Integrating Formal Predictions of Interactive System Behaviour with User Evaluation

Rimvydas Rukšėnas[1], Paul Curzon[1], and Michael D. Harrison[1,2]

[1] School of Electronic Engineering and Computer Science, Queen Mary University of London
{r.ruksenas,paul.curzon}@eecs.qmul.ac.uk
[2] School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
michael.harrison@newcastle.ac.uk

Abstract. It is well known that human error in the use of interactive devices can have severe safety or business consequences. It is important therefore that aspects of the design that compromise the usability of a device can be predicted before deployment. A range of techniques have been developed for identifying potential usability problems including laboratory based experiments with prototypes and paper based evaluation techniques. This paper proposes a framework that integrates experimental techniques with formal models of the device, along with assumptions about how the device will be used. Abstract models of prototype designs and use assumptions are analysed using model checking techniques. As a result of the analysis hypotheses are formulated about how a design will fail in terms of its usability. These hypotheses are then used in an experimental environment with potential users to test the predictions. Formal methods are therefore integrated with laboratory based user evaluation to give increased confidence in the results of the usability evaluation process. The approach is illustrated by exploring the design of an IV infusion pump designed for use in a hospital context.

1 Introduction

Experiments in usability laboratories play an important role in providing understanding of the way people behave when using interactive devices in specific circumstances. These experiments can be used to identify flaws in the design of devices before deploying them in a wider context. Experiments, by their nature, are not exhaustive with respect to behaviour and so some behaviour that would occur under the assumptions of use made, may not be observed within the confines of the experiment. It is also difficult to predict all possible interactions that may lead to a particular observed behaviour and so ought to be dealt with in the interaction design. This is particularly difficult when investigating human error. For example, Vicente et al [1] have estimated the rate of errors related to number entry tasks, such as those arising from programming medical infusion devices, to be in the range of 1 in 33,000 to 1 in 338,800 for an example device. In experiments error rates have to be increased artificially by, for example, adding secondary tasks to increase working memory load to overcome such problems. This can lead to those experiments not being ecologically valid: the errors seen may not actually correspond to real situations.

This paper describes a framework that can be used to highlight error prone interaction design given a set of cognitive assumptions. When evaluating an interactive system design it is necessary not only to describe assumptions about the device design but also the assumptions that are being made about the user in terms of capabilities and context. An approach to exploring the consequences of such assumptions is proposed that combines formal verification techniques with laboratory-based experiment. We claim that this approach gives increased analytical power to experimental results. The behavioural assumptions and configuration of an experiment including the device are modelled at a high level of abstraction. This allows model checking to be used to explore the consequences of the behavioural assumptions exhaustively in a way that is not possible using experimental or simulation techniques. Formal methods are therefore integrated with laboratory based user evaluation to give increased confidence in the results of the usability evaluation process.

The approach contrasts but also has some parallels with the use of cognitive modelling to analyse user assumptions [2], in particular the use of cognitive architectures. A base set of assumptions is used, similar to cognitive architectures, that are relatively independent of the task. However cognitive architectures provide much more detailed models of cognitive processes such as visual or auditory perception, memory and learning, whereas the approach described here takes a more abstract view of these processes as discussed in more detail in Section 3.2.

Another important difference relates to the way the models are used to carry out analysis. In cognitive modelling approaches, the analysis of system properties is based on individual simulation runs of a relatively small number of possible behavioural traces. The idea is that each trace represents statistically ‘average’ or ‘likely’ behaviour. The result is that models which are effectively deterministic are used to predict likely properties of an interactive system. In the approach described here the models are more abstract and involve a high degree of non-determinism. This non-determinism generates a wide range of behaviours. It allows *exhaustive* exploration of the consequences of the modelled assumptions that lead to these behaviours using automatic tools such as model checkers.

In addition to testing predictions based on the models the analysis gives insight about the design of the interactive device and helps the evaluator to consider the validity of the experimental design. It therefore provides insight into the confidence with which the results of the experiment and its interpretation can be considered. It can highlight mismatches between the consequences of the assumptions and experimental results and so lead to suggestions of further experiments as well as ruling out potential explanations for those mismatches.

To illustrate the proposed approach we explore a design that supports the access of information within two tasks. The tasks can either be interleaved or carried out sequentially. Our aim is to consider the effectiveness of the design given a set of cognitive assumptions, in particular the soft constraints hypothesis [3]. Modelling predicted that errors would be made that matched those observed in the experiment. However the specific traces identified by the model checker that led to errors were different to the sequence of actions followed by participants in the experiment. In particular the formal analysis predicted a different form of interleaving from that assumed or seen in the experiment. This suggests that the modelled assumptions are not sufficient to fully explain the observed behaviour. Possible explanations for why the actual behaviour differs from that derived from the assumptions could be explored both by further experiment and/or by modelling them and model checking to derive the consequences.

The contribution of this paper is therefore to:

- present a novel way of combining formal reasoning technology and laboratory-based experiments to explore the consequences of the design of an interactive environment, and
- demonstrate on a specific example how the technique can be used to give insight into experimental results and in particular make explicit the behavioural assumptions made and its design consequences.

2 Related Work

The use of formal modelling and analysis as a means of developing hypotheses for experimental evaluation of interactive systems appears to be a novel approach. There is however a significant literature on combinations of user or use assumptions with models of devices. These range from assumptions based on task models to assumptions based on cognitive models. A recent example of the former approach is the work of Bolton and others [4] which also contains a good review of related material. They use the Enhanced Operator Function Model (EOFM) to describe operator tasks. This task model is combined with a device model as a basis for a model checking analysis using SAL [5]. The analysis involves considering variants of the task by inserting automatically "phenotypes of erroneous human behaviour. These variant models are checked against correctness properties - that the combined model would reach specified goals. Observable manifestations of erroneous behaviour are also explicitly modelled by Fields [6] who also analysed error patterns using model checking. Both approaches, however, whilst giving specific kinds of errors to explore in the form of the mistake model lack discrimination between random and systematic errors. The also assume implicitly that there is a correct plan, from which deviations are errors. Beckert and Beuster [7] on the other hand take a step towards combining GOMS modelling and correctness verification. They present a verification environment with a structure similar to the models described here — connecting a device specification, a user assumption module and a user action module, the latter being based on CMN-GOMS. The selection rules of their GOMS model are driven by the assumption model while the actions drive the device model. This gives a way of exploring the effect of erroneous user behaviour in the form of incorrect selection decisions as specified in the user assumption module. However, the assumption module has no specific structure and, thus, does not provide systematic guidance as to what kind of potential errors to explore. These decisions are left to the analysts of the system.

Other relevant research concerns the use of more general assumptions about cognition. A similar approach to the one that forms the basis for this paper is taken by Bowman and Faconti [8]. They formalise one model of human information processing (Interactive Cognitive Subsystems [9]) using the process calculus LOTOS, and then apply a temporal interval logic to analyse constraints on the information flow and transformation between the different cognitive subsystems. Their approach is more detailed than the one described in this paper. It focuses on reasoning about multi-modal interfaces and analyses whether interfaces based on several simultaneous modes of interaction are compatible with the capabilities of human cognition. One source of of user error is cognitive mismatch between user beliefs about the system state or behaviour and the reality. Rushby et al [10] focus on mode errors resulting from cognitive mismatch and the ability of pilots to track mode changes. They formalise a mental model of the system that is specific to the example being considered and then analyse it using the Murϕ verification tool. Their models make no explicit appeal to cognitive principles.

In our earlier work [12], a similar integration of formal verification and laboratory based experiments is employed to provide cognitively grounded accounts of interactive user behaviour. That work, however, does not attempt to evaluate device designs.

3 The Modelling Framework

The proposed analysis of an interactive system is based on a combination of a device model and a user model that together capture assumptions about the design relevant to the context of use. The level of abstraction used for the device specification is determined by the issues under investigation. In the example described here the issue is whether certain task steps are prone to omission given specific design assumptions, not about precise details as to whether a particular task step is carried out. For this reason the device specification is given a high level of abstraction. The specification of plausible user behaviours then follows the same abstraction level. The models are developed using the SAL verification environment [5].

3.1 The Device Model

Our example involves the programming of infusion pumps. Infusion pumps are used both in hospital settings and at home to provide intravenous infusions. They are safety-critical devices, since infusing a drug at the wrong rate or volume may seriously harm patients. As such they provide a realistic test of the viability of the approach.

Analysis is concerned with the task of programming a pump with the prescribed infusion parameters and then commencing the infusion. However because simultaneous programming of two infusion pumps is a common activity in operating theatres, the focus is to consider this multitasking activity and designs where the setting of a pump requires entry of two infusion parameters: the volume to be infused (VTBI) and the duration (time) of infusion[1]. Different makes of infusion pump provide different mechanisms for entering these numeric values. The concern here is not with the details of number entry. This level of abstraction of the pump model captures the generic characteristics of a range of models of infusion pump. The general insights provided by the analysis are therefore likely to be associated with the design characteristic of these different pumps.

Pump operation. The first step in developing the device model is to describe the interactive aspects of pump operation. When the pump is switched on it goes through a setup procedure. Since this step is not a concern of the example, the model simply assumes that the initial state of the pump is on when the setup has finished. The programming

[1] The model would be similar if the rate of infusion is required instead of duration.

options available to the user are presented at a main menu on the pump display on completion of setup. Setting the VTBI and setting the duration are two options which, when selected, move the pump into a mode where the relevant numerical value can be entered. The entered value must be confirmed by pressing the confirmation key. If confirmation occurs when both values have been entered the pump calculates the infusion rate automatically. Pressing the confirmation key returns the pump to the mode where the main menu is displayed. Infusion can then be started using the appropriate key. However, before that, a roller clamp on the pump must be opened.

Pump model. The SAL model of the interactive behaviour for this pump is given in Fig. 1. The variables `vtbi`, `time` and `rate` represent the values of the the infusion parameters. The analysis is to be carried out at a level of abstraction where these numeric values are irrelevant. For this reason the three variables have boolean type. The value `true` indicates that a numeric value for the corresponding infusion parameter has been entered, whereas `false` indicates the opposite. Depending on its mode, the pump shows (some of) these infusion values on the display. The boolean variables `vtbiDisp`, `timeDisp` and `rateDisp` indicate whether the corresponding value is displayed or not. Finally, the boolean `clamp` specifies whether the roller clamp is closed (`true`) or open (`false`).

The mode of the pump operation is specified by the variable `mode`. Three modes of operation are assumed defined as an enumerated type, `Mode`:

```
Mode: type = { off, hold, infusing };
```

The modes `off` and `infusing` indicate that the pump is switched off and infusing, respectively. The `hold` mode represents the remaining states of the pump, when it is switched on but not infusing (for example, being programmed).

The mode of the pump display is specified by the variable `dmode`. The mode can take values defined as the following type `DispMode`:

```
DispMode: type = { dblank, mainmenu, dvtbi, dtime, dinfusing };
```

The value `mainmenu` represents the main pump menu with the programming options presented. The values `dvtbi` and `dtime` represent the numeric entry displays for VTBI and time respectively. The value `dinfusing` represents the display shown during the infusion process. Finally, `dblank` indicates that the display is blank.

The device model of the infusion pump is driven by a set of actions that are defined by input events represented by the following enumerated type `Event`:

```
{ onoff, mvtbi, mtime, enter, confirm, open, close, infuse, tick }
```

At the level of abstraction relevant to the analysis, an input event may correspond to a sequence of button presses. Thus the event `onoff` represents a key press that switches the pump on or off. The events `mvtbi` and `mtime` model users choosing the VTBI and time entry options in the main menu. The events `enter` and `confirm` represent the entry and confirmation of a numeric value (depending on the display mode, this can be either the VTBI or time). Opening and closing of the roller clamp is modelled as the events `open` and `close`. Finally, the time ticking event `tick` represents cases when there is no user action taken.

```
pump: module =
begin
 input event: Event
 local mode: Mode
 output dmode: DispMode, vtbi, rate, time, vtbiDisp, rateDisp, timeDisp, clamp: bool
 initialization
  mode = hold; dmode = dvtbi; vtbi = false; rate = false; time = false; clamp = true;
 definition
  vtbiDisp = dmode = mainmenu or dmode = dvtbi or dmode = dinfusing;
  rateDisp = dmode = mainmenu or dmode = drate or dmode = dinfusing;
  timeDisp = dmode = mainmenu or dmode = dtime or dmode = dinfusing;
 transition
 [  event = onoff and mode = hold --> mode' = off; dmode' = dblank;
 [] event = mvtbi and dmode = mainmenu --> dmode' = dvtbi
 [] event = enter and dmode = dvtbi --> vtbi' = true
 [] event = confirm and dmode = dvtbi --> dmode' = mainmenu; rate' = rate or time;
 [] event = mtime and dmode = mainmenu --> dmode' = dtime
 [] event = enter and dmode = dtime --> time' = true
 [] event = confirm and dmode = dtime --> dmode' = mainmenu; rate' = rate or vtbi;
 [] event = open and clamp --> clamp' = false
 [] event = close and not(clamp) --> clamp' = true
 [] event = infuse and dmode = mainmenu and vtbi and rate and time -->
                  mode' = infusing; dmode' = dinfusing
 [] event = infuse and dmode = dinfusing --> mode' = hold; dmode' = mainmenu
 [] else -->
 ]
end
```

Fig. 1. Pump model in SAL

These events have behaviours (see `transition` section in Fig. 1) that depend on the mode of the device (`mode`). For example, the event `enter` sets `vtbi` to `true` if the display mode is `dvtbi`. This models the entry of the VTBI value when the pump display is in the corresponding mode. In general, events may have the effect of changing two modes: the mode of the device and the mode of the display. They may also change the variables associated with the infusion parameters (`vtbi`, `time` and `rate`).

The example considers the simultaneous programming of two pumps. The model for each pump is derived from `pump` by simple renaming of all variables. For example, `event` is renamed to `events[1]` for the first pump and to `events[2]` for the second. The full device model, `Pumps`, is then defined by composing the pump models.

3.2 The User Model

The purpose of the user model is, when combined with the device model, to restrict the device behaviours to those that are consistent with user behaviour given the cognitive assumptions. The particular user model that is relevant to the analysis of the device is based on an instantiation of an abstract generic user model. These models are generic because they provide the means to replace sets of cognitive assumptions and also because they can be instantiated with the particular task assumptions relevant to the analysis. The model makes it possible for the experimenter to make, and explore, conjectures about use of the interactive system. In this way the approach is not locked into a set of assumptions about how the device will be used.

Flexibility of user models is achieved through three modelling layers. The base layer in the generic model captures core assumptions that are unlikely to be modified by the approach. The intermediate layer, also part of the generic model, specifies the current set of cognitive assumptions. The third layer is an instantiation of the generic model and captures specific assumptions that relate to the details of the device and the task to be performed on it.

The Base Layer. This layer focuses on the mechanisms that relate to the users choice of actions and forms a set of core assumptions. It postulates that actions are chosen non-deterministically but some actions may be preferred to others for cognitive reasons such as their salience. User preferences are modelled using a notion of action *salience* that is informed by the ideas of activation theory [11]. The base layer describes actions and their salience in generic terms. It also specifies termination behaviour, marking the conditions that may lead to a person ending an interaction. More detail on this modelling layer is provided in the earlier paper [12].

The Intermediate Layer. The second (intermediate) layer refines the underlying non-determinism of action choice by introducing salience levels which it uses to partition actions. The notion of salience is also refined by specifying how action salience is derived from associated cues. At this level different assumptions about the salience levels and the relation between salience and cues can be specified. One possible set of such assumptions is described in more detail below and used in the analysis of the example.

The set of assumptions used in the example specify that actions may have two types of cues: sensory (that is external) and internal. The sensory cues are provided by the device and its environment. In the model, they are used to represent any kind (visual, audio, etc.) of external stimulus. The internal cues originate from the user's knowledge of task and device.

Task-knowledge cues can be thought of as a mental representation of the task, and what is necessary to achieve the main task goal. This knowledge is assumed to derive from general training as well as previous experience. The form that task-knowledge cues take in the model is as follows: 'action A $\leadsto$ action B'. This corresponds to learned behaviour where each action in a sequence provides longer term activation for the next action. Namely, if 'action A $\leadsto$ action B' and another action C (or series of actions S) are taken instead of B, then the latter still gets activation after execution of C (or S). The second form of task-knowledge cue in the model deals with the first step in a learned series of actions. In this case, there is no preceding action to provide activation and it is assumed that activation may also come from task goals.

The set of *device-knowledge* cues can be thought of as a mental representation of how the device works, and what is necessary to achieve the task using the device. It is assumed that this knowledge derives from repeatedly doing the task on the same device. Thus, the device-knowledge cues as a whole capture learnt sequences of actions. These sequences may also include device-specific actions. This is not the case with the task-knowledge cues. The device-knowledge cues take the following form in the model: 'action A $\rightarrow$ action B'. They represent more procedural aspects of learned behaviour and, consequently, are assumed to provide shorter term activation for the next action in a sequence. Namely, if ‘action A $\rightarrow$ action B’ and another action C is taken immediately after A, then action B ceases to get activation from A.

Action salience and activation levels. The overall salience of an action is determined by the activation provided by its sensory, task-knowledge and device-knowledge cues. The effect of different kinds of cue is assumed to be *equal* and *additive* in nature. Here the equality means that each kind of cue, if present, is assigned a unit (say 1) of activation, while the additivity means that the overall salience of an action is calculated as the sum of these units. Thus, if an action gets activation from all three kinds of cues, then its overall salience will be 3, whereas if it does not have any cues, then the overall salience will be 0. In this set of assumptions, four discrete levels of activation are assumed, each corresponding to one of the possible values (from 0 to 3) of overall salience, so that all the actions are partitioned into these levels. Only actions with the highest level of salience are assumed to be candidates for execution.

Not all user actions that are possible at some point are equally relevant to achieving task goals. For example, the action of starting an infusion is irrelevant when the prescribed infusion parameters have not yet been provided. In the model, the concept of *specificity* refers to the dynamic aspect of cue relevance for such actions. It is assumed that an action being non-specific acts as an inhibitor reducing the activation due to the sensory and task-knowledge cues from 1 to 0. On the other hand, the activation due to the device-knowledge cues is not linked to the specificity of actions in this set of assumptions.

A set of cognitive assumptions like this is chosen on the basis that it is believed to be sufficient to explain behaviour for the given task. If discrepancies with experiments arise then one possibility is that this understanding is incomplete, which in itself is a useful result.

The Concrete Layer. The third (concrete) layer instantiates the generic user model specified by the other two layers to a specific interactive system and its associated tasks. It does this by defining the state space of the user model, the main task goal and the actions and their associated cues specific to the device. In this case the task is programming two infusion pumps.

State space. The state of the user model is specified by the following components: `inp:Inp` giving things the user can perceive in the world, `mem:Memory` giving their beliefs about the state of the system, and `out:Out` giving the actions they can take. The type `Inp` represents assumptions about what the pump users can perceive:

```
Inp: type =
 [# dmode:array [1..2] of DispMode, vtbi:array [1..2] of bool,
   time:..., rate:..., clamp:..., prescription:... #];
```

Here `dmode[i]` ...`clamp[i]` indicate how the user perceives the corresponding attributes on the pump `i`, while `prescription[i]` indicates the perception of the prescription values for the same pump. The type `Memory` represents assumptions about the user’s beliefs about the system state:

```
Memory: type =
   [# pump:[1..2], vtbiSet:array [1..2] of bool,
      timeSet:..., prescription:..., interleave:bool #];
```

Here pump indicates which pump is the focus of user attention, vtbiSet[i] and timeSet[i] represent beliefs as to whether the corresponding infusion parameter has been set, prescription[i] represents the memorised prescription values, and interleave indicates whether the user has chosen to interleave programming the two pumps or not. Finally, the type Out specifies assumptions about which user action (action), and on which pump (pump), has been chosen by the model:

```
Out: type = [# action: Event, pump: [1..2] #];
```

Task goal. We assume that, from the users point of view, the main goal, task for the task of programming the two pumps is to reach a state such that their perception indicates that both pumps are infusing:

```
inp.dmode[1] = dinfusing and inp.dmode[2] = dinfusing;
```

User actions. Programming a pump involves entering the prescribed VTBI first, then confirming it. After that the time option must be selected from the available menu which, as in the case of VTBI, allows entry of the prescribed infusion time (duration) followed by confirmation. The required VTBI and time values can be read from the prescription form. In the experiment the form could have been positioned either nearby or further away from the pump so that the user had two plausible options: to read and memorise both values (VTBI and time) for one infusion, or to consult the prescription form at the time when each of these values had to be entered. When both values have been entered the user is required to open the roller clamp and start the infusion process.

The task description prompts the specification of a set of user actions (as opposed to device actions) in the concrete model layer. The type ActionNames defines the names of these user actions:

```
ActionNames: type =
 { memorise, enterVtbi, confirmVtbi, chooseTime,
   enterTime, confirmTime, openClamp, startInfusion };
```

Some of these actions such as enterVtbi or chooseTime represent groups of key presses. However, these details are deemed to be irrelevant for the analysis of interleaving behaviour and so abstracted away without loss of generality.

These user actions are associated with the action cues as specified in Table 1 (the actual SAL specification is given by defining the relevant parameters for the generic user model). Each cell in this table indicates an action (given by its name) and/or a state condition (written in italic) that is necessary to activate the corresponding cue (given by the column title) for the action given by the row title. For example, the action enterTime is cued by the action enterVtbi on the task-knowledge level and by the action chooseTime on the device-knowledge level. It also has sensory cueing, whereas its specificity (relevance) is defined by the conjunction of the following boolean conditions: "m.pump = *this one* OR m.interleave" (user is involved in programming this

Table 1. Specification of action cues

Action	Task cues	Device cues	Sensory cues	Specific, if
`memorise`	task goal to start infusion	NONE	NONE	`costs = true` AND `NOT(m.prescription[pump])`
`enterVtbi`	`memorise` OR `costs = false`	`memorise`	YES	(`m.pump` = *this one* OR `m.interleave`) AND `NOT(inp.vtbi[pump])` AND `inp.dmode[pump] = dvtbi`
`confirmVtbi`	NONE	`enterVtbi`	YES	(`m.pump` = *this one* OR `m.interleave`) AND `inp.vtbi[pump]` AND `inp.dmode[pump] = dvtbi`
`chooseTime`	NONE	`confirmVtbi`	YES	(`m.pump` = *this one* OR `m.interleave`) AND (`NOT(inp.time[pump])` OR `NOT(m.timeSet[pump])`)
`enterTime`	`enterVtbi`	`chooseTime`	YES	(`m.pump` = *this one* OR `m.interleave`) AND `NOT(inp.time[pump])` AND `inp.dmode[pump] = dtime`
`confirmTime`	NONE	`enterTime`	YES	(`m.pump` = *this one* OR `m.interleave`) AND `inp.time[pump]` AND `inp.dmode[pump] = dtime`
`openClamp`	`enterTime`	`confirmTime`	NONE	(`m.pump` = *this one* OR `m.interleave`) AND (`inp.vtbi[pump]` OR `m.vtbiSet[pump]`) AND (`inp.time[pump]` OR `m.timeSet[pump]`)
`startInfusion`	(`openClamp` AND `m.pump` = *this one*) OR `startInfusion` *on the other pump*	`openClamp`	YES	`inp.dmode[pump] /= dinfusing` AND *for both pumps:* (`inp.vtbi[pump]` OR `m.vtbiSet[pump]`) AND (`inp.time[pump]` OR `m.timeSet[pump]`)

particular pump or has chosen to interleave programming), "`NOT(inp.time[pump])`" (the user perceives that the time value currently displayed is different from the prescription value), and "`inp.dmode[pump] = dtime`" (the user perceives that the pump is in the time entry mode). Table 1 specifies action cues for programming one pump. In the two pump scenario considered, the specifications for both pumps simply duplicate that given in the table. This model layer also has a boolean parameter, `costs`. It is true, when the costs of accessing information (prescription form) are assumed to be high, and false otherwise.

These assumptions focus on the distinction between the task-orientated and device-orientated steps [13]. The device-orientated steps are potentially more problematic because they have lower activation levels than their task-orientated counterparts. These lower activation levels are assumed to be the result of the different ways in which device- and task-orientated steps are represented in a mental model. It is assumed that device-orientated steps are associated only with the device-knowledge cues, while task-orientated steps are associated with both task- and device-knowledge cues. Table 1 shows that the actions `confirmVtbi` , `chooseTime` and `confirmTime` are assumed to be device-orientated; they do not have task-knowledge cues. All other actions are cued on the task-knowledge level. The action `memorise`, if taken, is the first in a series. Therefore, it is assumed to be cued by the task goal.

As can be seen in Table 1, it is also assumed that all actions that involve the buttons on the front panel of an infusion pump are sensorily cued. On the other hand, the roller clamp (positioned at the side of the pump) provides no sensory cues for the action `openClamp`. It is also assumed that there is no sensory cueing for the action `memorise`.

3.3 The System Model

The specification of the interactive system as a whole involves combining and connecting the device model and the user model. This requires two additional models: firstly that of user interpretation of the device interfaces (`Interpretation`) and the environment, and secondly a model giving the effect of user actions on the pumps (`Effect`). These additional models connect the state spaces of the device and user models. These connectors are in fact simple. The `Interpretation` model renames appropriate variables as in the following case:

```
inp.dmode = dmodes
```

For the values of the infusion parameters (e.g., VTBI) such renaming takes into account whether the relevant value is displayed by the pump:

```
inp.vtbi[pump] = (vtbis[pump] and vtbisDisp[pump])
```

Finally, the perception of a prescription form is assumed to depend on the costs of consulting it. If these costs are considered to be low, a prescription form can always be perceived (consulted):

```
inp.prescription[pump] = (costs = FALSE)
```

The effect of user actions is specified in `Effect` by stating that the input event on `pump` (`events[pump]`) is either a “do nothing step (`tick`) or whatever action (`out.action`) the user model produced.

The SAL module `System` of the interactive system is then specified as the following composition of all these separate models:

```
(User || Effect) [] (Pumps || Interpretation)
```

The structure of this composition also applies to other interactive systems involving different devices.

4 Verification-Based Analysis

Given the model as specified in the previous section, the aim is to explore potential usability problems that might arise through the use of this interactive device under the cognitive assumptions made. The impact of the costs of accessing information are of particular interest. The aim is to generate predictions about use that can be compared with the results of an experimental study.

SAL model checking tools were used to analyse the properties of the interactive system model. The cognitive assumptions about user behaviour (a 'surrogate' user) help to identify unforeseen interaction issues by asking general questions: for example, does the user model always achieve the task goal? In the example, such a question is formulated as the following LTL property `goal` (`F` means 'eventually'):

```
F (task)
```

The property states that, in any interactive system behaviour, the task goal is eventually achieved (i. e., user perception indicates that both pumps are infusing). However, starting infusion before a roller clamp is opened is *unsafe*. Thus the following LTL property, `safe`, is formulated to check if that holds (`G` means 'always):

```
G ((dmodes[1] = dinfusing => not(clamps[1])) and
   (dmodes[2] = dinfusing => not(clamps[2])))
```

This property checks, for each pump, whether its roller clamp is open whenever the pump is infusing.

The analysis starts with the assumption that the costs of consulting a prescription form are low (`costs` was set to `false` in `System`). Indeed, before programming a pump, it makes sense for a nurse to position a prescription form nearer to the device so as to minimize the cost of looking back and forth between each. In this case, model checking the property `safe` produces a trace that describes a roller clamp error on the first pump. It is not obvious how to change the design of the infusion pump itself to prevent this error of forgetting to open the roller clamp. However, this error can be explained by the interleaving behaviour in programming two pumps, while such behaviour may be encouraged by easy access to a prescription form. Therefore, it is plausible to hypothesise that increasing the costs of accessing prescription may help to avoid the omission error on the roller clamp step.

The next step in the analysis is to verify this hypothesis by setting `costs` to `true` in the model. In this case, model checking both properties `safe` and `goal` succeeds. The successful verification can be interpreted as a prediction that any design change in the interactive environment, that increases the costs of accessing the prescription values, helps to prevent the omission error. In practice, such a change can be achieved in several ways: for example, by positioning a prescription form further away from the pumps, or by chunking prescription values (VTBI and time) for each pump on the form. The impact of such changes on the safety of pump programming can be further explored in experimental studies.

5 The Experiment

As already discussed one of the aims of the experiment was to investigate how the design of an interactive environment impacts on safety when programming two infusion pumps simultaneously. The experiment demonstrates that the seemingly sensible action to position a prescription form nearer to the device increases the likelihood of the error of forgetting to open the roller clamp when programming two pumps.

Method. Back et al [14] describe an experiment that investigates how the physical layout of the environment impacts on participants' interleaving behaviour when programming two infusion pumps. Participants were invited to program the pumps using information from a prescription form. The physical and mental effort involved in accessing information was manipulated by varying the physical distance between the prescription form and the devices.

The soft constraints hypothesis [3] maintains that when selecting between low-level memorisable procedures, those that tend to minimise performance cost while achieving expected benefits will be selected. Performance cost can be measured in terms of time for example. Depending on the situation, a perceptual strategy (where the prescription form is consulted when the relevant value is needed) may be more efficient than a memory-intensive one of memorising both prescription values at once. In the low information access cost condition, the soft constraints hypothesis suggests that people are more likely to use a perceptual strategy, when retrieving information needed to program a device, over a memory-intensive one.

Results. Participants were only able to use a low-level strategy when the prescription form was located alongside the devices being programmed. Critically when adopting a perceptual strategy, value entry may be driven by prompts from the devices, rather than what values are held in memory. A user may continue to enter values, consulting the prescription form and interleaving between devices as necessary, until all requested values are entered. Experimental data showed that adopting a perceptual strategy encouraged interleaving during device programming, which resulted in an increased omission error rate. Such errors were rare when people chose not to interleave until they finished programming one device.

Generally, these results corresponds to the predictions based on the verification of the interactive system model. However, the specific example of erroneous behaviour generated by the model suggests a different interleaving behaviour than the ones observed in the experiment. This discrepancy could be a behaviour that could plausibly happen in reality but was just not seen in the (limited) experiment. It may alternatively suggest that the cognitive assumptions believed to apply (so modelled in the intermediate layer) in this situation are actually insufficient. In particular it may be that there is something more that matters than is captured by the specified distinction between task- and device-oriented interaction steps. Alternatively, it could be that the set of concrete modelling assumptions about action salience and cueing (Section 3.2) is not sufficiently precise. In either case, the modelling has drawn attention to something that needs further investigation if the experimental results and so the usability of the design are to be fully understood. On the other hand, systematic experimentation can be used to validate the generic user model [12].

6 Conclusion

This paper has described a novel approach to evaluating the usability of an interactive device design using a formal method that focuses on the experimentation associated with user evaluation. The technique helps the experimenter to interpret results formatively to improve a potential design. It also makes more general predictions (e.g., about the impact of the costs of accessing information) as opposed to specific conditions and scenarios that are investigated in experiments. In the example an abstract description of a design of the interactive system was produced that not only described the device but also provided information about other resources, such as prescription forms, that could be used in the interaction. The results of the evaluation indicate potential changes to the larger system — the context in which the interactive device is to be used.

It is typically difficult both to interpret the results of usability evaluation, and to make appropriate changes to the design of an interactive system as a result of the evaluation. The approach presented in this paper uses formal methods in a novel way, integrating it with laboratory studies to improve an iterative design process in relation to the development of interactive systems.

There are several issues that require further study. Firstly, if these techniques are to be used effectively in bridging the gap between a formal specification of the interactive system design and its empirical evaluation then the assumptions that are captured by the interaction model must be developed in a format that is comprehensible and disputable by the evaluator who will come from a human factors tradition.

Secondly, the behaviours that are being investigated using this approach are intended to be error behaviours. These are behaviours that may be business or safety critical. Typically (and hopefully) these behaviours are rare. Experiment cannot always provide access to such errors and therefore other techniques intended to increase their likelihood must be chosen, for example by using secondary tasks. Aspects of experiments such as these are not explored using the modelling approach described in the paper.

These two issues are the basis for further study.

Acknowledgments. This project was partly funded by the CHI+MED project: Multidisciplinary Computer Human Interaction Research for the design and safe use of interactive medical devices (UK EPSRC Grant EP/G059063/1). We are grateful to Jonathan Back and Anna Cox of University College London Interaction Centre for input relating to the experiment and cognitive assumptions.

References

1. Vicente, K., Kada-Bekhaled, K., Hillel, G., Cassano, A., Orser, B.: Programming errors contribute to death from patient-controlled analgesia: case report and estimate of probability. Canadian Journal of Anesthesia / Journal canadien d'anesthésie 50, 328–332 (2003)
2. Ritter, F.E., Young, R.M.: Embodied models as simulated users: introduction to this special issue on using cognitive models to improve interface design. International Journal of Human-Computer Studies 55, 1–14 (2001)
3. Gray, W.D., Sims, C.R., Fu, W.T., Schoelles, M.J.: The soft constraints hypothesis: A rational analysis approach to resource allocation for interactive behavior. Psychological Review 113(3), 461–482 (2006)
4. Bolton, M.L., Bass, E.J., Siminiceanu, R.I.: Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking. International Journal of Human-Computer Studies 70(11), 888–906 (2012)

5. de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 496–500. Springer, Heidelberg (2004)
6. Fields, R.E.: Analysis of erroneous actions in the design of critical systems. Technical Report YCST 20001/09, University of York, Department of Computer Science, D.Phil Thesis (2001)
7. Beckert, B., Beuster, G.: A method for formalizing, analyzing, and verifying secure user interfaces. In: Liu, Z., Kleinberg, R.D. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 55–73. Springer, Heidelberg (2006)
8. Bowman, H., Faconti, G.: Analysing cognitive behaviour using LOTOS and Mexitl. Formal Aspects of Computing 11, 132–159 (1999)
9. Barnard, P.J., May, J.: Interactions with advanced graphical interfaces and the deployment of latent human knowledge. In: Interactive Systems: Design, Specification, and Verification (DSV-IS 1995), pp. 15–49. Springer (1995)
10. Rushby, J.: Analyzing cockpit interfaces using formal methods. Electronic Notes in Theoretical Computer Science 43 (2001)
11. Altmann, E.M., Trafton, J.: Memory for goals: an activation-based model. Cognitive Science 26(1), 39–83 (2002)
12. Rukšėnas, R., Back, J., Curzon, P., Blandford, A.: Verification-guided modelling of salience and cognitive load. Formal Aspects of Computing 21, 541–569 (2009)
13. Ament, M.: The role of goal relevance in the occurrence of systematic slip errors in routine procedural tasks. Technical report, UCL, PhD thesis (2011)
14. Back, J., Cox, A., Brumby, D.: Choosing to interleave: human error and information access cost. In: Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, CHI 2012, pp. 1651–1654. ACM, New York (2012)