Including User Behavior in Model Checking Analysis
José Creissac Campos, Gavin Doherty, and Michael D. Harrison

Abstract
Human reliability is an important element in the resilience of a software based system, yet formally based analyses of software that use techniques such as model checking do not in general take account of human interaction with the system. While the vast majority of accidents in safety-critical systems are attributed to “human error”, there has been little consideration within software engineering of how system designs might have an active role in leading users to those errors. The potential of formal techniques to analyze how systems behave in response to the many possible user behaviors has not been fully exploited. This is partly due to unrealistic assumptions made by existing techniques about how systems will be used. This paper describes a technique that analyzes interactive systems using model checking. The technique investigates properties of cognitively plausible interactions within the system. The notion of cognitive plausibility captures those behaviors that a user is likely to exhibit when faced with a given design. It is expressed in terms of assumptions about how the interface will influence actions taken by the users. The analysis considers whether the interface provides sufficient support to allow the user to achieve specific goals, and identifies scenarios where insufficient support is given. Addressing these concerns will typically involve changing the design of the system. Two examples are used to illustrate the approach. The analytic support for this technique, provided by two different notations and toolsets, is investigated through these examples.

I. INTRODUCTION
While the correctness of software is undoubtedly an important focus for formal techniques for mission-critical and safety-critical systems, demonstrating correctness is not enough to ensure the resilience or dependability of the software. Other factors, for example errors or failures that arise through human computer interaction, are also important [17]. Extending the techniques that support correctness to include human interaction would allow a greater proportion of design problems to be detected and rectified early in the development process. Model checking is now a widely used verification technique [1]. Use of model checking is being facilitated by the development of generic models describing classes of system (see, for example, [12] on publish-subscribe systems), and approaches designed to ease the problem of property formulation. The latter includes the use of general property templates (see, for example, [11]), and visual languages and tools to help express verification requirements (see, for example, [26], [2]).

The use of behavioral models, focusing on the system and supported by automated reasoning, to analyze human-computer interaction has been the subject of previous research [22], [6], [25], [18] but how best to fold the user into the analysis of system resilience remains an open problem. The paper addresses this issue by proposing a technique that narrows the model checking analysis of interactive systems to cognitively plausible paths.

Assumptions about how the system is to be used are established in the technique by making information needs explicit. This process includes establishing whether the right information is provided to the user at the right time in support of activities. The designer must consider a range of issues: support for a range of user strategies, making the most of available screen space, avoiding information overload, and reconciling competing information requirements when the system supports a number of different activities. A typical interaction design issue that relates to information resources is the “keyhole problem” in which users are distracted from achieving their primary goals by, for example, accessing different screens within a hierarchical menu structure in order to get the required information [27]. These information needs provide constraints on user behavior, and together with additional resources afforded by the environment as a whole in which the system is situated, act to shape the likely behaviors of the end user. Encoding these information needs, and using them to drive analysis, enables consideration of a broader class of uses and user behaviors than could be achieved by restricting analysis to the behaviors encoded in more prescriptive models of user behavior (for example, task models [15], [20]).

The approach described here builds on previous work on automated analysis of user interaction, and aims to show how a model of information use, in the form of resources, can be used to constrain the presumed activities of users of a device/system to a plausible and interesting subset. This model of information use is explored in combination with a model of the operated system, and potentially a model of the environment or controlled process. The aim of the paper is:

• to illustrate the use of resources as a modeling concept, demonstrating that it allows a richer analysis of the behavior of the interactive system, thus informing the design of resilient systems

J.C. Campos is with the Departamento de Informática, Universidade do Minho, Braga, Portugal.
G. Doherty is with Trinity College Dublin, Ireland.
M. D. Harrison is with Newcastle University, UK.
to formalize and to automate a portion of the analysis, thereby reducing analyst effort and increasing the likelihood of finding problems.

- to demonstrate the feasibility of applying the approach in distinct verification contexts using two different analysis tools. A preliminary version of this work has been published as [4], [10]. While the previous papers focused on the concepts surrounding constraint by information resources and were intended for an audience focussing on human interaction issues, this paper focuses on a method of analysis that uses resources. Two case studies are explored. The first illustration is a photocopying machine and demonstrates the method using both the IVY tool [8], and SPIN [13]. The second uses SPIN to analyse a simple ubiquitous computing environment. The second example has been discussed in general terms previously [19], [10] using models specified in Statecharts and UPPAAL. In Loer and Harrison’s paper [19] the primary focus was the tool (IFADIS) for supporting an analysis, while Doherty et al. [10] use an UPPAAL model to outline how a resource based approach would work without explicitly defining how the actions are actually to be resourced. SPIN makes it possible to analyze larger and more realistic models.

In the next section (Section II) the modeling approach is outlined. Resources are explained in terms of how they can be used to support the analysis of systems in use. The manner in which resources are specified and the way in which goals are used in property formulation are discussed, along with the possibilities for tool support. This section introduces the proposed method. Section III illustrates the method using both IVY and SPIN on the photocopying machine. It is shown how a device model can be used to determine the availability of resources. The second example is used as a further illustration in Section IV. Finally, discussion of the wider application of the method and of further developments is to be found in Section V.

II. REASONING ABOUT INTERACTIVE SYSTEM DESIGNS

Model checking generates traces when a property that is being checked fails to hold. The trace, a sequence of states, indicates a situation in which the property has failed. For example, starting from a situation where some error has occurred, the designer may wish to know whether it is possible that the user can always solve the problem while avoiding “unsafe” system states. Failing this property would produce a trace in which an unsafe state is reached. These sequences of states can be used by human computer interaction or domain experts to produce narratives that conform to a situation described by the sequence but include a richer sense of the context in which the situation might occur. Not all behaviors that satisfy or fail to satisfy a property are of interest. Those that are of interest are the ones that are likely to be carried out by users. Many of the behaviours that demonstrate the breaking of a property might never occur through user interaction.

These interesting behaviors will be described as cognitively plausible behaviors and will be intended for use as a basis for interdisciplinary analysis. Domain, software engineering, and human computer interaction experts can together discuss scenarios represented by the traces. Model checking a device without taking cognitive plausibility into consideration, however, can generate too many uninteresting behaviors, rendering this analysis impractical.

The most usual way to represent cognitive plausibility is by describing the possible use of the system as a task. This encodes a relatively small number of behaviors, and can be seen as the opposite extreme to considering all behaviors. While task-based analysis is useful, a more liberal view is adopted here of the constraints on paths that will permit a greater number of plausible paths than the task model. The advantage of this approach is that it recognizes that users often achieve goals in ways that were not envisaged by designers when defining their task assumptions while at the same time recognizing some constraints that will affect users. The proposal is that while some actions will be resourced effectively, others will not, leading to usability problems. The approach combines user actions with resources that support these actions to define appropriate paths.

A. Resources for action

In the proposed resource-based analysis, individual user actions are the basic units of analysis. The resourcing of each action is specified independently. The focus for analysis is whether each user action is appropriately resourced, or whether appropriate combinations of resourced actions lead to the achievement of user goals. For an action to be supported (depending on expertise with the system) in a particular context, certain information resources must be present in that context. For example, if a mobile phone (the device) has an action to save a draft text message, then it is necessary that (1) action availability is resourced (the “save” option is currently on the screen), (2) the action is enabled (the message memory is not full), (3) action-effect information is available (is the label “save to drafts” or just “save”?), and (4) required information about the current state is available (is it saved already?). Regardless of how a user edits a text message (is it in reply to another message or is it a group text?), or higher level user tasks and goals (which may be varied), the basic resourcing for this action remains much the same.

In the approach to be described the specification of the system is structured as a set of actions, which affect the state of the system, accompanied by an appropriate model of system state. Various forms of interactive system specification could provide a means to build this specification, and indeed interactors based on Modal Action Logic [6] focus on the actions supported in an interface with the aim of easing this specification.

The specification of the device or the system is enhanced with information resources. It is important to note that this approach is not just a vehicle for automated analysis of behavior, but also leads to a systematic consideration of the resourcing of situated user actions. In the process of thinking about and specifying the resources, assumptions that are being made about the design
will become explicit. Using model checking tools will make it possible to identify situations where actions are not adequately resourced.

Wright and others [28] note that, in typical interfaces, information either in the device or in the environment can take specific forms:

- **status/visible information** - a resource may simply consist of a piece of information, for example the display indicates that a message is waiting (a resource) in order for the user to perform an action to read the message. This is distinct from the system being in a state where reading a message is possible.
- **action possibility** - a resource may consist of information that an action is available. There are two issues here, one is the information that the possibility for carrying out the action exists (e.g. the resource lets the user know they can save an unsent message for resending later, a feature they were unaware of), the second is that the action is enabled (or not) in the current state - perhaps the message memory is full.
- **action effect information** - a resource may let the user know what the likely effect of an action will be. The same piece of information on action availability may also convey information on action effect; “press ok to save” conveys information both on action possibility and on action effect.
- **plan information** - some resources provide plan information, that is, they aid in the sequencing of user actions. For example, interfaces in which an overall task performance sequence is made explicit (“You are in step 3 of 5”) are providing a plan resource.
- **goal information** - some resources may correspond to user goals, helping the user to formulate and keep track of multiple goals. For example, “there are new messages” could act as a goal resource within the interaction. In complex, real-world situations, there may well be a hierarchy of different goals, and goals may possibly conflict, so denoting resources as goal resources is only a small part of the analysis of goals.

These resources may be found in the physical context, in the device itself or in the user’s head. The model makes these resources explicit. However it will be seen that these categorizations are not explicit in the specification language used to create the models that demonstrate the method. The reason for this is that a particular resource could play a number of roles in the use of the system. While specifying the resourcing for particular action it may be clear that a particular resource would not be available in the proposed design, and an immediate consideration would be given to the problem. However, many resourcing problems may be more subtle in their evolution, and will not be clear from inspection, particularly if the user has multiple goals, and interleaves actions which contribute to different goals. Other issues could relate to the impact of interruptions on the resourcing of particular actions.

**B. A method of analysis**

Specifying and examining the resourcing of individual actions can form a useful vehicle for goal based analysis, as it is possible to ask questions such as whether resourced actions are available which will support achievement of the user’s goal. The basic process proposed is as follows:

- **Step 1**: Develop a device model, in which the actions available to the user are specified. In the first illustration, the IVY framework and PROMELA are used.
- **Step 2**: Augment the device model by describing resources that are intended to be used in the context of the actions associated with the device model.
- **Step 3**: Define a relation between resources and actions describing the intended role of the resources in supporting actions.
- **Step 4**: Consider and specify potential user goals.
- **Step 5**: Formulate properties, including those surrounding user goals.
- **Step 6**: Run the properties over the model, and use them to explore the role that strategies play in supporting the actions that are to be carried out. Analysis of the results will again lead to redesign of the system and refinement of the specification.

Building the model is first concerned with specifying the actions that are to be resourced. In the case of IVY, the notation is oriented around action and therefore the process is relatively straightforward. In the case of the SPIN specification it is a little more complicated as will be discussed. Having specified the actions (Step 1), the resources that are required for the user to carry out these actions are then considered (Step 2). To do this effectively it is necessary to know whether information, which is potentially available through the system, is visible when the action is to be carried out. Thus some visibility model must be included. This can include what is seen in the environment as well as the device. There is a choice between specifying the exact information to be displayed and indicating only that the resource is available. Existing mechanisms for denoting visible state, such as those in interactor models can be used. In the second example (Section IV), additional state attributes are introduced to capture these resources as well as nominating existing state attributes that are considered to be visible in the proposed design. The system specification therefore defines: the resources which are available in a given state, and the actions which can be performed, which affect the set of available resources.
To support reasoning about resourced (i.e. cognitively plausible) behaviors, resourcing requirements will be used as constraints, defining which actions might happen at any given point (Step 3). Hence, this form of analysis will view usability problems in terms of insufficiently resourced actions, and suggest increased resources at key points in the interaction. The act of specifying these resource constraints in itself constitutes an initial analysis, and may well lead to redesigning the system and refining the specification.

Analysis is carried out with respect to user goals. The question that is important to ask (Step 5) is whether fully resourced sequences exist that allow the reaching of the goal state. To illustrate the idea, consider the states and transitions in figure 1.

Fig. 1. Resourced actions and properties

and an attempt to prove that some goal state \(s_g\) is always reachable. Consider a state in which a failure has occurred, \(s_1\), and \(s_g\) represents recovery to a safe system state. The dotted transitions \(t_{3g}\) and \(t_{5g}\) are not well resourced. A traditional analysis would indicate \(s_7\) and \(s_6\) as problematic, even though reaching \(s_6\) is not plausible. Taking resources into consideration, \(s_6\) is no longer considered (since reaching it is not plausible), but \(s_3\) becomes problematic (since reaching \(s_g\) from \(s_3\) is possible but not plausible). Next, consider a procedure (e.g. from a task model) to reach the goal state via \(s_1-s_2-s_3-s_g\). This trace is legal, but it is not plausible as insufficient resources are available in \(s_3\). Only one fully resourced trace exists, \(s_1-s_4-s_5-s_g\).

Modeling resource assumptions as restrictions on the system’s behavior makes it possible to determine whether specific (goal) states can always (or eventually) be reached in a cognitively plausible way. If the goal state cannot be reached, two possible situations can be envisaged. The existing resourcing leads users in the wrong direction for the goal, or the existing resources do not enable the user to carry out the actions needed to satisfy the goal. In any case, the model checkers will attempt to provide counter examples to illustrate the specific problem found. The behaviors illustrated by the counterexamples can be analyzed (Step 6) to understand how the resourcing can be changed to prevent users taking a wrong direction.

III. APPLYING THE METHOD TO A PHOTOCOPIER

A photocopier is chosen first to illustrate the method described in Section II-B. Two specification notations and associated tools are chosen. The first demonstrates how the technique can be used to augment the systematic analysis technique supported by the IVY tool [7], [8]. The second uses PROMELA and SPIN [13]. The two approaches will be compared before providing further analysis of a ubiquitous system in Section IV.

Clearing a jam in a photocopier is an activity that is infrequently carried out by users. It is a simple example of a general class of diagnosis and repair tasks, it is important in such tasks to understand the resilience of mechanisms for ensuring that the activities are carried out correctly. Clearing a jam is an activity that is often mishandled and therefore provides a good context for resource based analysis while at the same time being simple enough to illustrate the method. Because there are too many possible user behaviors that will not solve a given jam, resource constraints will be used to characterize those behaviors that are cognitively plausible given the current design. The analysis in this section, then, is concerned with whether there is enough information provided by the photocopying machine and its environment to allow a typical user to deal with jams of different kinds.

A. The device model

Following the method outlined in section II-B, for (Step 1) a specification of the device and its actions is required. This will be achieved with a MAL model. The MAL interactor notation is described elsewhere [6], [7]. For the purposes of brevity a deliberately simplified model of the photocopier is adopted.

The copier uses a tray to hold paper and displays relevant information to users. The tray can be taken out and the internals of the device can be accessed by opening a door. It is possible to use the door to load paper, change cartridges and clear
The photocopier also provides actions that enable the user to start and stop copying, but in addition the user can open and close the door to inspect the machine and fix problems at three different points (A, B and C) of the internal mechanism. This is particularly useful in the case of a copier jam. These facilities support a set of user actions: start, stop, open, close, checkA, checkB, checkC. The copier has functions to produce a single copy (makecopy) or to jam (jam).

Some of the description above is captured in the model excerpt in figure 2.

interactor photocopier
aggregates
  tray via tray1
  screen via display
attributes
  vis door: DoorT
  vis copying: boolean
  error: ErrorT
actions
  vis open close start stop checkA checkB checkC
  jam makecopy
axioms
  per(start) → ¬copying ∧ door=close ∧ error=ok
  [start] copying' ∧ keep(door, error)
  per(stop) → copying
  [stop] ¬copying' ∧ keep(door, error)

Fig. 2. Basic structure of the model

The interactor defining the behavior of the device is named (photocopier). It incorporates (by instantiation) two other interactors: tray1 (instance of tray) and display (instance of screen). It has state attributes: door and copying (that are visible to the user via the vis tag), and error which can only be perceived when the machine does not work (the display will be used to convey information about system state). DoorT and ErrorT are types describing the possible door states (open, closed), and possible errors. It declares the nine actions identified above. By focusing on actions, and how they affect perceivable state attributes, the model aims to capture the possible interactions with the device.

Of the nine possible actions, seven of which are user actions (as indicated by the vis tag), two actions (start and stop) are defined by axioms in figure 2. The guards on these actions (per(action) → guard), and the effects of each action ([action]effect) are specified separately in Modal Action Logic. The model excerpt describes the start/stop actions in terms of whether copying takes place and how they change or are changed by door and error states. The remaining features of the system are specified similarly.

The exact number of copies being requested is abstracted from this version of the model. The axioms capture only the knowledge that once the copier is started it will eventually stop. The complete model used in this section can be found at [5].

B. Exploring the device specification

The IVY workbench [7] enables model checking (via NuSMV [9]) of MAL models. The model defines four possible jam errors: ABC, AC, BC, and C. To check whether the ABC jam can be cleared the following CTL formula can be used:

\[ AG(error = abc \rightarrow EF(error = ok)) \]  

(1)

It states that whenever an ABC error occurs, there is a possible future state of the device where the error has been solved. The property checks true for this model. The jam problem, if it occurs, can be solved. However, no information is provided about the solution. To obtain a trace that solves the jam, a formula expressing that the jam cannot be solved is used:

\[ AG(error = abc \rightarrow !EF(error = ok)) \]  

(2)

This property fails and the trace produced shows the sequence of actions: checkA, checkB, checkC. This is indeed the expected strategy to solve the problem. Figure 3 shows the trace as presented by IVY. Each column represents an interactor in the model, and shows the sequence of actions that are executed.

This analysis demonstrates that it is possible to solve the jam condition and provides an example of how to do it. The model checker produces the shortest possible path that falsifies a property. Hence, the trace in figure 3 can be considered to be the shortest possible strategy to solve the jam.
C. Resourced User Behaviors

Figure 3 describes a trace that is not constrained by resources. There is no guarantee that the chosen trace is cognitively plausible in the terms described in previous sections. Although at first sight the generated trace gives an obvious behavior, it is not guaranteed that a user will be able to carry out the sequence of actions in the trace. In order to analyze that, help from domain or human factors experts would be needed. As commented above, this becomes tedious and error prone when a large number of non-plausible traces is produced. In order to avoid that, user actions are related to needed resources (Step 3). Note that available resources (Step 2) are already encoded in the model via \[ \text{vis} \] annotations.

There will not always be an exact match between the modeled information resources and the information made visible by interactors. Resources needed for an action may be provided by several interactors. For example, an engineer repairing a machine might use an onboard display of status (state) information, along with a handheld device displaying procedure information for the repair (plan information in a resource based view). Both the machine status panel and the handheld may be displaying additional information not relevant to the repair task. Conversely, when specifying the resources needed for an action, much of the information provided by an interactive system might not be relevant to this particular activity. Once defined, however, they help to eliminate unlikely behaviors from the analysis, and can be used for any number of verification tasks.

In the MAL model resources are associated with perceivable attributes, for example the content of the display. Resource constraints are encoded in the specification through permission axioms for the actions that are described in the device specification:

\[
\begin{align*}
\text{per(device.opend)} & \rightarrow \text{device.display.info} \in \{\text{errorABC, errorBC, errorAC, noPaper}\} \\
\text{per(device.start)} & \rightarrow \text{device.display.info} = \text{idle} \land \text{device.door} = \text{closed} \\
\text{per(device.close)} & \rightarrow \text{device.display.info} = \text{idle} \land \text{device.door} = \text{open} \\
\text{per(device.checkA)} & \rightarrow \text{device.display.info} \in \{\text{errorABC, errorAC}\} \\
\text{per(device.checkB)} & \rightarrow \text{device.display.info} = \text{errorBC} \\
\text{per(device.checkC)} & \rightarrow \text{device.display.info} = \text{errorC} \\
\text{per(device.tray1.opend}) & \rightarrow \text{device.display.info} = \text{noPaper}
\end{align*}
\]

An explanation of how these axioms capture the resource constraints is illustrated by considering the action “close”. The axiom
states that it is enough for users to know that the door is open to correctly perform the action (i.e., perform the action at the appropriate time), a strong resources definition would state that the user would need to be told to close the door only when the problem has been solved.

D. Analysis

After adding the resourcing information to the model, only cognitively plausible user behaviours (as defined by the resourcing) are considered by the model checker. Attempting the verification of property 1 fails with the trace in figure 4. It can be concluded that the system does not support users in solving the jam. The next step in the analysis is to determine why the verification fails.

Since a resourced strategy for solving the jam does not exist, the model checker was not able to produce a complete trace. Hence, the exact resourcing need that will make users find it difficult to solve the jam problem is not clear from figure 4. It can be identified by starting with the unresourced model and selectively adding the resourcing requirements. This enables identification of the particular constraint that causes the problem. By doing this, it can be concluded that the problem lies in the fact that once the door is open the information regarding the strategy needed for solving the jam problem is no longer available. This happens because the door open information replaces the jamming information. However, that information is needed for the user to decide on the checkA action. The conclusion then is that the system is not satisfying the resourcing requirements for action checkA. Note that these requirements are derived from user rather than system considerations.

If the display is enhanced to present two items of information so that the door open item does not hide the jam item, then it is possible to verify the model, and conclude that all resources that are needed to solve the jam are available. Alternatively it might be noticed that the door open display information is not being used in any of the resourcing assumptions. If this is the case for all of the resourcing needs of the system, then it will be necessary to consider whether that information needs to be on the display.

Analyzing resource constraints might show alternatively that users take paths that deviate from the expected. This has particular relevance for system safety; are there cognitively plausible paths that lead to undesirable situations? Consider, for example, that opening the paper tray is a salient enough option that it might be considered a plausible action after the door has been opened. Hence the constraint determining when the open tray action (openT) can be loosened (previously the assumption was that it was resourced if the “no paper” message was displayed). In that case, the verification above would succeed. Determining a possible trace would conclude that users would open and close the tray after opening the door. The effect of this would be to update the information on the screen to show the error message. However, an attempt to show that this user strategy would always work, would fail. This happens because, as modeled, whenever a message is discarded the screen is updated non-deterministically with one of the possible messages that might be displayed. In some cases the needed piece of information might get displayed when the tray is closed, in others it might not.

E. Comparing and contrasting the PROMELA model

In PROMELA [13], the concept of action is implicit in the specification and therefore the production of a model that clearly supports the notion of resource is more difficult. The PROMELA model for the photocopier uses a channel called button to communicate the different button presses and actions. No distinction is made in this case between different types of action though this can be done relatively simply by distinguishing channels for opening and closing trays, opening and closing doors.
button!start;

do
  :: display[status]==idle -> break
  :: display[status]==jammed-> button!opendoor
  :: (display[status]==dooropen) && (display[errortype]== abc) -> button!checkB
  :: (display[status]==dooropen) && (display[errortype]==ac) -> button!checkA
  :: (display[status]==dooropen) && (display[errortype]==bc) -> button!checkB
  :: (display[status]==dooropen) && (display[errortype]==c) -> button!checkC
  :: display[status]==needspaper -> button!removetray
  :: display[status]==trayopen -> button!replacetray
od

Fig. 5. The PROMELA jam dealer process

and checking A, B and C. The additional channels do not materially change understanding of the model and so have not been included.

Resources are encoded as state attributes in the model. Actions in the model update the values of these attributes. Hence the photocopier model updates display[status] which shows the status of the machine. The indicated status is one of: idle, copying, jammed, needspaper, trayopen, dooropen. A further resource display[errortype] displays the error that caused the jam: ok, abc, bc, ac, c. Notice that, following from the discussion on the MAL model, an enhanced version of the display is being modeled. More specifically, a display capable of presenting both status and error type information is being represented.

A contrasting feature of the PROMELA model, compared with the MAL model, is the need to model the environment and user behavior explicitly. In MAL, under-specified features of the system are assumed to exhibit random behavior by default. Modeling resource assumptions in MAL (in fact modeling any behavior) amounts to restricting the range of possible behaviors. The PROMELA model is more operational, with resource assumptions defining which user actions become possible for each resourcing configuration. In PROMELA, non-determinism is expressed explicitly by allowing the possibility of multiple options being available at the same time. Rules must then be written to define how actions affect the state attributes. An environment process is defined to inject jams randomly into the system. The user is defined to set up the device for 5, 10 and 15 copies randomly. Details of the model are to be found at [5]. The interesting part of the model from a resourcing point of view is the mechanism for dealing with the various possibilities that occur once photocopying has been started. The resource constraints are described in the jam dealer process (see figure 5). Hence if the displays indicate that the door is open and the error type is abc then the checkB action is resourced.

With this resource arrangement a similar analysis to the MAL analysis can be performed. Using the LTL property:

$$[]((error==abc) \rightarrow !<>(error==ok))$$

is equivalent to demonstrating that an ABC jam cannot be solved. The verification fails, and the path that is constructed by the SPIN model checker provides a counterexample indicating a situation where error is ok is achieved. This is similar to the one produced in the MAL case, but considerably harder to read. As an illustrative example, consider the fragment of the trace in figure 6. This part of the trace indicates that the display[0] resource (status display) indicates the door is open, and that the display[1] (error display) shows abc. It then indicates that the next step taken by the user is to check B. The correct action in this case would be check A. The full trace identifies the strategy induced by the resourcing information to solve the jam problem.

This section has explored the applicability of the approach using two different modelling languages and verification tools showing that resources can easily be integrated into the models and verification processes.

67: proc 3 (user) line 26 "pan_in" (state 7)  
  [(((display[0]==dooropen) && (display[1]==abc)))]
69: proc 1 (photocopier) line 46 "pan_in" (state 3)  
  [display[0] = dooropen]
71: proc 3 (user) line 26 "pan_in" (state -) [values: 2!checkB]
71: proc 3 (user) line 26 "pan_in" (state 8) [button!checkB]
72: proc 1 (photocopier) line 103 "pan_in" (state -) [values: 2?checkB]
72: proc 1 (photocopier) line 103 "pan_in" (state 86) [button?checkB]

Fig. 6. Trace fragment from SPIN
IV. RESOURCING UBIQUITOUS SYSTEMS

The method is now applied to a design scenario involving a mobile device used by a single operator to control a chemical process (based on the device presented by [21]). The example described here can be seen as an example of a general class of process control activities, where human error can be a major concern, embedded within a ubiquitous system. As well as dealing with a more complex scenario, this example illustrates the modeling of a controlled process as well as the device, and introduces a location model representing the position of both the user and the device.

A. Background to the example

The system controls a physical process comprising tanks, pipes and pumps that move material between tanks for processing. The control of the process is achieved electronically, see Figure 7, by operating pumps to move material between tanks. Two tanks are designed to process raw materials. Tank 2 processes material A to produce C and tank 3 processes material B to produce D. A holding tank (tank 1) contains raw and manufactured materials until they are ready for processing. These raw materials are pumped in from two independent sources by means of pump 1 and pump 2. The product in tank 1 is loaded and taken away by a tanker using pump 5. Three different types of pumps are used, each with its own controls. Pumps 1 and 2 allow the operator to specify the volume that is to be reached before the pumping process cuts out. Pumps 3 and 4 are directional and continue until the source tank is empty. Hence in forward mode pumps 3 and 4 move material to the tanks for processing and in backward mode they are moved from the processing tanks back to tank 1. When in forward mode the directional pump initiates the automatic processing of the material in the destination tanks (2 and 3). If for any reason the pump is stopped when in forward mode (either because the source is empty, the destination is full or the off button has been pressed) then processing of the material in the destination tank is initiated.

In the envisaged design the operator combines visual inspection of the plant with active control of the pumps and valves to manufacture the product. The operator moves around the plant carrying a mobile device (see Figure 8) and can use it to control the process. In the existing system the control room gives the operators a good overview of the operation of the plant. The new design offers relatively limited information to the operator and so analysis is required to assess whether the design provides appropriate information to the operator for safe and effective use.

To produce C a material (A) must be pumped into tank 1 using pump 1 (and the tanks involved must be empty for this process to be carried out successfully). Once tank 1 is full pump 3 is put into forward mode to move the material from tank 1 to tank 2, thereby filling tank 2. The pump then pauses while tank 2 processes the material, changing it from A to C. The flow of pump 3 is then reversed and tank 1, which had previously been emptied, is filled with the product. The final stage involves using pump 5 to remove the product from tank 1 to a tanker for transportation. The second goal is achieved in a similar manner. Tank 1 is also used in this process but this time it is fed from pump 2 and the manufacturing process takes place in tank 3 producing product D.

![Fig. 7. The process involving three tanks and five valves](image-url)

The method outlined in Section II-B is used to analyze cognitively plausible traces in relation to the operator’s use of the proposed device. The model (see [5]) provides a limited but adequate specification of the chemical process that is being controlled. For example, it is sufficient for the purposes of understanding the human interfaces that tanks are defined as variables and pumps are described as processes. Two features of the tanks are specified because they are key to the operator’s understanding of the chemical process:

- the type of material contained within them
• the volume of each tank, treated as a variable in the range 0 (empty) to 5 (full).

When manufacture of the product from raw material takes place a time delay is assumed. Timing is not a focus in the present analysis and therefore simplifying assumptions are made about time that do not affect the way interactive behavior is resourced.

B. Modelling actions in the device: step 1

The device is designed to control the pumps. Which pump is controlled depends on the mode of the device and the pump information that has been collected from the relevant position within the plant room. Part of the model captures the position of the operator and the device. The mobile device (Figure 8) (full specification see [5]) supports the various actions associated with controlling the pumps. Two processes are specified to achieve this: a device process and a space process. The space process captures physical movement of the operator in carrying the device around the space. It defines the paths that the operator can take between the pumps illustrated in Figure 9. While user action is the focus in this specification, a number of system actions that are controlled by the mobile device will also be relevant to the analysis.

Fig. 8. The mobile device

Two pumps can be accessed by the device wherever the operator and device are located in the plant: device[display] (identifies the pump that can be controlled using the touch screen) and device[store] (identifies the pump that can be controlled by the touch screen after “switching” it from store). In the device depicted in Figure 8, device[display]=1 and device[store]=5. The volume pump (pump 1) is controlled using the touch screen and a simple pump (pump 5) can be switched with the displayed pump. A number of actions are supported by the device depending on which pump is displayed. These are invoked in the specification by sending an appropriate message (using an mtype value) to the buttons channel. buttons?switch for example specifies that a switch action is invoked. Which actions are permitted depends on the mode specified by the pump currently located in dev[display]. This value is used to select the channel that communicates with the relevant pump. Hence pump[cd] where cd is assigned to dev[display] is used to describe the pump action; pump[2]!on therefore turns pump 2 on. The switch button (called the bucket selector in Figure 8) is always available and switches the displayed controls with the stored controls. In Figure 8, Pump 5’s controls will be switched with Pump 1’s controls. The collect button finds which pump is in the locality and downloads it (overwriting the displayed pump). By this means it is possible to use any two pumps from anywhere in the space.

Fig. 9. The paths between pumps in the control room

All of the pumps can be switched on and off and automatically switch off when the destination has reached its limit or the source is empty. The following actions are available depending on the pump.

volume pump: up, down (to change the destination tank limit)
directional pump: backward, forward (to change the direction of flow)
simple pump: only allows turning on and off
The additional controls offered when the volume and directional pumps are displayed can only be used when the pump is switched off. Hence, although an operator can move around the plant, it is only possible to switch or collect pump information if the previous action is completed.

C. What resources are required? (Step 2)

A number of information resources are relevant to the operator in carrying out an action. Step 2 makes resource assumptions. This step should ideally be carried out with a domain or human factors expert. Through such a consultation it is determined that to produce C the operator is likely to need to know at different stages in the process:

1) whether tanks 1 and 2 are empty
2) whether pump 1 is available to transfer A to tank 1
3) whether pump 3 is in forward mode and whether it transfers product A to tank 2
4) when to reverse pump 3 — this should take place when the material has been processed
5) when to use pump 5 to extract the material in tank 1 to a tanker.

Hence a number of resources are specified in the model:

1) resource[pumptype] indicating the type of the pump currently available to the operator. The types are: simple, volume or directional. This resource provides an action possibility (see Section II-A) for the operator.
2) resource[pmpstate] indicating to the operator whether the currently available pump is on or off. This offers information about whether the pump can be turned on or off in the next action. It also provides status/visible information indicating what the displayed pump is currently doing.
3) resource[pmpdirection] indicating what the pump direction is if it is directional. This resource offers action possibilities for the backward and forward actions. This resource is only available if the current pump is directional.
4) resourcevol indicating the limit at which the volume pump will turn itself off. This state information is not visible in the current version of the device.
5) resourcescevol indicates how much material is in the source tank
6) resourcedstvol indicates how much material is in the destination tank.
7) position indicates where the operator and device are currently located
8) schedule indicates the order of pumps that have to be used in the process.

One of the goals of the analysis is to support exploration of the availability of these resources, and in some cases (e.g. 5, 6), whether and when they are required by the operator. In several cases the designed device may provide no help to the user in offering these resources. However other features of the whole system may make some of the proposed resources available.

D. Defining the relation between resources and actions (Step 3)

Having defined a plausible set of resources, the relation between resource and action is defined. The action associated with this resource context is to switch on the pump. An example of the way these assumptions were built into the model can be seen in the following example which specifies the resources associated with switching the directional pump to backward.

:: (resource[pumptype]==directional) &&
   (resource[pmpstate]==off) &&
   (resource[pmpdirection]==forward) &&
   (resourcescevol==empty) &&
   (resourcedstvol==full) ->
   {buttons!backward; buttons!on}

This action of changing the mode to backward is supported when the operator knows that the pump type is directional, that the pump state is off, that the pump is set in the forward direction, and that the source tank is empty and the destination tank full. The system model determines separately whether the action is enabled.

The proposed relations between resources and action (Step 3) are defined in PROMELA using a number of processes to describe differing assumptions made about the operator. Three operator processes were produced to explore alternative resourcing assumptions. The first operator process captures a number of assumptions about the information resources in relation to each pump action: what the state of the tanks is; which pump is currently accessible; and what the state of the pump is. This process also makes assumptions about the operator’s movement within the space, that (i) a pump is only visited once and (ii) one switch or collect only will take place at each visit.

E. Deriving the goals (Step 4)

The system is designed to manufacture products C and D. The analysis therefore concerns cognitively plausible paths that achieve these goals. This is achieved by checking LTL properties that the two products are eventually produced and transported from the relevant tanks, for example:
Variable $vtank$ holds the volume stored in each tank, $mtank$ the contents type. The property expresses that the goal of having tank 1 full of product C will never be reached. The verification will produce cognitively plausible traces that achieve this goal, if they exist.


**F. Exploring the paths (Step 5)**

For each operator processor capturing different resource constraints, the cognitively plausibility of the paths generated can be explored with the aid of domain and human factors input. This exploration includes an assessment of whether the resourcing assumptions are adequate. To achieve this, further properties may be analysed to identify intermediate paths towards achieving the goals. Hence

$!>(mtank[3]==A)$

can be used to explore whether the resource assumptions may lead to a hazardous state (remember that material A must not be pumped into Tank 3). In summary the process of analysis involves:

1) formulating goals
2) exploring paths
3) formulating additional properties to assess whether the resource constraints, as specified, permit alternative behaviors.

Exploring the initial operator assumptions with the model checker leads to a conclusion that the operator might also require confirmation that the appropriate material is in the tank. This prevents the possibility of moving the wrong material into the wrong tank. New information resources can be identified: $resource[desttnktype]$ and $resource[scetnktype]$ intending to communicate the type of material in the current source and destination tanks to the operator.

A second resource configuration with these additional assumptions can be explored, noting that the operator also (a) knows the order in which the pumps are required to be used to produce the material; (b) knows which pump is nearest to the operator’s current location. The position of the operator / device is specified in the model by the variable $position$. This variable, which is already part of the model, is now assumed to be an information resource. A further resource variable, the schedule, is defined as an ordered list of pumps to be used by the operator to achieve the goal. In the case of production of C the schedule is a list 1, 3, 5. Pump 1 must be used first to fill tank 1. Pump 3 must be used both to move the material to tank 2 and to move it back to tank 1 once processed. Finally pump 5 is used to move it back. The position resource represents the assumption that the operator can see where they are, which is important in carrying out certain actions. The schedule could be a checklist, or a notice on the wall of the plant, to remind operators of the order in which to do things.

Resourcing for a typical completion of an action (moving material A or B into tank 1) now has additional elements.

\[
(resource[pumptype]==volume) \land \\
(resource[pmpstate]==off) \land \\
(resource[dstvol]==full) \rightarrow \\
\{ complete=true; next++ \}
\]

The new operator definition describes not only what resources are required to complete the activity but also when the operator’s focus changes to the next stage in the schedule, indicated by a further variable $next++$. The requirement that the operator remembers where they are in the schedule in order to successfully operate the system is made explicit. Such assumptions again provide a point for discussion between human factors, domain, and software engineering experts on whether these are reasonable, or what changes to the user interface or system behavior are needed to remove the requirement for the operator to remember this information. There is an additional feature of the model that checks whether the current position is suitably resourced. The key elements of this resourcing are as follows.

1) $move?any$ specifies a move within the space. This move occurs if the next pump to be scheduled is neither displayed nor stored on the operator’s device. When a move takes place the new position is checked. If it is the next position to be scheduled it is collected. If it is contained elsewhere in the schedule but is not relevant immediately then it is stored. The pump is stored by collecting it then switching it into the storage position.

2) If the currently displayed pump is the next one on the schedule then the next action is defined as incomplete using the variable $complete=false$. This causes the operator to check which action is currently resourced. This process is otherwise identical to that defined in the earlier operator model.

3) If the pump that is stored is the next one on the schedule then it is switched with the currently displayed pump and the next action is defined as incomplete.

When a pump operation has been completed the operator moves on to the next item on the schedule. The operator checks to see whether the displayed or stored pumps are the next ones on the schedule. If they are not, the operator moves to a new location and checks to see whether the new position is the next item or is in the schedule at all. If it is the next item then it is collected and used otherwise if it is on the schedule it is stored. When the displayed pump is the next one in the schedule
then resource constraints, identical to those defined in the first version of the operator, are used to assist in deciding the next action.

Analysing the model with the revised operator definition is achieved by placing an assertion at the point in the new operator script at which the simple pump constraint described in Figure 10 is invoked.

This aims to produce a cognitively plausible path that includes the display switches that must take place so that the goal can be achieved. It indicates that if the stored value is the next item then it is switched with the currently displayed control. The path generated by the model checker for the goal of producing Product D involves the following steps (the actual trace is not included for reasons of brevity but may be seen by carrying out the appropriate analysis on the model at [5]).

1) moves to 4 and collects the controls
2) presses the switch button
3) moves to 2
4) presses the collect button
5) uses the up button until the limit is reached
6) uses the on button to turn the pump on — this turns pump 2 on, begins filling Tank 1 with Material B
7) when the pump switches itself off, then switches controls on the device
8) turns pump on (pump is already in forward mode) — this turns pump 4 on, begins filling Tank 3 with Material B
9) when pump turns itself off, presses the backward button
10) turns the pump on — this turns pump 4 on in backward mode, begins filling Tank 1 with Product D
11) when the pump turns itself off, presses the forward button
12) moves to 1
13) moves to 5
14) presses the collect button
15) presses the on button — this turns pump 5 on, pumps product D out of the plant
16) when the pump turns itself off the goal is achieved.

Inspecting the path can provide useful insights about what information is available to the operator at different stages in the process. Hence in the trace fragment in figure 11, from which the sequence above was extracted, it is possible to see how the first phase of moving material to tank 1 is resourced. This particular resourcing occurs when the limit in the volume pump is not set to maximum (see line 78). The state of the resources is that the user can tell that the volume pump is currently the available pump, that it is switched off and that the volume set for the pump is still less than the maximum (limit=5). In line 80 it can be seen that the operator presses the up button (to increase the volume). The lines relating to device refer to the device activities that occur as a result of the user initiated actions. Once that action is carried out the resource constraints are checked again (line 83) and (not visible in this fragment) the up button is pressed once more. By this means it is possible to check every step in the path that is generated to achieve the given goal. The trace indicates that as far as the operator is concerned each increase in volume is resourced separately by checking at each step what the new defined volume is. In practice it is likely that the required volume will be specified by tapping the volume button the required number of times without checking at each step. By this means assumptions about how the operator is likely to behave can be explored. Subtle distinctions regarding available information are important. A significant factor in the Three Mile Island accident was the fact that operators believed a valve was closed thanks to an indicator light - in fact, it indicated that the valve had been ordered to close, when it was stuck open [23].

As well as demonstrating the broad scope of the modeling approach, which is useful for addressing the rich context of user behavior, this example has also illustrated the iterative nature of the analysis method, and the way in which a number of different operator strategies may be explored, while at the same time making changes to the system design. It also illustrates how assumptions regarding the responsibilities placed on the user (such as remembering their place in the schedule) can be introduced to the analysis in a controlled way, in order to support analysis of likely user behaviors.

V. CONCLUSIONS

The resilience (or dependability) of a system does not depend only on the correctness of the system in terms of a set of pre-defined functional requirements. Folding the user into the analysis of a system’s resilience will help obtain a better understanding of how the system operates in practice.
A resource based approach can help in identifying potential usability problems by exploring what should be available at the interface to support users. A method has been presented for the inclusion of user behaviors into a model checking analysis. The method can be applied iteratively, making changes to the system design, and exploring different operator strategies. The presence of tool support not only makes it possible to explore complex systems which are difficult to reason about reliably "by hand", but also allows the possibility of searching through the many possible behaviors for those which are of more interest to the analyst. The presence of resources introduces a notion of plausibility and a more realistic conception of user behavior, which is based neither on rigid plan following nor random behavior. User behavior is shaped, in sometimes subtle ways, by system affordances and available resources, and by the user’s own goals and strategies.

Other approaches exist that attempt to fold human factors considerations into model-based approaches to usability reasoning. One possibility is to build an explicit model of the user as in programmable user models (eg. PUMA [3]) and executable cognitive architectures (SOAR [16], EPIC [14], etc.). While these may deliver more detail in respect to cognitive and performance aspects, they are also more complex to construct in a form that is automatable [24]. Another approach is to encode assumptions about the user directly into the model (compare [25]). In this case the separation between device model and user assumptions is not clear and can bias the user assumptions towards those that are needed to make the system work. By working with assumptions at a resource level, a clear separation is made between models and assumptions about users as expressed in terms of resources. These models are also relatively easy to build and straightforward to work with, providing a natural way for the HCI expert to contribute to a more rigorous analysis.

The analysis presented complements a more unconstrained style of analysis where all possible behaviors of the system are analysed (which is useful when considering safety issues, for example). However, unlike many safety-oriented analyses, this approach is not about “proving” the system to be usable, but rather identifying and investigating plausible and interesting situations and behaviors in order to find and fix usability problems (e.g. situations where insufficient resources are available to the user) and to investigate the effectiveness of different user strategies for achieving goals.

The two examples have shown the feasibility of the approach with different modeling languages and verification tools. Efforts are currently under way to apply the approach to a number of other systems. These systems include devices from everyday life, and safety critical systems that are being explored in collaboration with industrial partners. Development of the approach is continuing in a number of directions. Of particular interest is to extend the approach to multiple users and to make the approach easier to user by designers.

- Systems typically involve multiple collaborating users who may share some resources, and for whom some resources are private. Tool support for the analysis of multiple concurrent activities and their resourcing is therefore of interest. Analysis of the kind described in this paper should make it possible to understand conflicts and interference between different activities.
- The transition between different resource configurations in the process of completing an activity is also an interesting feature of the design of an interactive system. [28] discusses the role of interaction strategies such as plan following or goal matching. Strategies such as these are of interest because they provide a more holistic view of interaction and the way that it is resourced. It enables consideration of inconsistencies and consistencies over sets of actions which could be further explored through the present analysis.
- Finally, tools that support the routine analysis of interactive systems are of interest. The IVY tool [7], [8] has been developed to enable system designers to analyze interactive systems against a battery of usability properties. These properties are currently checked against all possible paths, not necessarily plausible paths. A relatively simple step is to provide support for resourcing in this system.

REFERENCES
