Analysing Mode Confusion: An Approach Using FDR2

Bettina Buth[1,2]

[1] BISS, Bremen Institute for Safe Systems
bb@tzi.de
[2] EADS SPACE Transportation, Bremen

Abstract. Mode confusion situations or more general automation surprises can arise in the context of sophisticated control systems which require the interaction with human operators as for example flight monitoring systems in airplanes. A "mode" is defined by a subset of system variables the values of which determine distinguishable forms of system behaviour. Critical situations can arise if the operator interacts with the system assuming a wrong mode. The identification and analysis of such situations needs to take into account both the system design and the operators mental model of the system. Recent research showed that model-checking techniques are useful for identifying mode-confusion situations. Two different approaches can be found: the first tries to identify mode confusion potential in system design, the second analyses actual mode confusion situations to identify the discrepancies between the mental model of operators and the system design. This paper reports an experiment in using the model-checker FDR2 for comparing system and mental models based on CSP refinement. In contrast to earlier attempts using model-checkers for this task, this approach allows a direct comparison of the two models which can be easily derived from a rule-based description.

1 Introduction

The ever-increasing complexity of computer-based systems has lead to a changed role of human operators, especially in safety-critical applications such as air-craft and train control, chemical and nuclear plants, medical equipment, or automobile components. The use of computers in such systems has a high potential for automation as well as extended functionality, but also requires sophisticated control and monitoring mechanisms due to the inherent complexity. These themselves can be implemented as computer-based processes which allow to take away the strain from human operators who would otherwise have to cope with a multitude of information and a higher demand on reaction times required for the interaction with such systems.

Nonetheless, in many applications a total automation of the system control is not accepted or (not yet) possible. Human operators are often the ultimate instance for dealing with emergencies or have to provide necessary information not directly available to the computer-based kernel systems. Research activities

M. Heisel et al. (Eds.): SAFECOMP 2004, LNCS 3219, pp. 101–114, 2004.
© Springer-Verlag Berlin Heidelberg 2004

in the Human Factors community focus on human-computer interfaces based on psychological as well as design-oriented considerations. As Sarter, Woods and Billings [1] and Leveson et.al. [2] point out, technology-centred automation potentially leads to designs which are problematic for the human interaction.

One area which has found attention during the last years is the investigation of so called *automation surprises*, particularly *mode confusion*. This paper discusses the use of model-checking for the comparison of abstract system models and mental models with the objective to analyse mode confusion situations. The emphasis is on the specification and model-checking aspects rather than on the socio-technological perspective. The remainder of the introduction provides the background for the approach as well as an informal description of the example, a *kill-the-capture* scenario. Section 2 describes one possible approach to the analysis of this example using FDR2. Section 3 summarizes the experiences and tries to generalize the results.

1.1 Mode Confusion Analysis – Background

Modes are identifiable and distinguishable states of a system which differ with regard to the effect of interactions. The complexity of a system is reflected in a large number of different modes and complex rules for mode transitions as well as functionality in a mode. Mode confusion scenarios or in general automation surprises describe situations where the operator's assumption about the system mode differs from the actual mode of the system and actions performed under this assumption result in critical situations. In order to detect and eliminate mode confusion, a thorough analysis of the system design and functionality as well as the human-computer interface is required.

Techniques from the formal methods field prove to be useful for mode confusion analysis. Several approaches based on abstract models of the system are documented; see e.g. Leveson et.al. [2], Miller and Potts [3], or Lüttgen and Carreño [4]. These experiments focus on the identification of situations that potentially lead to mode confusion, particularly the identification of categories of indicators for mode-confusion situation and use an abstract model of the system as starting point. Rushby [5,6] suggests a complementary use of model-checking based on two different models of the system. The actual model is an abstract model of the actual system behaviour; the second reflects the mental model of the operator which may be a reduced version of the full model or even may contain wrong assumptions about the system.

In contrast to the approaches of Leveson and Miller and Potts, Rushby's approach aims at identifying critical discrepancies between the models rather than investigating the mode confusion potential of the actual model. Such differences reflect deviations in the observed behaviour of the models which can point to potential mode confusion situations. Rushby formalizes the models in the Murϕ [7] model-checker notation and employs Murϕ to perform a full state exploration. The system not only uncovers the flaw already known from the analysis by Leveson, but also detects several other problems of the suggested

corrections. The model can also be enhanced by incorporating a more specific operator behaviour.

Since Murϕ is not able to compare two models directly, a usual trick is used: both models are merged into one by renaming the relevant state components such that these have disjoint names. The Murϕ rules then are used to describe the effect of inputs or events to the full set of state variables.

This is slightly unsatisfactory, since an untrained person will not be able to determine such a specification from the distinct views of the actual and mental models respectively, even if the Murϕ rules can be easily understood with a basic knowledge of state transition machines or simple automata. Similarly, the formalization of invariants as criteria for the absence of mode confusion will in general require some explanation or even a manual analysis of the models (which may very well uncover the problems in the models). Rushby himself [5,6] suggests to employ a different type of model-checker, namely the CSP-based tool FDR2 as an alternative, since FDR2 allows to compare models in a more direct way. In the following, this suggested approach is investigated, taking the Murϕ model as a starting point.

1.2 The Example

The example in Rushby's papers [5,6,8] is taken from an article by Palmer [9], which reports two cases of altitude deviation scenarios. These cases and three others were observed in a NASA study in which several crews flew realistic missions in DC-9 and MD-88 aircraft simulators. This example has previously been investigated by Leveson [10].

In the following, the scenario description as stated by Rushby [5,6] is presented, which is the starting point for the Murϕ model. In order to follow the scenario it is necessary to explain some features beforehand. The `PITCH` mode is a control element for the autopilot which determines the climbing behaviour of the aircraft. The modes are

`VERT SPD`	vertical speed; climb at a specified rate (feet per minute)
`IAS`	indicated air speed; climb at a rate which is consistent with holding the air speed (knots)
`ALT HLD`	altitude hold; hold current altitude
`ALT CAP`	altitude capture; provide smooth levelling off when reaching desired altitude

The second relevant component is the `ALT` capture mode (one of several possible capture modes) indicates that the aircraft should climb to the the desired altitude and then hold that altitude. For the example it suffices to imagine this mode as a binary value which reflects whether the mode is set (armed) or not.

The interaction between the modes is of particular interest:

- if `ALT` capture is armed and the desired altitude is reached, the pitch mode is set to `ALT HLD`.

- the `ALT CAP` pitch mode is entered automatically when the aircraft gets near the desired altitude under the condition that `ALT` is armed; it switches off the `ALT` capture mode
- if `ALT CAP` pitch mode is set and the desired altitude is reached, the aircraft levels off and pitch mode is changed to `ALT HLD`.

The scenario as reported by Palmer [9] describes a potentially critical situation where an aircraft leaves its assigned flight corridor and enters a flight altitude which could be assigned to other aircrafts. The cause for this situation is obviously that the `ALT` capture was switched off without the Captain noticing it (the only information provided is that the `ARM` window switches to blank). Analysis of the situation shows that the interaction between pitch modes and `ALT` capture mode is more complex than first assumed.

Leveson and Palmer [10] present the essential information of this incident as presented in Fig. 1. Note that the information presented is only part of the overall interface of the pilot, especially the control instruments for setting speed and target altitude are placed on a separate panel (the Mode Control Panel) which also presents some of the information available from the FMA.

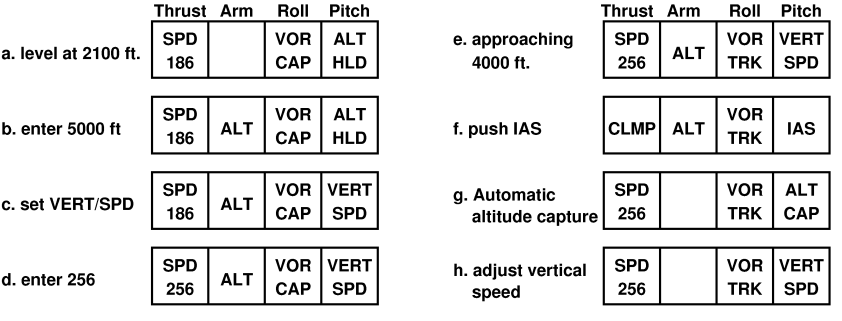

Fig. 1. FMA Displays for the Example Incident

Rushby [8] also derives a state machine representation of the abstract behaviour of the autopilot with regard to pitch mode and altitude capture mode. This model takes into account the relevant modes and the inputs of both the plane crew and the events from the environment. This model, which is shown in Fig. 2 abstracts from the general status of the plane, as for example altitude, speed, motion or similar and from related values the pilot could enter. What remains is an abstraction of the behaviour focused on pitch mode and capture mode restricted to `ALT`. Similarly, Rushby provides a state machine representation of the mental model as derived from the case study. This is shown in Fig. 3.

The obvious difference between the two models is the number of states. The mental model does not contain an explicit state for `ALT CAP`, the pitch mode which is entered automatically without pilot interaction. This omission models the fact that the pilot was not aware of this particular mode and the related

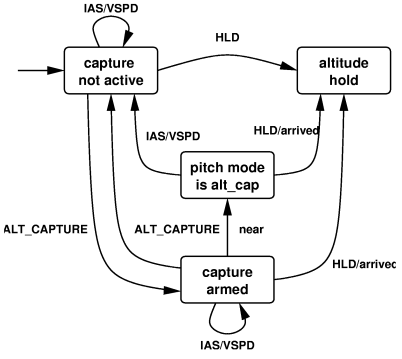

Fig. 2. State Machine for Actual Model

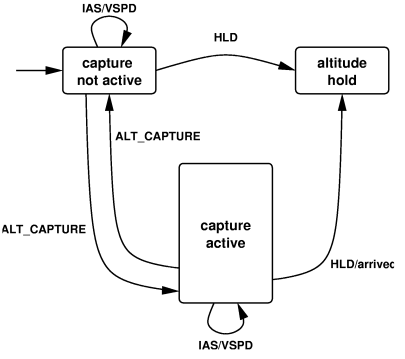

Fig. 3. State Machine for Mental Model

changes to the ALT capture mode. A formal analysis of these automata models with regard to their language equivalence will also reveal deviations in the possible mode transitions, but further analysis is required to examine whether these differences are indeed critical.

2 FDR2 Formalization and Analysis

This section presents an approach of employing FDR2 [11] for the investigation of the mode confusion situation reported by Palmer [9]. The Murϕ specification described by Rushby provided the starting point. The main objective of the experiment is to evaluate the benefits of FDR2 with regard to its ability to compare two models.

For this purpose several alternative approaches were investigated, which involved both rule-based and automata-based models, where the latter is based in the automata for actual and mental model as presented in Fig. 2 and 3. For the experiment the error situations uncovered by FDR2 were compared to the problems found using the Murϕ system. In the following, the rule-based description of separate actual and mental model are presented and analysed. A full report of this and the other approaches can be found in Buth [12].

2.1 Actual and Mental Model – A Rule-Based Specification

The objective of this subsection is to investigate the possibilities of checking mental and actual model by comparing them with regard to refinement properties of CSP models.

The specification language employed by FDR2 is $\mathrm{CSP_M}$, a machine-readable version of CSP. The core concepts of these specifications are processes which communicate over synchronous channels. State spaces in form of global variables are not directly available in CSP, but $\mathrm{CSP_M}$ allows the parametrized specification of processes.

The first definitions of the specification introduce the data type pitch-mode similar to the type in the Murϕ specification and a set of channels, which in this case represent the possible events from the environment (pilot input and external events). The event names are chosen to correspond to the rule names in the Murϕ specification.

```
datatype pitch_modes = vert_speed | ias | alt_cap | alt_hld
channel ALT_CAPTURE, HLD, IAS, VSPD, near, arrived
channel obs : pitch_modes.Bool
```

The additional channel `obs` of observables is introduce to make the mode transitions visible in the model traces. It is required in addition to the usual observable event traces over the channels to monitor the progress of the system. The motivation is that the information about the state transitions is essential for identifying the causes for error situations in the check.

Two distinct processes `ASYS` and `MSYS` are specified for the actual and the mental model, respectively. The state spaces of both models are determined by the pitch-mode and the capture-mode, which are stated as parameters of the processes. Note that the changes of these two state components are observed via the above mentioned channel `obs`. The initial value is a non-deterministic selection from the set of pitch-modes without the `alt_cap` mode for the pitch-mode and `false` for the capture-mode.

```
ASYS = |~| p: diff (pitch_modes, alt_cap) @ ASYS1 (p, false)

ASYS1 (pitch_mode, capture_armed) =
    obs!pitch_mode.capture_armed ->
    (
     ALT_CAPTURE -> ASYS1 (pitch_mode, not capture_armed)
     []
     HLD         -> ASYS1 (alt_hld, capture_armed)
     []
     IAS         -> ASYS1 (ias, capture_armed)
     []
     VSPD        -> ASYS1 (vert_speed, capture_armed)
     []
     (capture_armed) & near -> ASYS1 (alt_cap, false)
     []
     ((pitch_mode == alt_cap) or capture_armed) & arrived
             -> ( let pm = if ((pitch_mode == alt_cap) or capture_armed)
                           then alt_hld
                           else pitch_mode
                  within
                   ASYS1 (pm, false)
                )
    )
```

Note the way the boolean guards are used to express that certain transitions are not possible under all circumstances. Here, additional constraints as derived from

the automata model in Fig. 2 have been used to provide a more complete model. The event `near` only has effect if the capture-mode is active. Similarly `arrived` is regarded if the capture-mode is active or the pitch-mode already is `alt_cap`.

The mental model works with a reduced set of pitch-modes, which does not include the `alt_cap` mode; here a subset of the full set of pitch-modes is defined. The process `MSYS` then is defined analogously to the process `ASYS`, omitting the `near` event and the behaviour in the ignored `alt_cap` pitch-mode.

```
pm_ideal = {pm | pm <- pitch_modes, pm != alt_cap}

MSYS = |~| p: pm_ideal @ MSYS1 (p, false)

MSYS1 (pitch_mode, ideal_capture) =
   obs!pitch_mode.ideal_capture ->
    (
     ALT_CAPTURE -> MSYS1 (pitch_mode, not ideal_capture)
     []
     HLD         -> MSYS1 (alt_hld, ideal_capture)
     []
     IAS         -> MSYS1 (ias, ideal_capture)
     []
     VSPD        -> MSYS1 (vert_speed, ideal_capture)
     []
     (ideal_capture) & arrived
                 -> MSYS1 (alt_hld, false)
    )
```

This specification of the mental model could easily have been derived from a set of rules or an automata representation such as the one in Fig. 3. In particular, it can be derived without information about the actual model. Note that for the analysis of the mode-confusion situation such a rule-based or automata model will originally needs to be derived from the understanding of the operator rather than other material. For this study, the mental model was taken from the Murϕ example.

In order to prove that the models are equivalent, it is necessary to map them to a common set of observable events. This means that the event `near` which is not visible in the mental model as well as the artificial events related to the invisible pitch-mode `alt_cap` need to be hidden from the interface of the actual model. This is done using the hiding construct in $\mathrm{CSP_M}$:

```
Mental_Model = MSYS
Actual_Model = ASYS \ {|near, obs.alt_cap|}
```

Now it is possible to specify the desired equivalence. Since FDR2 only allows to check refinement in any of the three models (traces, failures or failure-divergence; see [13,11] for details), it is necessary to prove the two directions of the equivalence separately. Furthermore, it is of interest to prove equivalence both in the trace as well as the failures model. Trace equivalence which is checked as mutual

trace refinement ([T=) in FDR2, only ensures that both systems are able to perform the same sequences of events. For the given example this ensures that both systems are able to react to external inputs in the same way and that the state changes are performed accordingly (by observing the obs events). In addition, it is of interest to know whether at any point in time one of the models could refuse an event which can not be refused by the other. This is not of direct interest for the artificial state events, but could point to a problem with the inputs and external system events. Refusal properties are checked using failures refinement ([F=) in FDR2. The following checks were performed in the example:

```
assert Actual_Model [T= Mental_Model
assert Mental_Model [T= Actual_Model
assert Actual_Model [F= Mental_Model
assert Mental_Model [F= Actual_Model
```

Of these checks, the first and third succeed, while the second and fourth fail. FDR2 reports the following error scenarios, which are the same for the two failing checks. The process the behaviour of which is reported in the following is the process ASYS. The behaviour of MSYS is not explicitely reported for these errors, which shows that MSYS is able to perform the same traces except for the last event. Note that the traces of Actual_Model are subtraces of these traces, namely those where the hidden events are removed.

```
BEHAVIOUR:
Performs        <_tau, obs.ias.false, ALT_CAPTURE, obs.ias.true,
                 near, obs.alt_cap.false, IAS, obs.ias.false>
```

Analysis of this first error in connection with the specification of MSYS shows that the event obs.ias.false is not a possible event since the capture-mode is still true after ALT_CAPTURE, which is the last relevant event from the perspective of MSYS. This means that FDR2 detects the same kind of error as was found be Murϕ and the combined specification. The advantage of this usage of FDR2 is the documentation of other possible errors. The following two are but variants of the first case, where the final state is related to the pitch-modes vert_speed and alt_hld instead of ias.

The next three cases require further analysis of both actual and mental model. The first such error trace of ASYS is reported as follows:

```
BEHAVIOUR:
Performs        <_tau, obs.ias.false, ALT_CAPTURE, obs.ias.true,
                 near, obs.alt_cap.false, ALT_CAPTURE,
                 obs.alt_cap.true, IAS>
```

The other two traces vary in the initial state, where the different pitch-modes are taken. All of them show the same sequence of external events, namely

```
<ALT_CAPTURE, near, ALT_CAPTURE, IAS>
```

where IAS is the event which is refused by MSYS and thus Mental_Model. Without regarding the auxiliary obs events it is not possible to understand the underlying problem. Thus it helps to have a look at the trace of Actual_Model:

```
BEHAVIOUR:
Performs        <_tau, obs.ias.false, ALT_CAPTURE, obs.ias.true,
                 _tau, _tau, ALT_CAPTURE, _tau, IAS>
```

This perspective reveals that Actual_Model is able to perform the IAS event directly after the ALT_CAPTURE event, without an intermediate obs event. This is definitively not possible for MSYS respectively Mental_Model. The essential question here is how to interpret these error situations.

Obviously, the problem is connected to hiding the obs.alt_cap events, which means that it is related to the problem that one of the pitch-modes is invisible. Since the ALT_CAPTURE event only changes the capture-component of the state space but does not lead to a state transition, the full effect is hidden. Investigation of the behaviour of MSYS shows that the second ALT_CAPTURE event leads to the event obs.alt_hld.false in the mental model for the first error case and similar for the other cases. Several things need to be considered here:

- the mode-confusion problem is not only related to the inconsistency in the pitch-modes of the models; the different capture-modes pose a problem in itself. Although the invariant used in the Murϕ model would not be violated if the pitch-modes were the same, a following IAS event would immediately lead to an error trace as the first three reported above.
- what is the desired reaction to an ALT_CAPTURE during the alt_cap pitch-mode? The Murϕ model does allow changes of the capture-mode during the approaching phase after near, but is that a suitable abstraction of the behaviour? Comparison with the models as described in Fig. 2 and Figure 3 does provide a hint that this abstraction could indeed be a problem. While the mental model allows to switch between the "capture active" and "capture not active" state, this is not allowed for the "pitch mode is alt_cap" state. This suggests that the second ALT_CAPTURE event should not have been possible in ASYS. Actually, Rushby detects this problem in one of the later versions of the Murϕ model and assumes that ALT_CAPTURE should not be possible in pitch-mode alt_cap. The FDR2 specification can be corrected in a similar way:

  ```
  ...
       (pitch_mode != alt_cap) & ALT_CAPTURE
                    -> ASYS1 (pitch_mode, not capture_armed)
  ...
  ```

  By guarding the event ALT_CAPTURE in the actual model in this way, the error cases can indeed be reduced to the original three which are directly related to the mode confusion situation as related to the capture-modes.
- can the specification be modified in a way that allows to cope with such missing events; more generally: is there a way to deal with hidden state

changes in the case where events lead to explicit state changes in the second model? Up to now no general solution has been found for this problem.

After performing the corrections analogously to the suggestions for the Murϕ model, the resulting specification still shows an error, a refusal error for the check

```
Mental_Model [F= Actual_Model
```

Checking the traces and refusals reveals that while the actual model could engage in any external event but `ALT_CAPTURE` or `near`, the mental model could also engage in `ALT_CAPTURE`. Further analysis of this situation in comparison with the error-free Murϕ model reveals a flaw in that model: rule ‘‘`ALT CAPTURE` in the Murϕ specification reads as follows:

```
rule "ALT CAPTURE" pitch_mode != alt_cap ==>
begin
  capture_armed := !capture_armed;
  ideal_capture := !ideal_capture;
end;
```

But this means that the behaviour of the mental model, namely the changes to state variable `ideal_capture` are influenced by the value of `pitch_mode`, which is not part of the mental model. The FDR2 error shows the effect of the change to the actual model alone and reveals a new error situation. This error-situation is due to the the change with regard to the error found above: guarding `ALT_CAPTURE` in the actual model prevents a second such event in pitch-mode `alt_cap`, but in the mental model such a change is allowed. Thus the corrections still do not capture the problems arising from the hidden state properly.

3 Lessons Learned

The previous section discusses one possible approach to the analysis of the case study presented by Palmer [9] using CSP specifications and FDR2 for checking them. The specifications are based on the idea of Rushby [5,6,8] to compare abstract formalizations of actual system and mental model. This section tries to summarize and generalize the experiences using FDR2 the suitability of model-checking for mode-confusion analysis in general and the exploitation of mode-confusion analysis for system design.

3.1 Evaluating the FDR2 Approach

The essential difference between the Murϕ and the CSP_M specification is the way in which the models are compared. The FDR2 specifications allow a separate specification of actual and mental model, while the Murϕ specification presents a view of the combined models with a partially shared state space.

For both approaches it is necessary to determine how mode confusion situations can be identified, which parts of the specifications need to be observed

to detect such a critical situation. The study presented Buth [12] contains three different variants for the example: the first directly corresponding to the Murϕ version, a second separating the actual and mental model but still using the rule-based description as basis, and a third directly derived from the automata representation as given in Figures 2 and 3.

The overall experience with modelling these versions in FDR2 is quite encouraging: each of the models requires little effort for a CSP expert or even someone with a general specification background. Similarly, the evaluation of error scenarios reported by FDR2 does not pose any particular obstacles assuming a basic understanding of the overall system functionality. The following paragraphs summarize the experience with the different models.

The combined as the original Murϕ model requires the formalization of an invariant. The definition of such invariant properties already requires some form of analysis in order to correlate the state spaces of the two models. The approach of specifying two independent models seems to provide a more direct way for the comparison. It is not necessary to define the invariant explicitely; the models can be compared with regard to their external behaviour or - as shown above - additionally with regard to the values of their state components. The general assumption for this approach is that a critical situation only arises if the models react in a different way to their environment. The errors found are essentially the same as by Murϕ. The interesting error from the point of view of the comparison is the one not found by Murϕ. This error can not be detected in the Murϕ model due to the introduction of a dependency between the state transitions of mental and actual model in rule `ALT CAPTURE` of the combined model. The dependency does not exist in reality where the models can not influences each other. This modelling error can not occur in a model using two separate processes, since the state spaces of the processes are not directly related.

The difference of the automata model to the rule-based FDR2 specifications and the Murϕ model is the handling of states. The automata model does not work with the full state-space as defined by the cross-product of the possible values of the modes, but starts from a more realistic view where similar states are combined into one state and labelled according to their meaning. The specification of such models from given automata models is straightforward and could even be done automatically provided a suitable presentation of the automata is available.

While the automata-based approach does not have any disadvantages with regard to the errors found, the decision for the automata style does essentially depend on the availability of such a model. Often, it will be easier to capture a set of rules in the style of the rule-based FDR2 specification than to develop the automata view if it is not already available. The capture of rules in this way has the advantage of allowing incompleteness and non-determinism without explicit consideration - at least for an initial version of the specification. A refinement of the specifications in order to exclude unrealistic behaviour will in both cases require the same kind of considerations.

Note that a combination of the rule-based and automata-based specification styles is in general possible. This would allow to use an automata model for the actual system, where such automata presentations may be available from the system design documents, while specifying the mental model based on rules. In general, the main problem for the comparison of models will be the definition of the interrelation between the two systems; this includes the identification which events must be hidden and which renaming should be used to facilitate a comparison. But this is independent from a particular specification style.

3.2 Model-Checking for Mode Confusion Analysis

A first conclusion from the experiments with FDR2 is the confirmation of Rushbys résumé: model-checking provides a relatively easy approach to investigating models with regard to mode confusion situations. One particular benefit is the ease with which the models can be adapted and extended in order to check potential corrections. At least with the given example the model-checker provides almost immediate feedback on error situations.

Two essential questions need to be discussed with regard to the general usage of model-checking for this kind of task:

- How can the specifications for mental and actual model be derived in a systematic way and on basis of which input?
- How can the errors found by model-checking be related to situations in the real system?

Both questions are strongly connected to the topic of suitable abstraction for both the real system and the operators understanding of the system. With regard to the application of a model-checking tool, the specifications should be as abstract as possible, restricted to the minimal set of state and environment information. This is a prerequisite for a successful application of a model-checking tool since too much information will in general lead to a state explosion and thus to potential problems with the state-exploration approach.

A discussion of the questions concerning abstraction and error analysis in relation to the adequacy of the models for the presented example can be found in Buth [12]. The general conclusion is that the suitability of the abstraction and form of specification depends on the concrete application and the knowledge of the people involved; a systematic approach will only be possible when more experiences with this use of formal methods in the framework of human-computer interfaces are available.

3.3 Evaluating the Results from Mode Confusion Analysis

One essential topic not yet addressed is how the results of mode confusion analysis such as presented above could be used. This question is directly related to the goals of such an analysis and these need to be correlated to the development phase in which the analysis takes place. Essentially there are two possible objectives:

- using mode-confusion analysis during the design phase of a system to ensure an adequate design without or at least with a minimum of mode-confusion potential
- analyse critical situations encountered during the integration, acceptance or operation phase of such a system.

A comparative approach as presented in this paper requires the existence of two models. A mental model will not generally be available during the design of a new system. It may be available for new developments in domains where the user interfaces are standardized to a certain extent, as for example aircraft instrument panels, automobile or train control elements. A mental model can also be derived from training material of an existing system or from rules captured in interviews. In most cases a mental model will be derived from discussions with an operator, i.e. after deployment of the system under investigation.

If no such models are available during the design phase, two further alternatives are possible. Model-checking can also be employed to check for mode confusion indicators in an abstraction of the system design as suggested by Lüttgen and Carreño [4], an idea which is influenced by the approach of Leveson [2]. Alternatively, mode confusion analysis could also be used to derive a minimal mode confusion free mental model of new or existing systems. The idea has already been presented by Rushby [5,6,8]. He outlines the use of such minimal models for the evaluation of designs: if even the minimal mental model is very complex this could be an indicator for the inadequacy of the system design from the point of view of the human-computer interface.

In any case identified mode confusion situations should lead to changes in the design; the objective is to prevent the introduction of mode confusion potential in the implementation. For the necessary modifications of the interface, the design, or the operation procedures, results from the human-computer-interfaces community should be taken into account.

3.4 Limits of the Model-Checking Approach

Although the results presented in this study are very exciting and point to a very interesting direction of using model-checking and formal methods in general, some remarks are due with regard to the applicability of this approach. The example considered here as well as those discussed by the other authors, are fairly small parts of larger and more complex systems. It requires more examples to prove that the approach scales to realistic applications. This is essential for many of the potential uses of the mode confusion analysis discussed above, but particularly for the validation of designs.

References

1. Sarter, N., Woods, D., Billings, C.: Automation surprises. In Salvendy, G., ed.: Handbook of Human Factors and Ergonomics. Second edition edn. John Wiley and Sons (1997)

2. Levevson, N.G., Pinnel, L.D., Sandys, S.D., Koga, S., Rees, J.D.: Analyzing software specifications for mode confusion potential. In Johnson, C.W., ed.: Proceedings of a Workshop on Human Error and System Development, Glasgow, Scotland. Glasgow Accident Analysis Group, Technical Report GAAG-TR-97-2 (March 1997) p. 132–146
3. Miller, S., Potts, J.: Detecting mode confusion through formal modeling and analysis. Technical Report NASA/CR-1999-208971, NASA Langley Research Center (January 1999) available at `http://shemesh.larc.nasa.gov/fm/fm-pubs-larc.html`.
4. Lüttgen, G., Carreño, V.: Analyzing mode confusion via model checking. Technical Report NASA/CR-1999-209332, ICASE Report No. 99-18, ICASE - NASA Langley Research Center (May 1999) available at `http://shemesh.larc.nasa.gov/fm/fm-pubs-icase.html`.
5. Rushby, J.: Using model checking to help discover mode confusions and other automation surprises. In Javaux, D., ed.: Proceedings of the 3rd Workshop on Human Error, Safety, and System Development (HESSD'99), University of Liege, Belgium (1999)
6. Rushby, J.: Using model checking to help discover mode confusions and other automation surprises. Reliability Engineering and System Safety **75** (2002) 167–177 Available at `http://www.csl.sri.com/users/rushby/abstracts/ress02`.
7. Dill, D.: The Murϕ verification system. In Alur, R., Henzinger, T., eds.: Computer Aided Verification, CAV'96. Volume 1102 of LNCS., Springer-Verlag (1996)
8. Rushby, J., Crow, J., Palmer, E.: An automated method to detect potential mode confusions. In: 18th AIAA/IEEE Digital Avionics Systems Conference, St Louis, MO (1999)
9. Palmer, E.: "Oops, it didn't arm." A case study of two automation surprises. In Jensen, R.S., Rakovan, L.A., eds.: Proceedings of the Eightth International Symposium on Aviation Psychology, Columbus, OH. The Aviation Psychology Department of Aerospace Engineering, Ohio State University (April 1995) p.227–232 available at `http://human-factors.arc.nasa.gov/IHpersonnel/ev`.
10. Leveson, N.G., Palmer, E.: Designing automation to reduce operator errors. In: Proceedings of the IEEE Systems, Man, and Cybernetics Conference. (1997)
11. Formal Systems (Europe) Lts: FDR2 User Manual. (1997) Available under `http://www.formal.demon.co.uk/fdr2manual/index.html`.
12. Buth, B.: Formal and Semi-Formal Methods for the Analysis of Industrial Control Systems. Volume 15 of BISS Monographs. (2002) (Habilitationsschrift submitted May 2001).
13. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice-Hall International (1998)