the simplicity of our model facilitates the formal specifications of interactive systems and should make their verification more viable in practice.

2. Problem

The basic idea of our approach is to develop an abstract model of rational user behaviour that (a) is based on a combination of the reactive and planned behaviour, and (b) is sufficiently simple to be used for the (automated) reasoning and verification of interactive systems. Intuitively, the angelic resolution of the available choices of action can be viewed as an abstract representation of planned behaviour, whereas the demonic resolution corresponds to reactive behaviour. It turns out, however, that even a choice between several simple alternatives cannot be viewed as inherently demonic or angelic.

In this section we discuss in detail this research challenge. Namely, we show that neither demonic nor angelic non-determinism alone can adequately model the rational choices of humans that are related to non-deliberate interaction faults in taxonomy of dependable computing [2]. To highlight the essence of the problem without cluttering it with irrelevant details, we use a simple yet realistic task scenario involving a cash point (ATM). Then we outline our solution to the discussed problem. A formal model of rational user behaviour is developed in the next section, based on these observations.

```
module ATM1
 external choice : {none, balance, withdraw}
 interface menu, show, release, wait : bool
atom ATMinterface
 controls menu, show, release, wait
 reads choice, menu, show, release, wait
 awaits choice
init
[] true -> show' := false; release' := false;
           menu' := true; wait' := false
update
[] ~(choice' = none) ->
        wait' := true; menu' := false
[] choice = balance ->
        show' := true; release' := false;
        menu' := true; wait' := false
[] choice = withdraw ->
        show' := false; release' := true;
        wait' := false
[] default ->
        show' := false; release' := false;
        wait' := false
endatom
endmodule
```

Figure 1. ATM specification.

Reactive modules [3] are used as the formal modelling framework. The main reason for this is that the verification environment based on it, MOCHA [3], provides a natural way of specifying agents and their coalitions, thus supporting game-based modelling. We will explain the notation and its meaning as we describe the example.

2.1. Example system: a cash point

We assume that our cash point provides two menu options: cash withdrawal and balance check. If the balance option is selected, the machine shows the balance information on the display. If the withdraw option is selected, the requested amount of cash is provided. A MOCHA specification of this machine is given in Fig. 1 as the module ATM1. In this paper, for ease of comprehension, we omit from our specification authentication steps (inserting a card and entering its PIN) and abstract away from modelling how the desired amount of cash is selected.

The boolean variables menu, show, release and wait specify whether the corresponding elements of ATM interface—menu choices, balance information, released cash and wait message—are displayed or otherwise available. They are output variables and can be read by the environment (other modules). The input variable choice specifies the two possible choices that users can make (or the absence of any choice). It may only be read by the module.

Mocha modules consists of *atoms*. An atom defines the next-state relation for the variables it controls. This relation is specified as a set of guarded commands: an action may only be taken when its guard is true. For example, the guarded command from Fig. 1

```
~(choice'= none) -> wait':=true; menu':=false
```

is executed only if the environment (user) has chosen one of the menu options. In this case, the next values (primed variables) of wait and menu specify that a wait message is displayed at the same time as menu choices become unavailable. The next two commands in Fig. 1 specify appropriate ATM responses—displaying the balance and releasing cash—to the user's menu selections. Note that the choice between guarded commands is *non-deterministic*: if several guards are true, any of those commands may be executed. Finally, the guard default indicates that this action is taken when all the other guards are false.

The initial values of variables are computed as specified by the *init* sections of the atoms; their subsequent updates are specified by the *update* sections. The variables in different atoms are updated in one round in an arbitrary order. However, if an atom *awaits* some variables (choice in our example), then the variables this atom controls are updated after those it awaits. Thereby, the next values of the awaited variables can be used in a computation round. For example, the next value of choice used in the guard of the above command ensures that it is evaluated once the user model has exercised an opportunity to choose one of the menu alternatives.

Next we specify the user model for this ATM.

User model. We assume that the user model (see Fig. 2) may take four actions: choose either the balance or withdraw option from menu, read the displayed balance information and take the released cash. If none of these options is available, the user model simply waits by doing nothing. Reading balance information and taking cash are modelled as the boolean variables read and take. The other variables used by the module ATMuser were explained above.

As can be seen from this specification, the user model is non-deterministic—it can choose between the balance and withdraw options. We show next that there is no simple answer to the question of what is the most appropriate way to interpret this non-determinism.

```
module ATMuser
 external menu, show, release : bool
 interface choice : {none,balance,withdraw};
           read, take : bool
atom UserActions
 controls choice, read, take
 reads menu,show,release, choice,read,take
init
[] true -> choice' := none;
           read' := false; take' := false
update
[] menu & ~show & ~release ->
                      choice' := balance
[] menu & ~show & ~release ->
                      choice' := withdraw
[] show -> read' := true; choice' := none
[] release -> take' := true; choice' := none
[] default -> choice' := none
endatom
endmodule
```

Figure 2. Simple ATM user model.

2.2. Modelling non-determinism

Angelic choice. Let us consider the *angelic* interpretation first—the user is sufficiently smart to choose an alternative that is relevant to the goal they try to achieve. At first, it may seem that such a commonsensical assumption works as intended. For example, consider an interaction the goal of which is to withdraw cash. The corresponding assertion in Mocha is expressed as the following formula in *alternating temporal logic* (ATL) [1] where the variable take being true specifies that the user has taken cash:

```
<< ATMuser >> F(take)          (1)
```

Here the deviation from the analogous CTL formula is that the path quantifier (angle brackets) is parametrised by the atoms from ATMuser. Thus the above formula is true, if the atom UserActions can *eventually* (operator F) take the system into a state such that take is true, no matter how the rest of the system behaves. In other words, non-determinism in the atoms of ATMuser is interpreted in formula (1) angelically.

As one would expect, model checking shows that (1) is true for the parallel composition of ATM1 and ATMuser: the user model is able to withdraw cash using this ATM. Consider next a slightly more complicated task goal—both to check balance and withdraw cash (for simplicity we assume that the user does not care about the order of these actions):

```
<< ATMuser >> F(read & take)          (2)
```

Again, model checking shows that (2) is true. Unfortunately, this time the positive outcome of verification is actually undesirable, since it hides an important property of the interface design specified by `ATM1`. The latter is restrictive in the sense that it allows one to check balance and withdraw cash during the same interaction only if the first choice was the balance option. Otherwise, once cash has been released, the menu is no longer displayed making it impossible for the user to check the balance. However, the angelic interpretation of non-determinism allows the user model to overcome this design feature by guessing the "right" order of actions and always taking it.

Such an outcome of user-centred verification would result in misleading conclusions about the usability of this ATM design, since there is no cognitively plausible reason to expect that a real person would favour any of the menu options in this situation.

Next we look at what happens if one takes the demonic view of users.

Demonic choice. The demonic interpretation is at the basis of our earlier work on generic user models [7, 19]. Its justification relies on the fact that, in the same interactive situations, people behave differently, depending on their previous experience, the salience of device signals, etc., as long as their actions are cognitively plausible. Thus, to be able to detect a wider range of potential interaction problems, one has to assume the worst possible scenario. Essentially, this amounts to asking—is it cognitively plausible that someone will eventually take a "wrong" action for systematic reasons?

Such an approach solves the problem of the omniscient user we identified above. Namely, the demonic analog of formula (2) where `ATMuser` no longer necessarily cooperates in reaching the desired state

$$\ll \gg \texttt{F(read \& take)} \tag{3}$$

is false. Thus, the demonic view of non-determinism allows one to detect the potentially problematic aspect of our ATM design. Unfortunately, it introduces a new problem—the demonic analog of the previously true formula (1)

$$\ll \gg \texttt{F(take)} \tag{4}$$

is false. The analysis of our interactive system shows that the user model now chooses (repeatedly) the worst possible course of action, the balance menu option, instead of doing the "right" thing—choosing the withdrawal option. Such model behaviour of course does not reflect reality: normal users do not maliciously behave contrary to their task goal. The observed behaviour is simply an artefact of our modelling choice to resolve non-deterministic situations demonically.

Next we outline our solution to the discussed problem.

2.3. Towards a solution

The main problem with the simple approach discussed so far is that it takes a fixed view (angelic or demonic) of users. However, the nature of user choices in our example actually depends on the task goal. More generally, human behaviour varies within the population and depends on context and situation [22]. To take this into account while still retaining our abstract view of the planned and reactive behaviour, we model user choice as the following two-step, *decision* and *action*, process.

The decision step is angelic. Our user model decides which available actions are or have become no longer *relevant* to the task under execution and eliminates them from further consideration. The angelic decision about the relevance of actions represents abstractly the human ability and propensity to act according to the context and situation. When the model operates in the planning mode, it also selects an action to take. The angelic selection of the next action represents abstractly the human ability to make plans. To counterbalance the angelic power so that slips are not eliminated from planned behaviour, only *predictable* (see Sect. 3) and *salient* actions can be selected in this step within our approach. The salience issues, however, are not dealt with here (see our earlier work [18]).

The action step is demonic. In the reactive mode, our user model executes one of the possible and still relevant actions. The demonic choice now represents the human propensity not to respond in a fixed way to the device prompts, as long as the suggested actions are cognitively plausible. In the planning mode, the model simply executes the action selected in the decision step.

In our example, for the task of withdrawing cash, choosing the balance option is an irrelevant action and can be eliminated from further consideration by the angelic choice. In this case, the user model would no longer be able to choose the wrong action, even when behaving reactively. Consequently, model checking (4) would not yield the above false positive, correctly reflecting our commonsensical expectations. On the other hand, for the task of withdrawing cash and checking the balance, both actions for choosing a menu option are initially relevant. Thus the reactive user model would execute them in an arbitrary (demonically chosen) order. In this case, model checking would be able to identify the peculiar feature of the design in Fig. 1 we discussed earlier.

Next we develop a formal model based on these intuitive ideas.

3. Formal user model

In this section, we specify an abstract, generic cognitive architecture that models rational user behaviour as discussed in Sect. 2. By modelling the planned behaviour, we extend our earlier work [19] that dealt only with reactive behaviour. That work was based on a higher-order framework, SAL [14]. This allowed us to develop a proper generic architecture, which could be instantiated to derive a specific user model for a concrete interactive system. The SAL tools, however, resolve all instances of non-determinism in the same way, either demonically or angelically, depending on the property verified. Therefore, to be able to combine both angelic and demonic non-determinism, we use the Mocha framework here. Since Mocha does not support higher-order specifications, we describe our cognitive architecture as a generic template that can be later expanded/modified to derive concrete user models.

Our cognitive architecture is parametrised by the task actions users might take and the task goal. The number of task actions is defined as the Mocha constant `$NACTS`. In the description below, we will refer to each task action by its number `j:(1..$NACTS)`.

The state of the user model is specified by the variable `fin` of the type `Finished` = {`run`, `exit`, `abort`}. The value `run` indicates that the model is still performing a task, while the other two values indicate that task execution has been terminated, either upon achieving the task goal (value `exit`) or abnormally (value `abort`). The latter arises when there are no cognitively plausible actions to execute. The mode of behaviour is specified by the variable `beh` of the type `Behaviour` = {`planned`, `react`}.

The cognitive architecture consists of three Mocha modules: *mode*, *decision* and *action*. These modules are specified and explained below.

Mode of behaviour. The formal specification of the mode component is given in Fig. 3. `GOAL` is a placeholder that denotes the task goal; other placeholders are indicated by `/.../`. All the placeholders are to be expanded as necessary when concrete interactive systems are specified. This is the way we deal with developing a generic model without a higher-order formalism.

The specification states that the user model stops planning (i.e., behaves reactively) once the task goal has been achieved. In such a case, its behaviour may also include the termination of an interaction (see below). Until then the mode of behaviour is chosen non-deterministically (the Mocha keyword `nondet`) in a way determined by the concrete system under verification (see below). This is a coarse specification of the mode of behaviour. It can, however, be refined, taking into account other cognitive factors and environment impact (e.g., interruptions).

The non-deterministic choice of mode provides a means of modelling different classes of users. For example, we may assume that expert users are sufficiently trained and knowledgeable to plan their actions for achieving the task goal. They will tend to prefer the planning (angelic) mode. Such behaviour is captured by resolving the non-determinism in the `mode` module angelically. On the other hand, the demonic resolution of this non-determinism better captures the behaviour of novices who will tend to rely on the reactive (demonic) mode of behaviour. By refining the `mode` module, these user classes can be further fine-tuned.

```
module mode
 external fin : Finished; /.../
 interface beh : Behaviour
atom behaviour
 controls beh
 reads beh, fin, /.../
update
 [] fin = run & GOAL -> beh' := react
 [] fin = run & ~(GOAL) -> beh' := nondet
endatom
endmodule
```

Figure 3. Mode module.

Decision step. The decision step is formally specified in Fig. 4. The module `decision` consists of two components: an indexed collection of atoms, `relevance_$j`, each of which models the decision about the relevance of the action `j`, and the atom `planning`. As explained in Sect. 2.3, we assume that the non-deterministic choices will be resolved angelically in the `decision` module.

The variable `rel` is an array of booleans such that `rel[$j]` specifies whether the j-th action is still relevant to the task under execution. We assume that all task actions are relevant initially, thus `rel` is initialised to true. Then, each time the decision is made, any subset of the relevant task actions may non-deterministically be judged as no longer relevant. Note that the reverse decision is impossible: once the user model judges an action as irrelevant, it remains so during the task execution.

The atom `planning` models the selection of the next action to execute, denoted by the variable `plan`. Setting this variable to `0` indicates that no action has been planned (planning failed), which means that the user model executes the next action reactively (see below). The atom consists of an (indexed) collection of guarded commands. Each command simply chooses an action by setting `plan` to the corresponding index `k`. As explained earlier, this non-deterministic choice will be modelled as angelic.

To counteract the omnipotency of the angelic choice, only "predictable" actions may be chosen in this abstraction of planning. In the specification, the set of predictable actions is denoted by the placeholder `PREDICTABLE`. In concrete systems, this placeholder is to be substituted with an

```
module decision
 external fin : Finished; beh : Behaviour
 interface rel : Relevant; plan : (0..$NACTS)

#foreach j = (1..$NACTS)
atom relevance_$j
  controls rel[$j]
  reads fin, rel[$j]
init
[] true -> rel'[$j] := true
update
[] fin = run & rel[$j] -> rel'[$j] := nondet
endatom
#endforeach

atom planning
 controls plan, rel[0]
 reads fin, plan
 awaits beh
init
[] true -> plan' := 0
update
#foreach k = PREDICTABLE
[] fin = run & beh' = planned -> plan' := $k
#endforeach
[] true -> plan' := 0
endatom
endmodule
```

Figure 4. Decision module.

appropriate subset of (1..$NACTS).

Predictable actions are mental user actions and those physical actions whose outcomes depend solely on the user (e.g., taking the released cash, reading the balance information). This contrasts with the actions whose outcomes depend on the operation of the environment or device (e.g., choosing a menu option). However, we do not claim that the above list is exhaustive. Further research may well identify other instances of predictable actions. We further discuss this when considering concrete examples in Sect. 4 and 5.

Action step. The action step models the execution of an action. If there is an action planned in the decision step, then that action is executed. Otherwise, one of the enabled actions is non-deterministicaly chosen and executed.

This step is formally specified in Fig. 5. There the first guarded command in the `update` section is actually a generic template for specifying concrete task actions. These are derived by substituting (a) an action number for `j`, and (b) the actual guard and body of the corresponding action for the placeholders `GUARD_j` and `BODY_j`, respectively. In

```
module action
 external rel : Relevant; plan : (0..$NACTS);
          beh : Behaviour; /.../
 interface fin : Finished; /.../
atom actions
 controls fin, /.../
 reads fin, /.../
 awaits beh, rel, plan
init
[] true -> fin' := run; /.../
update
[] (beh' = react | plan' = 0 | plan' = j) &
   fin = run & rel'[j] & GUARD_j -> BODY_j
[] fin = run & beh' = react & GOAL ->
      fin' := exit
[] ~(fin = run) -> fin' := fin
[] default ->
      fin' := if WAIT then fin else abort fi;
      /.../
endatom
endmodule
```

Figure 5. Action module.

this template, its generic part

```
  (beh' = react | plan' = 0 | plan' = j) &
  fin = run & rel'[j]
```

states that the task action `j` can only be executed when the user model is still performing the task, the action is judged relevant and at least one of the following holds: (a) the model operates in the reactive mode, or (b) planning has failed, or (c) `j` has been planned in the decision step.

The second guarded command in Fig. 5 models the voluntary task completion once the task goal (placeholder `GOAL`) has been achieved. In such a case, the user model may terminate an interaction by setting `fin` to `exit`. The third command simply states that, once an interaction has been terminated, it remains so. Finally, the last command models users either waiting or forced to terminate an interaction when there are no enabled and still relevant task actions to execute. In such a situation, the user model waits for further prompts by doing nothing, if there is a cognitively plausible reason (e.g., a "please wait" message) to do so, or abnormally terminates by setting `fin` to `abort`, otherwise. In the specification, the wait condition is denoted by the placeholder `WAIT`.

Angel-demon games. There are three agents in the cognitive architecture we just developed: `mode`, `decision` and `action`. The decision agent has been specified assuming that non-determinism is resolved angelically, whereas the action agent has been specified assuming that non-determinism is resolved demonically. On the other hand, we have not made fixed assumptions about the nature of non-determinism in the mode agent.

A safer position is to have weaker presumptions about the capabilities of users and thus associate the mode agent with the demon. In this case, the game takes place between the decision agent on one (our) side and the mode and action agents on the other side. The ATL formulas to be verified are then of the following form:

```
  << decision >> property
```

In some scenarios, however, one might also want to consider users that are capable of planning and carrying out their plans. In this case, the game takes place between the mode and decision agents on our side and the action agent on the other side. Correspondingly, the ATL formulas to be verified are of the following form:

```
  << mode, decision >> property
```

In the subsequent sections, we give concrete examples that illustrate the game aspects of our cognitive architecture by providing concrete instances of the above two formulas.

4. Reactive example: ATM revisited

Now we return to the cash-point example introduced in Sect. 2. We show that the analysis based on our new cognitive architecture avoids the problems encountered previously. We start with the same ATM specification as before (Fig. 1). However, the concrete user model is now derived as an instantiation of our generic cognitive architecture by replacing its placeholders and action template appropriately.

Cash-point user. Our user model includes five concrete actions. The first four are those from the `update` section in Fig. 1. They are used as replacements for the placeholders in the action template from Fig. 5. For example, the guard and the body of the first action

```
menu & ~show & ~release -> choice' := balance
```

are substituted for `GUARD_1` and `BODY_1`, respectively. The fifth action is a new one:

```
  wait -> choice' := none
```

It models users waiting upon observing a "please wait" message. The variable `wait` is substituted for the placeholder `WAIT` as a wait condition, thus indicating that the user model does not terminate an interaction when a wait message is displayed in case of a delay in ATM response.

We also have to specify predictable actions. There are no mental actions in this user model. Out of the five physical actions, we consider as predictable the two actions that immediately achieve their goal (reading the balance information and taking the released cash). Also, the wait action can be viewed as predictable, since, by itself, it achieves nothing. Thus, the set {`3, 4, 5`}, which represents these actions, is substituted for the placeholder `PREDICTABLE`.

The condition of users achieving the task goal (placeholder `GOAL`) depends on the task performed. It will therefore be specified separately for each correctness property analysed. The above instantiation (including `GOAL`) defines the following specification, `ATMuser`, of ATM users:

```
  ATMuser := mode || decision || action
```

Finally, the whole interactive system is defined as follows:

```
  System1 := ATMuser || ATM1
```

Verification. Here we consider an average ATM user and do not assume that planning is involved in task execution. Therefore, only the decision agent is assumed to be angelic.

We start with the task of checking the balance and withdrawing cash. The task goal (`GOAL`) is thus `read & take`. Recall that our previous verification in Sect. 2.2 missed a potentially undesirable feature of the design specified by `ATM1`. This time, however, model checking shows that the corresponding property

```
          << decision >> F(read & take)          (5)
```

is false as one would expect for the reasons already explained in Sect. 2.2. Thus, our new user model allows us to identify problematic ATM designs that prevent users from checking the balance once the released cash has been taken.

Let us now modify the ATM design so that this feature is eliminated. This can be achieved by keeping the menu options available once cash has been released. The only change required to our ATM specification (`ATM1`) is adding to the withdrawal action a relevant assignment to `menu`:

```
  choice = withdraw ->
           show' := false; release' := true;
           wait' := false; menu' := true
```

This gives us new specifications, `ATM2` and `System2`, of the ATM and whole system, respectively. Now property (5) is satisfied for the new system. This indicates that the users interacting in a cognitively plausible way, as specified by the user model, will be able to successfully perform this task and achieve their goal with the interface design `ATM2`.

Similarly, the verification of property (4) shows that a simpler task of withdrawing cash (or checking the balance) can be successfully performed by the new user model. This contrasts with our verification in Sect. 2.2, which indicated an unrealistic problem (false positive) with the ATM design.

Summarising, the example demonstrates that analysis based on our new cognitive architecture yields less false

```
module BoatDevice1
  external m, b, push : bool
  interface home : bool
atom Move
  controls home
  reads push, b
  awaits b
init
  [] true -> home' := true
update
  [] push & b ->  home' := true
  [] default -> home' := ~b'
endatom
endmodule
```

Figure 6. Boat device specification.

positives. At the same time it allows one to detect potentially problematic design features like the one in ATM1.

5. Planning example: river crossing

In this section, we present an example that illustrates how our new cognitive architecture captures planned behaviour (problem solving) that is a part of task execution. The example is based on the classical puzzle of the wolf, the goat and the cabbage. The scenario is as follows.

A man arrives at a river. There is a boat at the bank, which can be used to cross the river. The boat, however, is big enough to contain only one of the man's belongings. Also, the wolf and the goat (and similarly the goat and the cabbage) cannot be left on the same side, since one of the pair will be eaten in each case. The man's goal is to travel to the other side with all the belongings safe. We extend this classical scenario by assuming that the river crossing is a part of a one-way road, thus the boat must be returned to its home station after the transfer is finished. For that, there is a device that moves the boat back. It is activated by pushing a button at the exit on the other side.

Next we formally model the above scenario in Mocha using our generic cognitive architecture.

Boat device. This device is specified in Fig. 6. A boolean variable, home, indicates whether the boat is at home (this side). The other three booleans, m, b and push, reflect the man's actions and are read by this specification. Thus, the variables m and b are true when, respectively, the man and the boat are on the other side, while push indicates whether the button to activate the device has been pushed.

Initially the boat is at home. When it is on the other side and the button is pushed, the boat is transferred home (first action). In all other cases (second action), the value of home depends on the man's actions as reflected by the variable b.

User model. Again the user model is an instantiation of our cognitive architecture. The man in our puzzle can cross the river with any of the three belongings. The corresponding actions to transfer, respectively, the wolf, the goat and the cabbage are as follows:

```
(m = w) & (b = m) ->
 m' := ~m; w' := ~w; b' := ~b; push' := false
(m = g) & (b = m) ->
 m' := ~m; g' := ~g; b' := ~b; push' := false
(m = c) & (b = m) ->
 m' := ~m; c' := ~c; b' := ~b; push' := false
```

As with m and b, the variables w, g and c are true when the wolf, the goat and the cabbage, respectively, are on the other side. Thus, the first action above states that the man can take the wolf to the other side only when all three, the man, the wolf and the boat, are on the same side. The other two actions are analogous. If he wishes, the man can also cross the river alone, specified as:

```
b = m -> m' := ~m; b' := ~b; push' := false
```

Finally, the man can press the button to activate the boat device at the exit on the other side:

```
m & b -> b' := false; push' := true
```

We use these five actions to instantiate our cognitive architecture as explained earlier. Also, we consider the first four actions as predictable, since their goal—crossing the river with one of the belongings or alone—is immediately achieved by performing them. On the other hand, pressing the button does not by itself achieve the goal of transferring the boat back. Thus, the set of the predictable actions, {1, 2, 3, 4}, is substituted for the placeholder PREDICTABLE.

There is no reason for the man to wait in this puzzle. Therefore, the wait condition false is substituted for the placeholder WAIT. Finally, the man's goal is to move to the other side with all the belongings. This is specified as the condition m & w & g & c, which is substituted for GOAL.

Note that we had to specify explicitly only concrete task actions such as crossing the river. Our cognitive architecture is such that the problem solving aspects are automatically dealt with during the model checking process.

Verification. Let us assume that this man is clever enough to sequence transfers so that no one is eaten during the process. In our approach, this assumption is simply represented by viewing the mode agent as angelic.

The correctness property we want to verify consists of two parts. One part, denoted done, states that the task has been performed properly. In other words, the man with all the belongings has moved to the other side and the boat is at home. This condition is formally expressed as follows:

```
m & w & g & c & home
```

The other part, denoted safe, states that there is no danger for anyone to be eaten. This can be ensured if the man is on the same side as the wolf and the goat (similarly for the goat–cabbage pair). This condition is specified as follows:

```
((w = g) => (m = w)) & ((g = c) => (m = g))
```

The correctness property can then be expressed as

```
      << mode, decision >> (safe U done)     (6)
```

Here U denotes the "until" operator. It states that the condition done is eventually achieved and, in every state before that, the condition safe is true.

For BoatDevice1, model checking shows that property (6) is false. The reason is that the man transfers all the belongings to the other side and then simply terminates the task without pressing the button to transfer the boat back. This is an example of the widespread post-completion error. It is a slip where a person terminates a task upon achieving its main goal, notwithstanding the fact that there might be subsidiary tasks necessary to perform for the proper completion of the main task. Our user model allows us to detect that the design of the boat device is not good enough even for a "clever" man to solve this puzzle properly, without committing a post-completion error.

Now let us enhance the design of our boat device. Suppose, there is a sensor installed at the exit that detects when a person leaves and then automatically activates the device. In our formal specification, this can be simply expressed by modifying the first action in Fig. 6 as follows:

```
(push | fin' = exit) & b -> home' := true
```

Here the condition fin' = exit models the sensor detecting a leaving person.

Will this improvement affect the outcome of verification? Indeed, model checking now shows that property (6) is true. Thus, with the improved design, the river crossing task is performed correctly by people capable of choosing an appropriate order to transfer their belongings.

The focus of our work is not dealing with problems potentially occurring in the planning process *per se*. Note, however, that many interactions involve some aspects of planning (problem solving). User interfaces might facilitate this process by, e.g., representing in some way the decisions made. In other words, interactive tasks frequently involve both planning behaviour and rather simple device specific actions that might be easily affected by cognitive slips. The example demonstrates that our approach can deal with planned behaviour in a simple and elegant way, thus facilitating the detection of errors due to cognitive slips in interactive systems.

6. Conclusion

Related work. Our work falls within the scope of approaches to reasoning about the behaviour of interactive systems known as *syndetic modelling* [9]. The latter combines, and considers as central components, a formal user model (e.g. [9, 5, 10]) and a formal specification of the device. Moher *et al* [13] also consider various types of users. Contrary to ours, however, their approach requires a separate specification for each user type. Rushby *et al* [21] focus on mode errors and the ability of pilots to track mode changes. They formalise plausible mental models of systems, though these are essentially abstracted system models, not based on a cognitive structure.

Back *et al* [4] use a game based approach to reason about interactive systems. They, however, assume that users are able to choose the right alternatives in any circumstances, i.e., their behaviour is purely angelic. Consequently, their approach does not represent real people whose behaviour, due to certain aspects of human cognition, can be seen as a mix of angelic and demonic aspects.

Game based ideas are applied [16, 17] to the analysis and construction of social procedures that involve multiple agents such as voting and allocation of shared resources. ATL and the Mocha environment is also used by van der Hoek *et al* [11] to model social laws in multi-agent systems.

Summary. People normally do not act maliciously and against their goals when performing interactive tasks. However, because of the limitations of human cognition, they do not always perform necessary actions either. In this paper, we presented a novel treatment of this dual nature of rational user behaviour in interactive systems. Relying on game based ideas, we developed an abstract cognitive architecture as a collection of competing/co-operating agents. The angelic planning agent represents abstractly the human ability to choose and order their actions according the situation and their goals. The demonic reactive agent represents the unpredictability of human choices, as long as they seem cognitively plausible. Finally, the mode agent represents the choice between planned and reactive behaviour. Whether this choice is angelic or demonic depends on a concrete system and the type of its users one is modelling.

Our cognitive architecture describes fallible behaviour. However, the model does not imply that mistakes will always be made. Erroneous behaviour only emerges if the model is placed in an environment that allows it. The architecture is generic in the sense that concrete user models are derived as its instantiations by providing suitable arguments. In this paper, we focused on modelling planning and reactive behaviour. Other important issues such as cognitive salience of actions had been dealt with in our earlier work [18]. In a similar way, they can be incorporated into the cognitive architecture presented here.

To illustrate our approach, we presented two simple examples[1] that involve reactive and planned behaviour. The examples showed that our game based view of the cognitive architecture allows us to avoid problems that are inherent when all instances of non-determinism in the user model are resolved in the same, either angelic or demonic, way.

One of the advantages of our approach is its abstract handling of planned behaviour. There is no explicit representation of the planning process in our model. This activity is encoded using angelic non-determinism and is carried out as part of the model checking process. As a result, the formal models become simpler and easier to comprehend, while modelling and analysis can focus on cognitive slips that occur during both reactive and planned behaviour in interactive systems. At the same time, the simplicity of our models makes their application more viable in practice.

Acknowledgments. We are grateful to Ralph-Johan Back for the stimulating discussion on this topic. This research has been funded by EPSRC grants EP/F02309X/1, GR/S67494/01 and GR/S67500/01.

References

[1] R. Alur, T.A. Henzinger, and O. Kupferman, "Alternating-time temporal logic", *ACM 49(5)*, 2002, pp. 672–713.

[2] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing", *IEEE Trans. Dependable and Secure Computing 1(1)*, 2004, pp. 11–33.

[3] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, S. Tasiran, "Mocha: Modularity in model checking", *CAV'98, LNCS 1427*, Springer-Verlag, 1998, pp. 521–525.

[4] R.-J. Back, A. Mikhajlova, and J. von Wright, "Reasoning about interactive systems", *Proc. FM'99, LNCS 1709*, Springer-Verlag, 1999, pp. 1460–1476.

[5] H. Bowman and G. Faconti, "Analysing cognitive behaviour using LOTOS and Mexitl", *Formal Aspects of Computing 11*, 1999, pp. 132–159.

[6] R. Butterworth and A. Blandford, "The principle of rationality and models of highly interactive systems", *Proc. INTERACT'99*, IOS Press, 1999, pp. 417–424.

[1] MOCHA specifications of these are available from `http://www.dcs.qmul.ac.uk/research/imc/hum/examples/sefm08.zip`.

[7] P. Curzon and A.E. Blandford, "Detecting multiple classes of user errors", R. Little and L. Nigay (eds), *Proc. EHCI'01, LNCS 2254*, Springer-Verlag, 2001, pp. 57–71.

[8] P. Curzon, R. Rukšėnas, and A.E. Blandford, "An approach to formal verification of human-computer interaction", *Formal Aspects of Computing 19(4)*, 2007, pp. 513–550.

[9] D.J. Duke, P.J. Barnard, D.A. Duce, and J. May, "Syndetic modelling", *HCI 13(4)*, 1998, pp. 337–394.

[10] D.J. Duke and D.A. Duce, "The formalization of a cognitive architecture and its application to reasoning about human computer interaction", *Formal Aspects of Computing 11*, 1999, pp. 665–689.

[11] W. van der Hoek, M. Roberts, and M. Woolridge, "Social laws in alternating time: effectiveness, feasibility, and synthesis", *Synthese 156*, 2007, pp. 1–19.

[12] D. Kieras and D.E. Meyer, "An overview of the EPIC architecture for cognition and performance with application to human-computer interaction", *Human-Computer Interaction 12(4)*, 1997, pp. 391–438.

[13] T. Moher, V. Dirda, R. Bastide, and P. Palanque, "Monolingual, articulated modelling of users, devices and interfaces", *Proc. DSVIS'96*, Springer-Verlag, 1996.

[14] L. de Moura, S. Owre, H. Ruess, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, "SAL 2", *Proc. CAV'04, LNCS 3114*, Springer-Verlag, 2004, pp. 496–500.

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

[16] R. Parikh, "Social software", *Synthese 132*, 2002, 187–211.

[17] M. Pauly, *Logic for Social Software*, PhD thesis, University of Amsterdam, ILLC 2001-10, 2001.

[18] R. Rukšėnas, J. Back, P. Curzon, and A. Blandford, "Verification-guided modelling of salience and cognitive load", submitted to *Formal Aspects of Computing*.

[19] R. Rukšėnas, P. Curzon, J. Back, and A. Blandford, "Formal modelling of cognitive interpretation", *Proc. DSVIS'07, LNCS 4323*, Springer-Verlag, 2007, pp. 123–136.

[20] R. Rukšėnas, P. Curzon, and A. Blandford, "Modelling and analysing cognitive causes of security breaches", *Innovations in Systems and Soft. Engineering 4(2)*, 2008, 143–160.

[21] J. Rushby, "Analyzing cockpit interfaces using formal methods", *Elec. Notes in Theoretical Computer Science 43*, 2001.

[22] L.A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication*, Cambridge University Press, 1987.

[23] M. Wooldridge and N.R. Jennings, "Agent theories, architectures, and languages: a survey", *ECAI-94 Workshop on Agent Theories, Architectures, and Languages, LNCS 890*, Springer-Verlag, 1995, pp. 1–39.