Safety-Relevant Mode Confusions – Modelling and Reducing Them

Jan Bredereke *, Axel Lankenau

Universität Bremen, FB 3 · P.O. box 330 440 · D-28334 Bremen · Germany

Abstract

Mode confusions are a significant safety concern in safety-critical systems, for example in aircraft. A mode confusion occurs when the observed behaviour of a technical system is out of sync with the user's mental model of its behaviour. But the notion is described only informally in the literature. We present a rigorous way of modelling the user and the machine in a shared-control system. This enables us to propose precise definitions of "mode" and "mode confusion" for safety-critical systems. We then validate these definitions against the informal notions in the literature. A new classification of mode confusions by cause leads to a number of design recommendations for shared-control systems. These help to avoid mode confusion problems. Our approach supports the automated detection of remaining mode confusion problems. We apply our approach practically to a wheelchair robot.

Key words: human factors, mode confusion, shared-control systems, safety-critical systems

* Corresponding author.
Email address: brederek@tzi.de (Jan Bredereke).
URL: www.tzi.de/~brederek (Jan Bredereke).

1 Introduction

Many safety-critical systems today are shared-control systems. These are interdependently controlled by an automation component and a user. Examples are modern aircraft and automobiles. Shared-control systems can cause automation surprises, and, in particular, mode confusions.

The American Federal Aviation Administration (FAA) considers *mode confusion* to be a significant safety concern in modern aircraft. For instance, consider the crash of an Airbus A320 near Strasbourg, France, in 1992 [1]. Probably due to heavy workload because of a last-minute path correction demanded by the air traffic controller, the pilots confused the "vertical speed" and the "flight path angle" modes of descent. The display read "3.3", meaning 3,300 feet per minute. But the crew intended to descend at 3.3 degrees, which translates into about 1,000 feet per minute. There was no ground view due to night and poor weather. As a result, the Air Inter machine descended far too steeply, crashed, and 87 people were killed. Another example is the often cited kill-the-capture bust [2]: an MD-88 jet plane was supposed to climb to 5,000 feet. The captain set the capture mode of the autopilot for this. But the aircraft climbed dangerously higher than 5,000 feet. The captain had adjusted the vertical speed before. This had disarmed the capture mode without the pilot's knowledge. The literature contains considerable research work on mode confusions. Nevertheless, it remains surprisingly unclear what a mode confusion actually is.

This article is structured as follows. Section 2 introduces to the research work on mode confusions in safety-critical systems. We then present a rigorous

definition of mode confusion in Section 3. This allows us to classify mode confusions in Section 4, and Section 5 uses this classification to derive recommendations for avoiding some of the problems. Section 6 validates our definitions against the informal notions in the literature. Our approach supports the automated detection of remaining mode confusion problems; we therefore apply this practically to a wheelchair robot in Section 7.

2 Mode Confusions in Safety-Critical Systems

A mode confusion occurs when the observed behaviour of a technical system is out of sync with the user's mental model of its behaviour. We now introduce to mental models of behaviour, we give an informal intuition of the meaning of mode confusion, and we briefly recapitulate the pertinent research results on mode confusions.

2.1 Mental Models of Behaviour

People form internal, *mental models* of themselves and of the things with which they are interacting [3]. (The term "mental model" has also another, different meaning in the pertinent literature. We refer to the above one, introduced by Norman [3].) There is ongoing research on the nature of such mental models, and on how people use them when interacting with their environment.

Here, we restrict our interest to mental models of the behaviour of a technical system, in particular of an automated system. We exclude the aspects unrelated to behaviour. For example, we are not interested in how people mentally represent spatial relations. We concentrate on shared-control, automated,

technical systems, because many safety-critical systems are shared-control systems. We concentrate on their behaviour, because the notion of safety is usually defined with respect to the behaviour, for these systems. An advantage of this restriction is that we have powerful mathematical tools for analyzing models of behaviour.

Mental models of the behaviour of technical systems appear to be based on state transition rules. This motivated many experiments to derive an explicit description of a mental model, in form of a *state machine* with modes and mode transitions. Mental models have been extracted from training material, from user interviews, and by user observation. For example, Cañas *et al.* [4] survey work on this. They also performed three experiments with 115 participants. They exposed these users to different knowledge elicitation tasks and made conclusions about their mental models.

Extracting a mental model from an individual user is notoriously difficult and expensive. In particular, mental models are unstable [3]. The user constantly learns and therefore adapts his/her mental model. The user also forgets. Furthermore, the model which is the user's long-term knowledge, the *conceptual model*, is different from the user's current working abstraction. When performing a task, the user concentrates on the part of his/her knowledge which he/she assumes to be relevant [4].

Nevertheless, even imperfect descriptions of mental models have value. If they are extracted from individuals by interview or by observation, they will have some randomness. If they are extracted from training material, they are generic only. The value is that any mode confusion problem showing up here has some chance to repeat itself with other individuals. We should therefore try to find

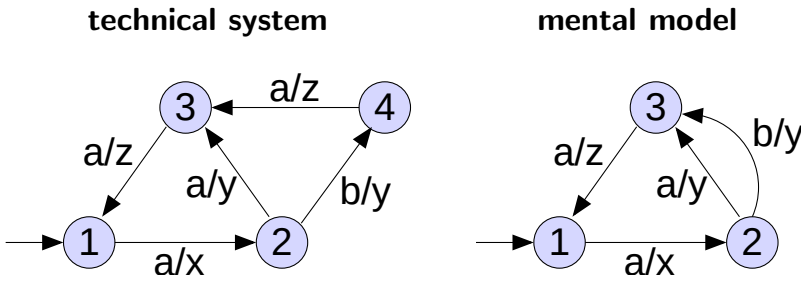
its causes and tackle them. This is in line with Rushby's argument on this issue [5]: "most automation surprises reported in the literature are not the result of an errant operator holding a specific and inaccurate mental model but are instead due to the design of the automation being so poor that *no* plausible mental model can represent it accurately." A basic assumption of our work is that one can produce descriptions of mental models at all that have at least some resemblance to the actual mental models of individuals.

2.2 An Informal Intuition of Mode Confusion

Intuitively, a mode confusion occurs when the observed behaviour of a technical system is out of sync with the user's mental model of its behaviour. Figure 1 shows the modes and mode transitions of some technical system and of some mental model of its behaviour. These automata are "similar": for the sequence of inputs given first, the outputs are the same. But the mental model misses one mode. For the second input sequence, the observed behaviour is different from the expected behaviour. The user will be surprised, probably unpleasantly. Even more, the surprise happens only later (after the fourth input), not when the modes actually get out of sync (after the second input). In Figure 1(a), circles denote modes, and arrows denote mode transitions, labelled with the corresponding inputs and outputs.

2.3 Survey of Work on Mode Confusions

We briefly recapitulate the pertinent state of the art here. Since the early 1990s, a number of research groups from the human factors community, in



(a) The automata.

input sequence	output sequence		mode confusion observed?
	technical system	mental model	
a a a a	x y z x	x y z x	no
a b a a	x y z z	x y z x	yes

(b) Reactions of the automata to different input sequences.

Fig. 1. Example with potential for mode confusion.

particular the aviation psychology community, work on mode confusions in shared-control systems. Recently, people from the computer science community, especially the formal methods community, also became interested. There are some promising results with respect to tool supported detection of mode confusion problems (see below). But it remains surprisingly unclear what a mode confusion actually is.

Definitions of Mode and Mode Confusion. While some relevant publications give no [6–8] or only an implicit definition [9,10] of the notions "mode" and "mode confusion", there are others that present an explicit informal def-

inition [11–14].

Doherty [15] presents a formal framework for interactive systems and also gives an informal definition of “mode error”.

Thimbleby [16] develops his “mode” definition over some stages from a generic and informal one (“a mode is a variable information in the computer system affecting the meaning of what the user sees and does”, [16, p. 228]) to a formal one. Doing so, he focuses his scope to the pure two-agent interaction between the human and the machine. He does not consider the physical environment. The latter is a third agent relevant in shared-control systems. As a result, he defines a mode to be a “mathematical function mapping commands to their meanings within the system” [16, p. 255]. Thimbleby does not deal with the mode confusion problem. He therefore does not provide a rigorous definition of the notion “mode confusion”.

Wright and colleagues give explicit but example driven definitions of the notions “error of omission” and “error of commission” by using the language CSP to specify user tasks [17].

Modelling and Tool Support. Interestingly, the way of modelling often seems to be influenced significantly by the tool that is meant to perform the final analysis.

Degani and Heymann use the language StateCharts to model separately the technical system and the user’s mental model of its behaviour [18]. Then they compose both models and search for certain composite states (so-called “blocking”, “error”, and “augmenting” states) which indicate mode confusions.

Butler *et al.* use the theorem prover PVS to examine the flight guidance system of a civil aircraft for mode confusion situations [7]. They do not consider the mental model of the pilot as an independent entity in their analysis.

Campos and Harrison [8] use the model checker SMV. They specify the system as a state transition system. They specify selected properties of the mental model as assertions in temporal logic.

Leveson and her group specify the black-box behaviour of the system in the language SpecTRM-RL that is intended to be both well readable by humans and processable by computers [14,19,20]. In [14], they give a categorisation of different kinds of modes and a classification of mode confusion situations.

Thimbleby (see above) uses the so-called PIE modelling approach [16] that describes human-machine interaction by specifying a sequence of user commands, the **P**rogram. Such a program is interpreted by the technical system by an **I**nterpretation function and causes some **E**ffect. PIE models are also readable by humans and processable by computers.

Sage and Johnson [21] describe a rapid prototyping approach for an air traffic control system. They are able to verify safety properties based on a system specification in the language LOTOS. They claim that their method can support the operator directed design process proposed in [22] (see below). Nonetheless, they do not specify the mental model of the user.

Rushby and his colleagues employ the model-checking tool Murϕ [23,9,6]. Technical system and mental model are coded together as a single set of so-called Murϕ rules. In each step, all rules are "fired" of which the condition is true; i. e. some manipulation of global state variables is performed. Fur-

thermore, a set of invariants is checked. The mode confusion situations are detected with these invariants.

Lüttgen and Carreño examine the three state-exploration tools Murϕ, SMV, and Spin with respect to their suitability in the search for mode confusion potential [24]. They find that each tool has its advantages but also its drawbacks: Spin supports the designer to find the sources of mode confusion situations by the animation of diagnostic information. SMV bears the advantages that it integrates temporal logics. And Murϕ provides the best specification language.

Buth [13] and Lankenau [25] clearly separate the technical system and the user's mental model in their CSP specification of the well-known MD-88-"kill-the-capture" scenario and in a service-robotics example, respectively. The support of this clear separation is one reason why Buth's comparison between the tool Murϕ and the CSP tool FDR favours the latter [13, pages 209-211].

Meanwhile, also Rushby [26] acknowledges this need to separate both entities. For mode confusion detection, he affirms the advantages of model-checking tools for process algebras such as FDR over tools such as Murϕ. A conformance relation between two descriptions has to be checked. The concepts of refinement and abstraction are required for this. They are provided directly by FDR.

Case Studies. Almost all publications refer to the aviation domain when examining a case study: an MD-88 [2,11,14,23,13], an Airbus A320 [6,10], or a Boeing 737 [9]. For a non-aviation case study, refer to Thimbleby's running (pedagogical) example, a calculator [16].

Recommendations. The literature has several recommendations for avoiding mode confusions.

Reason [27] is concerned with human error in general. He recommends to minimize the affordances for error. He takes up the design principles of Norman's "Psychology of Everyday Things" [28]: use both knowledge in the world and in the head in order to promote a good conceptual model. Simplify the structure of tasks. Make both the execution and the evaluation sides of an action visible. Exploit natural mappings. Exploit the power of constraints, both natural and artificial. Design for error; make it easy to reverse operations and hard to carry out non-reversible ones; exploit forcing functions. When all else fails, standardize.

Sarter and Woods [11] propose several measures against mode confusions: reduce the number and complexity of modes (if possible). Focus training on knowledge activation in context. Train skill at controlling attention. Provide better indications of what mode the system is in and how future conditions may produce changes. Maybe provide displays for the history of interaction. Also use nonvisual, e. g., aural or kinesthetic, channels to reduce load on the visual channel. Use forcing functions to guide the user, if the system has enough overall context to do it sensibly.

Butler et al. [7] propose to create a clear, executable formal model of the automation and use it to drive a (flight-deck) mockup for (pilot) training. It can be augmented with an additional display, for training only, that directly exposes the internal structure of the automation and its internal changes.

Leveson et al.[14,29] have identified six categories of system design features that can contribute to mode confusion errors (and thus should be avoided):

ambiguous interfaces, inconsistent system behaviour, indirect mode transitions, lack of appropriate feedback, operator authority limits, and unintended side effects.

Degani and Heymann [30] propose to check formally whether all necessary information is presented to the user in order to avoid mode confusion. This requires a formal model of both the machine and of user's mental model. They also propose an algorithm to generate automatically the interface to the machine and the corresponding user manual information [18,31].

Rushby proposes a procedure to develop automated systems which pays attention to the mode confusion problem [5]. The main part of his method is the integration and iteration of a model-checking based consistency check and the mental model reduction process introduced by [32,6].

Vakil and Hansman, Jr. [22] recommend three approaches to reduce mode confusion potential in modern aircraft: pilot training, enhanced feedback via an improved interface, and, most substantial, a new design process (ODP, for operator directed design process) for future aircraft developments. ODP aims at reducing the complexity of the pilot's task, which may involve a reduction of functionality.

Critique. Hourizi and Johnson [10,33] criticize that automation surprises are not only due to mode error, but also due to a "task knowledge gap". It is more than a perceptual slip. The underlying problems are a (mode) confirmation bias and selective (mode-confirming) perception of the human user.

3 A Rigorous Definition of Mode Confusion in Safety-Critical Systems

Interestingly, none of the work surveyed above defines the notions of "mode" and "mode confusion" rigorously. We therefore propose such definitions. They will help to tackle mode confusion problems.

We will present our definitions in several steps. We start with a brief introduction to a suitable notion of formal refinement. It will be the base of our definition. We then introduce the notions that are part of our definition: the behaviour of the technical system, the mental model of the behaviour of the technical system, the user's senses, and the safety-relevant abstractions of all of these. The actual rigorous definitions conclude this section.

3.1 Brief Introduction to Refinement

We use a kind of specification/implementation relation in the following. Such relations can be modelled rigorously by the concept of *refinement*. There exist a number of formalisms to express refinement relations. We use CSP [34] as specification language and the refinement semantics proposed by Roscoe [35]. One reason for using CSP is that there is good tool support for performing automated refinement checks with the tool FDR [35]. This section shall clarify the terminology for readers who are not familiar with the concepts.

In CSP, one describes the externally visible behaviour of a system by a so-called process. Processes are defined over events. CSP offers a set of operators. One can use them to specify processes.

In CSP, the meaning of a process P can be described by the set $traces(P)$ of the event sequences it can perform. Since we must pay attention to what can be done as well as to what can be *not* done, the traces model is not sufficient in our domain. CSP offers the enhanced *failures model* for this case.

Definition 1 (Failure) *A* failure *of a process P is a pair (s, X) of a trace s ($s \in traces(P)$) and a so-called* refusal *set X of events that may be blocked by P after the execution of s.*

If an output event o is in the refusal set X of P, and if there also exists a continuation trace s' which performs o, then process P may decide internally and non-deterministically whether o will be performed or not.

Definition 2 (Failures Refinement) *P refines S in the failures model, written $S \sqsubseteq_F P$, iff $traces(P) \subseteq traces(S)$ and also $failures(P) \subseteq failures(S)$.*

This means that P can neither accept an event nor refuse one unless S does; S can do at least every trace which P can do, and additionally P will refuse not more than S does. Failures refinement allows to distinguish between external and internal choice in processes, i.e. whether there is non-determinism. As this aspect is relevant for our application area, we use failures refinement as the appropriate kind of refinement relation.

3.2 The Behaviour of the Technical System

We must use a *black-box view* of a running technical system for the definition. This is because the user of such a system has a strict black-box view of it and because we want to solve the user's problems. As a consequence, we can observe

(only) the environment of the technical system, not its internal workings. When something relevant happens in the environment, we call this an *event*. The user can cause such events, too.

There must be a general consensus on what the events are. This is a basic assumption of our approach about the domain where we apply it. In the safety-critical systems domain, this assumption is true. For example, there is no argument between pilots and cockpit designers about whether the lighting of a sign or the pressing of a button is relevant for flying a plane.

The technical system has been constructed according to some *requirements* document REQ. We can describe REQ entirely in terms of observable events, by referring to the *history* of events until the current point of time. For this description, no reference to an internal state is necessary. Usually, several histories of events are equivalent with respect to what should happen in the future. Such equivalences can greatly simplify the description of the behaviour required, since we might need to state only a few things about the history in order to characterise the situation.

During any run of the technical system, it is in one specific state at any point of time. The (possibly infinite) state transition system specified by REQ defines the admissible system runs.

3.3 The Mental Model of the Behaviour of the Technical System

We call the user's mental model of the behaviour of the technical system REQ^{M}. Ideally, REQ^{M} should be "the same" as REQ. During any run of the technical system, REQ^{M} is also in one specific state at any point of time. You

may think of the behaviour of REQ^{M} as a "parallel universe" in the user's mind. Ideally, it is tightly coupled to reality. Each time an event happens and the technical system changes into another state, the user keeps track of what has happened and adjusts his/her expectations about future events accordingly.

Our approach is based on the motto "*the user must not be surprised*". This is an important design goal for shared-control systems. We must make sure that the reality does not exhibit any behaviour which cannot occur according to the mental model of its behaviour. Additionally, the user must not be surprised because something expected does *not* happen. When the mental model prescribes some behaviour as necessary, reality must not refuse to perform it. For example after dialling a number, a phone must either produce an alert tone or a busy tone, and it must never ring itself.

The rule of non-surprise means that the relationship between the reality's behaviour and the user's mental model of its behaviour must be a *relationship of implementation to specification*. The reality should do exactly what the mental model prescribes, no less and no more. In case that the user does not know what to expect, but knows that he/she does not know, then the reality is free to take any of the choices. A common example is that the user does not know the exact point of time at which the technical system will react to an event, within some limits.

We can describe such an implementation/specification relationship formally by a refinement relation. In CSP, *failures refinement* is precisely the relation described above.

The user does not always notice when his/her mental model of the behaviour REQ^{M} is not the same as the behaviour of the reality REQ. This is because the user's mind does not take part in *any* event in the environment. The user perceives the reality through his/her senses only.

The user's senses SENSE translate from the set of events in the environment to a set of events in the user's mind. SENSE is not perfect. Therefore we must distinguish these two sets. For example, the user might not hear a signal tone in the phone due to loud surrounding noise. Or the user might not listen to all of a lengthy announcement text, or he/she might not understand the language of the announcement. At the very least, there is always a larger-than-zero delay between any environment event and the respective mental event. In all these cases, what happens in reality, as described by REQ, is different from what happens according to the user's perception of it, as described by SENSE(REQ).

The user is surprised only if the *perceived* reality does not behave the same as his/her expectations. This is why the user does not always notice a difference between the actual reality REQ and the "parallel universe" REQ^{M} in his/her mind.

We cannot compare the perceived reality SENSE(REQ) to the mental model of the reality REQ^{M} directly. They are defined over different sets of events (mental/environment). We need a translation.

The user has a mental model of his/her own senses SENSE^{M}. SENSE^{M} trans-

lates the behaviour of the mental model of the technical system $\mathrm{REQ}^{\mathrm{M}}$ into events in the user's mind. It does this in the same fashion as SENSE does it for REQ.

The user's knowledge about the restrictions and imprecisions of his/her own senses is also part of $\mathrm{SENSE}^{\mathrm{M}}$. Ideally, the user should know about them precisely, such that $\mathrm{SENSE}^{\mathrm{M}}$ matches SENSE exactly. The user is not surprised if the process SENSE(REQ) is a failures refinement of the process $\mathrm{SENSE}^{\mathrm{M}}(\mathrm{REQ}^{\mathrm{M}})$. [1]

3.5 The Abstractions

We restrict our definition of mode confusion to safety-critical systems. This is because traditionally the safety-critical systems community has perceived mode confusions as a problem. As a consequence, we need to abstract to the *safety-relevant* aspects of the technical system.

When the user concentrates on safety, he/she performs an on-the-fly simplification of his/her mental model $\mathrm{REQ}^{\mathrm{M}}$ towards the safety-relevant part $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$. This helps him/her to analyse the current problem with the limited mental capacity. Psychological studies show that users always adapt their current mental model of the technical system according to the specific task they carry out [4]. The "initialisation" of this adaptation process is the static part of their mental model, the *conceptual model*. This model represents the

[1] In [36], we used the name MMOD for $\mathrm{SENSE}^{\mathrm{M}}(\mathrm{REQ}^{\mathrm{M}})$. We did not define $\mathrm{SENSE}^{\mathrm{M}}$ and $\mathrm{REQ}^{\mathrm{M}}$ separately. We now make a distinction between these two different kinds of mental model for clarity.

user's knowledge about the system and is stored in the long term memory.

Analogously to the abstraction performed by the user, we perform a simplification of the requirements document REQ to the safety-relevant part of it $\mathrm{REQ}_{\mathrm{SAFE}}$. $\mathrm{REQ}_{\mathrm{SAFE}}$ can be either an explicit, separate chapter of REQ, or we can express it implicitly by specifying an abstraction function, i. e., by describing which aspects of REQ are safety-relevant. We abstract REQ out of three reasons: $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$ is defined over a set of abstracted events, and it can be compared to another description only if it is defined over the same abstracted set; we would like to establish the correctness of the safety-relevant part without having to investigate the correctness of the entire mental model $\mathrm{REQ}^{\mathrm{M}}$; and our model-checking tool support demands that the descriptions are restricted to certain complexity limits.

We express the abstraction functions mathematically in CSP by functions over processes. Mostly, such an abstraction function maps an entire set of events onto a single abstracted event. Other transformations are hiding (or concealment [34]) and renaming. But the formalism also allows for arbitrary transformations of behaviours; a simple example being a certain event sequence pattern mapped onto a new abstract event. We use the abstraction functions $\mathcal{A}_R$ for REQ and $\mathcal{A}_M$ for $\mathrm{REQ}^{\mathrm{M}}$, respectively.

The relation SENSE must be abstracted in an analogous way to $\mathrm{SENSE}_{\mathrm{SAFE}}$. They are relations from processes over environment events to processes over mental events. It should have become clear by now that $\mathrm{SENSE}_{\mathrm{SAFE}}$ needs to be rather true, i. e., a bijection which does no more than some renaming of events. If $\mathrm{SENSE}_{\mathrm{SAFE}}$ is "lossy", we are already bound to experience mode confusion problems. $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}$ accordingly is the user's mental model of

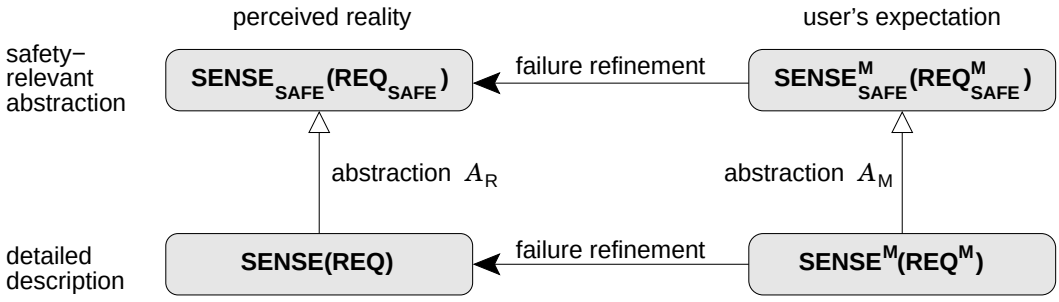


Fig. 2. Relationships between the different refinement relations.

$\mathrm{SENSE}_{\mathrm{SAFE}}$.

Figure 2 shows the relationships among the different descriptions. In order not to surprise the user with respect to safety, there must be a failures refinement relation on the abstract level between $\mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$ and $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}})$.

3.6 The Definitions

In the following, let $\mathrm{REQ}_{\mathrm{SAFE}}$ be a black-box requirements specification, abstracted to the safety-relevant aspects, let $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$ be a mental model of the behaviour of $\mathrm{REQ}_{\mathrm{SAFE}}$, and let $\mathrm{SENSE}_{\mathrm{SAFE}}$ and $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}$ be relations from processes over environment events to processes over mental events representing the user's senses and the mental model of them, respectively.

The definition of mode needs a precise definition of a potential future behaviour. We take it directly from the failures model of CSP (Def. 1).

Definition 3 (Potential future behaviour) *A* potential future behaviour *is a set of failures.*

A state is a potential future behaviour. We can distinguish two states of a system only if the system may behave differently in the future. This is because of the black-box view.

Definition 4 (Automation surprise) An *automation surprise* between $\mathrm{SENSE}(\mathrm{REQ})$ and $\mathrm{SENSE}^{\mathrm{M}}(\mathrm{REQ}^{\mathrm{M}})$ occurs if and only if $\mathrm{SENSE}(\mathrm{REQ})$ is not a failures refinement of $\mathrm{SENSE}^{\mathrm{M}}(\mathrm{REQ}^{\mathrm{M}})$, i.e., iff $\mathrm{SENSE}^{\mathrm{M}}(\mathrm{REQ}^{\mathrm{M}}) \quad \not\sqsubseteq_F \quad \mathrm{SENSE}(\mathrm{REQ})$.

The user is surprised if any detail of the technical system contradicts to his/her expectations.

A mode is just a state. But we reserve the word for the "states" of abstracted descriptions, i. e., of $\mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$ and of $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}})$. We can distinguish two modes only if the system may behave differently in the future with respect to safety.

Definition 5 (Mode) *A* mode *of* $\mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$ *is a potential future behaviour. And, a* mode *of* $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}})$ *is a potential future behaviour.*

We now finally can present our central definition for mode confusion:

Definition 6 (Mode confusion) A *mode confusion* between $\mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$ and $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}})$ occurs if and only if $\mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$ is not a failures refinement of $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}})$, i.e., iff $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}) \quad \not\sqsubseteq_F \quad \mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$.

A (safety-critical) mode confusion is an automation surprise, but only if it is

safety-relevant.

Every time a user's $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$ changes, one must decide anew whether a mode confusion occurs. Our definition of mode confusion is based on the (rather strong) assumption that $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$ is stable over time. The user generates $\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}$ on-the-fly from $\mathrm{REQ}^{\mathrm{M}}$ and must re-generate it later when he/she needs it again. This re-generation might lead to a different result. In particular, the re-generation requires the user's recollection of the current mode. A user's lapse [27] here can result in a mode confusion. This happens when the user selects a mode as initial mode which does not match the reality's current mode.

4 Classification of Mode Confusions

We classify mode confusions into four classes. The classification follows directly from the above definition of mode confusion. Each part where something can go wrong leads to a class.

(1) Mode confusions which arise from *incorrect knowledge* of the human about the technical system and its environment.

If $\mathrm{REQ}^{\mathrm{M}}$ does not match REQ, then the failures refinement relation can break.

(2) Mode confusions which arise from the *incorrect abstraction* of the user's knowledge to the safety-relevant aspects of it.

If the mental abstraction function $\mathcal{A}_M$ does not match the abstraction function for the technical system $\mathcal{A}_R$, then the failures refinement relation can break (compare Figure 2 above).

(3) Mode confusions which arise from an *incorrect observation* of the technical system or its environment. This may have *physical* or *psychological* reasons.

The sense organs may be physically imperfect; for example, eyes which cannot see behind the back. Or an event is sensed physically, but is not recognised consciously; for example because the user is distracted, or because the user currently is flooded with too many events. ("Heard, but not listened to.") If SENSE does not match SENSE^{M}, the failures refinement relation can break. We could call this class also the mode confusions which arise from incorrect knowledge of the human about his or her own senses. The confusion disappears when the human knows about the senses' limitations.

(4) Mode confusions which arise from an *incorrect processing* of the abstracted mental model by the user. There can be a memory lapse or a "rule-based" mistake [27], i. e., a mode transition that is not part of the correctly interpreted model.

An "execution failure" can spoil an otherwise perfect abstracted mental model. The model's semantics depends on the executing "machine".

In contrast to previous classifications of mode confusions, this classification is *by cause* and not phenomenological, as, e.g., the one by Leveson [14].

5 Recommendations for Avoiding Mode Confusions

The above causes of mode confusions lead directly to recommendations for avoiding them.

R1: Make the technical system deterministic. Non-determinism increases the user's effort for processing the mental model. The user must simultaneously track several alternative paths in the model. This can quickly exceed the user's mental processing capabilities and lead to incorrect processing. Therefore, the requirements of the technical system should allow as little non-deterministic internal choices as possible. To eliminate a non-deterministic internal choice, we must change the system requirements. We must add an environment event controlled by the machine and observed by the user which indicates the software's choice.

This recommendation generalises and justifies the recommendation by others to eliminate "hidden mode changes" [29,12].

R2: Check that the user can physically observe all safety-relevant events. This avoids incomplete observation. To also avoid incorrect observation, we must check that the user's senses are sufficiently precise to ensure an accurate translation of these environment events to mental events. To prevent observation problems, we can apply the same measure as used against non-deterministic internal choices: we add an environment event controlled by the machine which indicates the corresponding software input event.

Improving the user's knowledge about his/her own senses has little potential for avoiding mode confusion problems. If the user knows that some things may happen, but he/she cannot perceive them, then they are non-deterministic choices to the user's mind. Again, the user will have difficulties with the complexity of tracking alternative outcomes.

R3: Check that the user can psychologically observe all safety-relevant events. This avoids an incorrect observation because of a psychological reason. We must check that observed safety-relevant environ-

ment events become conscious reliably. The knowledge-based approach of Hourizi and Johnson [10,33] can be a starting point here.

R4: Document the requirements explicitly and rigorously. This helps to establish a correct knowledge of the user about the technical system and its environment. It enables us to produce user training material, such as a manual, which is complete with respect to functionality.

R5: Document the safety-relevant part of the requirements separately, or mark it clearly. This helps to produce training material which aids the user to concentrate on safety-relevant aspects. Such training material, in turn, helps the user to abstract correctly to the safety-relevant parts. It makes explicit the safety-relevance abstraction function for the machine, $\mathcal{A}_R$.

We must also *minimize the affordances for human error* in general. We already cited the respective recommendations of Reason [27] on page 10 above. Reason distinguishes three basic types of human error: skill-based slips and lapses, rule-based mistakes, and knowledge-based mistakes. Slips appear as incorrect observation for psychological reasons in our classification, and knowledge-based mistakes appear as incorrect knowledge. Lapses and rule-based mistakes cause incorrect processing.

6 Checking Our Definition Against Other's Notions of Mode Confusion

We now check whether our definitions of mode and of mode confusion indeed cover the informal notions in the literature. Appendix A contains a detailed comparison of eight informal notions to our definition. There, we investigate

the work of Thimbleby [16], Doherty [15], Sarter & Woods [11], Leveson et al. [14], Rushby [9,26], Degani & Heymann et al.[12,31,18], Buth [13], and Hourizi & Johnson [10,33]. Here, we discuss our findings in Appendix A.

We conclude that all authors, including us, agree about what a mode is. Only Degani & Heymann et al. disagree in one sub-topic. They have a white-box view of the system instead of the usual black-box view.

The abstraction from states to modes is discussed by only a few authors. Most just implicitly assume that it has been done. Some use abstraction for a different purpose. They use it to reduce the size of the state space such that model checking becomes feasible. Doherty makes a case to have a user-relevant abstraction. We specialize this to a safety-relevant abstraction.

All authors who use a model-checking tool require that the two models must be in some equivalence relation to avoid mode confusion. Here, we disagree. We require a (failures) refinement relation in one direction only. Equivalence would mean refinement in both directions. Our position gets some support from Sarter & Woods: a problem arises only if the user does something wrong.

We claim that equivalence is stronger than necessary. If the mental model is in a specification/implementation relation, i. e., a refinement relation, with the technical system, then no automation surprise will arise. We agree that a non-deterministic mental model can cause a problem indirectly. Non-determinism can quickly exceed the user's mental capacity, leading to incorrect processing of the model. But this does not happen necessarily. Therefore we prefer to distinguish an outright wrong mental model from an execution failure on a correct model.

We suspect that the general insistence on an equivalence relation roots in the tools used. There are many model checking tools, but only one can check for failures refinement. This is FDR, the tool we use. Without FDR, one needs to check for a more than sufficient condition if one wants tool support at all.

It is generally accepted that two models must be compared. Nevertheless, all model checking tools except FDR require to encode the two models into a single composite automaton. Buth's idea to use FDR makes the comparison much more natural.

Only few authors consider "incorrect observation" explicitly. Degani & Heymann et al. do it, and Hourizi & Johnson, too. We can probably safely assume that all other authors would agree that this can happen, even if they did not include it in their particular approach. Buth, for example, hides a non-perceivable event manually. Our rigour made it obvious that one needs an explicit translation from environment events to mental events.

Our approach does not allow to have more than one human controller or more than one automated controller, as Leveson et al.'s approach does. But both views can be translated into each other.

Our approach is specific to safety-critical systems. Other, earlier literature discusses mode confusion in (moded) text editors. We can adapt our definition by abstracting to other than safety-relevant aspects. We do this elsewhere [37,38], for the telephony domain.

All tool-supported approaches use the term "mental model" in the restricted sense of "mental model *of the behaviour* of the technical system". They furthermore assume that an explicit, useful description of such a mental model

can be extracted. We follow them in our attempt to clarify the definitions. This restriction loses the wider general meaning of mental model. But it also enables us to propose definitions for mode and mode confusion with mathematical rigour, and it enables us to exploit the analytical power of the tools.

All tool-supported approaches check for *any* kind of automation surprise. This includes our approach. One might argue that automation surprises exist that are no mode confusions. We are convinced firmly that we need a black-box view of the technical system. But this implies that we can distinguish two modes *only* by their potential future behaviour. If two behaviours are different, that is, if there is a surprise, then there must be two different modes. Our solution therefore is to have a suitable abstraction from states to modes. An automation surprise is no mode confusion if it is abstracted away.

7 Application: Mode Confusion Analysis for an Automated Wheelchair

We demonstrate the usefulness of our definition and of our recommendations by an application in the service robotics domain. We analyze the cooperative obstacle avoidance behaviour of a wheelchair robot. We specify its behavior formally and then we analyze it with an automated tool. This reveals several mode confusion problems. We then resolve these problems.

Fig. 3. The autonomous wheelchair Rolland that was model-checked for mode confusion problems. Photo: Rolf Müller

7.1 The Bremen Autonomous Wheelchair "Rolland"

The Bremen Autonomous Wheelchair "Rolland" is a shared-control service robot, that realizes intelligent and safe transport for handicapped and elderly people. The vehicle is a commercial off-the-shelf power wheelchair Meyra Genius 1.522. It has been equipped with a control PC, a ring of sonar proximity sensors, and a laser range finder (Fig. 3).

Rolland is jointly controlled by its user and by a software module, in contrast to other service robots. Depending on the active operation mode, either the user or the automation is in charge of driving the wheelchair. Conflict situations, often caused by mode confusions, arise if the commands issued by the two control instances contradict each other.

7.1.1 System Architecture

The user controls the commercial version (no control PC, no sensors) of the wheelchair with a joystick. The command set via the joystick determines the speed and the steering angle of the wheelchair.

The *safety module* [39] wiretaps the control line from the joystick to the motor. Only those commands that won't do any harm to the wheelchair and its user are passed unchanged. If there is an obstacle dangerously close to the wheelchair, the safety module performs an emergency brake by setting the target speed to zero. The notion "dangerously" refers to a situation in which there is an object in the surroundings of the wheelchair that would be hit, if the vehicle was not decelerated to a standstill immediately. Thus, this fundamental module ensures safe travelling in that it guarantees that the wheelchair will never actively collide with an obstacle.

Higher-level skills provide additional functionality above the safety module. Obstacle avoidance (i. e., smoothly detouring around objects in the path of the wheelchair), assistance for passing the doorway, behaviour-based travelling (wall following, turning on the spot, etc.) and others. These modules have been combined to the *driving assistant* [40]. It provides the driver with various levels of support for speed control and for steering.

7.1.2 Obstacle Avoidance Skill

The obstacle avoidance skill must satisfy two requirements. Firstly, the automation must support the handicapped user when braking or detouring around objects. The goal is a smooth and comfortable driving behaviour.

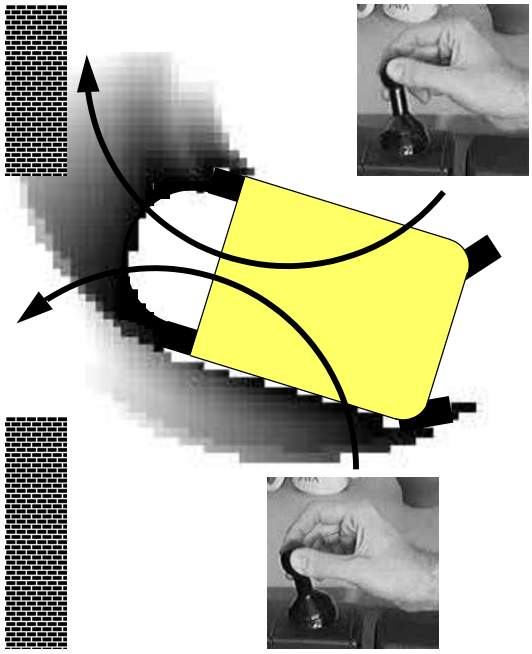


Fig. 4. Deciding on which side the user wants the obstacle to be passed.

Secondly, the user must not be surprised. Whatever the automation decides to do, it has to be consistent with the user's expectation.

This intelligent shared-control behaviour is realized by projecting the anticipated path of the wheelchair into a local obstacle occupancy grid map. Figure 4 shows a situation in which the wheelchair is supposed to pass through a doorway. The right doorpost is a relevant obstacle since it is on the current path of the vehicle. The distance before collision is visualised for positions on this path by the grey-shaded area: the darker the sooner some part of the wheelchair will reach the corresponding position. The joystick command shown on the photo in the upper right corner of the figure indicates a narrow right curve. Since the corresponding projected path (upper arrow) points to the right of the doorpost, this command is interpreted as "do not pass through the doorway". The lower photo shows a joystick command indicating a left curve. Since the corresponding projected path (lower arrow) points to the left of the doorpost

(i. e. through the doorway), this command is interpreted as "pass through the doorway".

The algorithm chooses the speed and steering angle depending on the side on which the projected path, indicated with the joystick, passes the obstacle. If the driver directly steers toward an obstacle, the algorithm infers that he or she wants to approach the object. Then it does not alter the steering angle. As a result, obstacles are smoothly detoured if desired, but they can be directly approached if need be. If the automation realizes that the projected path of the vehicle happens to be free after an avoidance manoeuvre, it again accelerates up to the speed indicated by the user via the joystick.

The transition to the obstacle avoidance mode is an "indirect" one [14]. The mode is not invoked by the user on purpose. Thus, the driver probably does not adapt to the new situation after an obstacle has been detoured, because he or she did not notice that the operation mode changed from operator-control to obstacle avoidance. It is very likely that the user would not react immediately after the avoidance manoeuvre and steer back to the original path. Instead, he or she would probably not change the joystick command. The driver would be surprised that the wheelchair follows a wrong track after the obstacle.

An additional feature of the obstacle avoidance algorithm fixes this obvious mode confusion problem. It steers back to the heading of the original path after the obstacle has been passed. If the user does not adapt to the new situation, i. e., he or she does not change the joystick position after a detouring manoeuvre, the algorithm interprets the command in the frame of reference that was current when the manoeuvre began.

The algorithm therefore is able to navigate through a corridor full of people

or static obstacles by simply pointing forward with the joystick. If there is an object that has to be detoured, the user keeps the joystick in an unchanged position and thereby enables the obstacle avoidance algorithm to steer back to the orientation of the original path.

Please note that we extend the narrow safety notion introduced here during our case study: Any behaviour of the wheelchair that contributes to its obstacle avoidance skill is considered to be safety-relevant.

7.2 Obtaining Specifications of the Behaviour

We obtained an explicit specification of the motion behaviour of the wheelchair robot, and we obtained an example of an explicit mental model of the wheelchair's motion behaviour, in order to demonstrate our mode confusion analysis approach.

7.2.1 Methodology

Our goal is to prove that our mode confusion analysis can be applied successfully to a problem of practical size. This requires that two explicit specifications are available for comparison. It is not our goal to explore suitable ways for the extraction of a mental model of behaviour. This is outside of our research focus and of our expertise.

A basic assumption of our work is that an explicit mental model of the safety-relevant aspects of the wheelchair's behaviour can be made available. Mental models have been extracted from training material, from user interviews, and by user observation (see Sect. 2.1). There is dispute about how reliable user

observations are, in particular (see, e.g., [4]). Even in a user interview, one has to take into account that the user may actually believe in one thing, but act in a different manner [3]. Solutions to such difficulties are outside the focus of our work.

We selected a simple way of obtaining an example of a mental model of behaviour. It is sufficient to prove the feasibility of our approach under the above assumption. We performed a naive user interview. The resulting explicit description probably does not match exactly the actual model of the user. Furthermore, the user interviewed is an expert user. This prevents us from detecting mode confusion problems that are specific to novice users. Nevertheless, we believe that the resulting explicit model has sufficient resemblance to an actual mental model to prove the feasibility of our approach. In addition, we think that the insights from the example analysis are still of value to a designer of an automated wheelchair, despite some limitations.

7.2.2 The Events

The relevant events are obvious for the wheelchair. The basic assumption from Sect. 3.2 above therefore is satisfied that there is a general consensus about the events. There is a number of variables of the technical system visible to the user in a black-box view. These are the position of the joystick, the actual status of the wheelchair motors, the orientation of the wheelchair in the initial inertial system, the locations of obstacles, and the current command to the wheelchair motorics. We therefore specified events in CSP that denote a change in one of the variables.

The safety-relevant abstractions of the events required a little more work,

but it was straightforward. First, we documented explicitly for the detailed data types that describe measured values and motor commands, which properties are safety-relevant. We then replaced the detailed data types by suitably abstracted ones. The latter only have two to three or sometimes four distinct values instead of some integer ranges. For example, we abstracted an integer-valued speed command range between −42 cm/s and 84 cm/s to the three values `standStill`, `slowSpeed`, and `fastSpeed`. They cover all distinct safety-relevant cases. We do not even distinguish between forward and backward driving, since the setting turned out to be symmetrical with this respect.

The most difficult abstraction was that of the virtual map of the obstacle situation. We kept only the closest obstacle on the current path of the wheelchair. An object in the surrounding is a relevant obstacle if driving further on the current path would cause some part of the wheelchair to collide with the object. We describe the position of the obstacle relative to the wheelchair by a potential wheelchair path and a distance. The path is defined by the steering angle that would be necessary for a collision of the centre of the wheelchair's front axle with the obstacle. The distance is the travel distance before impact. Since we are not interested in the distance as such, we abstract it to the corresponding criticality with respect to the current wheelchair speed: if the obstacle is far away, the required action is less demanding than it is if the obstacle is close.

7.2.3 Obtaining an Example Mental Model of the Behaviour

We "interviewed" a user who has built a mental model of the wheelchair robot through extensive use: the second author of this article. Even though he has

seen the source code of the software, he definitely does not use this knowledge while driving. Instead, he has built his own, intuitive, black-box mental model. This model is much easier to use for quick decisions. It also turned out that it is structured differently than the technical system. In addition, the user interviewed can express himself in CSP directly. This saved us from conducting a standard user interview for this research.

However, we conducted several interviews with different kinds of users for other purposes. We tried to optimize the different skills of the wheelchair such as obstacle avoidance, turning on the spot, etc. For this, we also experimented with wheelchair novices such as visitors and students.

7.2.4 Obtaining the Requirements Specification of the Behaviour

We extracted the CSP requirements specification of the behaviour of the wheelchair robot through "reverse engineering" from the source code. Unfortunately, no requirements document existed before this. (Of course, we did this only *after* we specified the mental model, in order not to spoil the latter by a fresh and close impression of the code.) The CSP specification is close to the source code. We restricted it to those parts related to motion. The rather complex sensor software is included at a high level of abstraction only.

The technical system is split into three parts: the input devices, the software, and the output devices. This separation is not present in the mental model. The mental model sees the entire technical system as a single black box.

The driver software of the sonar system provides "virtual sensors" [40]. They allow the other software to inspect a virtual map of the obstacle situation.

This design structure of Rolland helped a lot for the specification of the requirements of the input devices. The mapping of physical obstacle locations to the software's input variables became a nearly trivial mapping because of this.

The safety-relevant abstraction of the behaviour is rather similar to the detailed specification. Mostly, the parameters of the detailed events were replaced by the simpler parameters of the abstracted events. All motion-related behaviour is potentially safety-relevant. Accordingly, the mental model of the safety-relevant part of the behaviour is rather similar to its detailed version, too. But with other applications, more simplifications might be possible and necessary.

7.2.5 Overview of CSP Specifications

There are the four specifications in CSP. We have versions for the technical system and for the mental model, both subdivided into a detailed version and an abstracted version. They partially share the definitions of events and types as appropriate. Ultimately, we combine them for a refinement check of the detailed descriptions on the one hand, and for an automated refinement check of the safety-relevant abstractions on the other hand. As expected, an automated refinement check at the detailed level is not possible since the state space is way too large.

The user's mental model of the behaviour of the wheelchair's obstacle avoidance module is specified by four major CSP processes: a "halt" process entered whenever the joystick is in neutral position, a "user controlled" process for user controlled driving, an "avoid" process for the obstacle avoidance skill of the

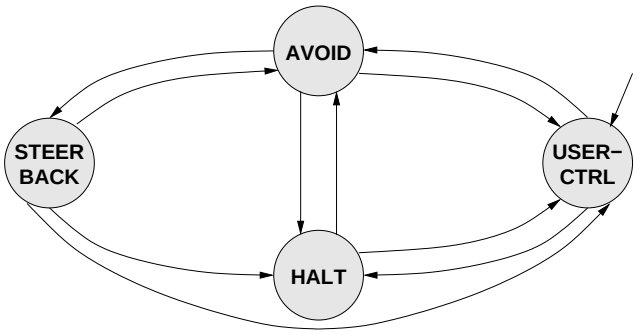


Fig. 5. The user's mental model of the behaviour of the wheelchair represented by four CSP processes.

wheelchair, and a "steer-back" process in which the wheelchair automatically returns to the original driving direction after an obstacle avoidance manoeuvre has been completed. Figure 5 shows the processes and the transitions among them.

The structure of the specification of the wheelchair behaviour requirements is different from that of the mental model. The specification of the wheelchair behaviour requirements `REQ` is the composition of the requirements on the technical system `SYSREQ` and of the requirements on its environment `NATREQ`. `SYSREQ` in turn is a composition of the input device requirements `IN`, the software requirements `SOF`, and the output device requirements `OUT`. `SOF` performs an infinite loop of reading sensors, choosing the appropriate software routine, processing the input, and setting the actuators. In contrast, the specification of the mental model of the behaviour of the wheelchair `REQ_M` is the composition of the mental model of the behaviour of the technical system `SYSREQ_M` and of the mental model of the behaviour of its environment `NATREQ_M`. `SYSREQ_M` performs a loop of perception, calculation, and acting. There are no separate input/output relations, and the detailed structure of `SYSREQ_M` is also quite different from that of `SOF`. For each of the above, the structure of the detailed

and of the abstracted specifications are identical.

The size of both safety-relevant abstractions together is about 1400 lines of commented CSP specification, or about 720 lines of pure CSP specification. The specification is available from us on request.

7.3 Mode Confusion Detected During Modelling

Our rigorous modelling process already revealed a first mode confusion problem:

Mode Confusion Due to Imperfect Vision

The user's vision is more restricted than one would think. We found out when we had to specify the user's senses SENSE explicitly. SENSE does a direct one-to-one mapping of monitored events to mental monitored events mostly (with some delay). But the explicit modelling made it obvious that there is one exception. The user cannot see obstacles behind his back. Of course, this is already a problem when driving backward. But they are likely to obstruct forward paths, too: when driving a curve, the back of the wheelchair swerves out to the side and may hit obstacles which are nearby alongside the wheelchair, but behind the user's head. The wheelchair robot will notice the danger, activate the obstacle avoidance skill, and change the motion into a safe one. The user will not notice the mode change, and he or she will be surprised. This is the reason why driving backward has the exactly same problems as driving forward. (It therefore can be ignored in our abstraction out of symmetry considerations.)

In our classification, the problem found is one arising from an incorrect observation of the environment by the user.

This mode confusion problem can be resolved by adding a feedback light to the user interface. It is on when the system is in the "avoid" or "steer-back" software routine.

7.4 Mode Confusions Detected by Model Checking

An automated analysis detected two mode confusion problems which were new to us. This happened despite the second author knows the wheelchair robot well, even its more obscure properties. Additionally, the automated analysis detected all expected mode confusion problems.

Mode Confusion Due to Fast and Slow Senses

The first new mode confusion problem occurs when the different senses of the user work at different speeds. The relation $\text{SENSE}_{\text{SAFE}}$ translates monitored events to mental monitored events. In a first version, we specified this translation independently for each of the user's senses (vision, tactile, motion-detection). This is realistic, since the organ of equilibrium can take some time before detecting a slow turning, and since the user might not see an obstacle in a complex surrounding immediately. These delays need not be correlated. And the joystick position is felt practically without delay by the user.

The automated model-checking tool FDR [35] detected a violation of the refinement property resulting from this. Figure 6 shows one of the generated counter-examples. Initially, the wheelchair does not move. The

`SENSE_M_SAFE(REQ_M_SAFE)`:

performs:

`<mmJoystickCommand.100.straight>`

then accepts:

`{mcMotorsCommand.fastSpeed.straight}`

`SENSE_SAFE(REQ_SAFE)`:

performs:

`<mmJoystickCommand.100.straight,`

`mmJoystickCommand.100.right>`

This proves that $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}) \not\sqsubseteq_F \mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$, i. e., that the perceived reality is not an implementation of the mental model.

Fig. 6. Counterexample by FDR proving the mode confusion due to fast and slow senses.

user then fully tilts the joystick forward. Shortly after that, the user points the joystick to the right. The user expects to feel the acceleration (`mcMotorsCommand.fastSpeed.straight`) of the forward command after issuing it. Instead, the user has time to issue another command while he or she cannot feel any reaction of the wheelchair. This is a mode confusion. FDR allows to investigate the cause by inspecting the traces of the CSP sub-processes from which `SENSE_SAFE(REQ_SAFE)` is composed. The wheelchair indeed reacts as expected, but the user's senses delay the perception of the reaction. In practise, the situation is not really grave. Human senses are sufficiently fast to clear up any such confusions before driving at 6 km/h becomes difficult. Nevertheless, the tool correctly pointed out that in principle there is a problem.

In our classification, the problem found is one arising from an incorrect observation of the system by the user.

Really resolving this mode confusion is hard. In principle, we can educate the user about the timing problems. But the mental model will become much more non-deterministic. Strongly non-deterministic mental models are hard to handle for the user. They can easily go beyond his or her mental processing capabilities. This leads to mode confusion problems because of incorrect processing. The most viable solution is to keep the timing of the system so slow that we can make the explicit assumption that the user's senses will not delay events noticeably. This was what we did in our case.

The mode confusion problem found will occur in most shared-control systems. It occurs if the user uses different senses, and if these can have different delays for perception. With this respect, a complex visual scene can already count as being perceived by different senses, like the collection of aircraft cockpit panels.

Mode Confusion Due to Wrong Knowledge About the Halting Wheelchair

The second mode confusion problem revealed by the FDR tool is caused by an erroneous simplification of the acquired mental model by the user. Rushby denotes this process of irregularly generalising often used knowledge as *inferential simplification* [5]. The user simplified his model of the behaviour of the "halt" routine (see above) such that the wheelchair was assumed to re-set the steering angle to its initial "straight" position whenever the intended user speed was set to zero. As a consequence, according to this mental model the wheelchair could not change its steering to a value other than `straight` when

`SENSE_M_SAFE(REQ_M_SAFE)`:

performs:

<`mmJoystickCommand.0.right`>

then accepts:

{`mcMotorsCommand.standStill.straight`}

`SENSE_SAFE(REQ_SAFE)`:

performs:

<`mmJoystickCommand.0.right`,

`mcMotorsCommand.standStill.right`>

This proves that $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}) \not\sqsubseteq_F \mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$, i. e., that the perceived reality is not an implementation of the mental model.

Fig. 7. Counterexample by FDR proving the mode confusion in the "halt" routine.

standing still. But the technical system allows this.

The FDR tool reported the refinement violation shown in Fig. 7: the user intends to steer to the right while the wheelchair stands still. Both, the perceived reality as well as the mental model of the reality engage in the corresponding mental monitored event `mmJoystickCommand.0.right`. The technical system correctly maps this joystick command to the corresponding motor command `mcMotorsCommand.standStill.right`. Due to the inferential simplification mentioned above, the mental model refuses to engage in this event, it only accepts `mcMotorsCommand.standStill.straight` here.

Please note that this mode confusion is safety-relevant: if the user's mental model does not allow to change the steering angle during a standstill, the user might lose track of the automation behaviour: consider a situation in which it

is necessary to set the steering angle to its maximum value on either side to avoid a certain object in front of the wheelchair. If your mental model refuses to steer while standing still, you might not be able to set the steering angle soon enough while driving. This is because the curve radius increases when you steer while you are already driving (even at a very low speed level). Therefore, this mode confusion may decide about whether or not it is possible to pass an obstacle, and is thus safety-relevant.

In our classification, this mode confusion results from incorrect knowledge of the user about the system caused by an erroneous inferential simplification.

We resolved this mode confusion by refreshing the second author's knowledge about the "halt" routine: The corrected version of his mental model allows to change the steering angle while the wheelchair is in a standstill. This enhanced version of the mental model is used in the following.

Detecting the Known Mode Confusion

The automated analysis also detected the mode confusion problem which we already found during modelling. We specified the user's senses in non-matching versions for reality and for the mental model of it. The mental model $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE,\,ideal}}$ maps all physical events to mental events perfectly. The reality $\mathrm{SENSE}_{\mathrm{SAFE}}$, however, may replace the visual perception of the closest obstacle by a less critical one.

The model-checking tool generated example traces for a mode confusion situation (Fig. 8 shows one of them). The wheelchair appears to change its motion behaviour without a cause. In the beginning, the user fully tilts the joystick to

SENSE_M_SAFE_IDEAL(REQ_M_SAFE):

performs:

<mmJoystickCommand.100.right,

mcMotorsCommand.fastSpeed.right,

mmObsLocChange.left.nonCriticalDist>

then accepts:

{mcMotorsCommand.fastSpeed.right}

SENSE_SAFE(REQ_SAFE):

performs:

<mmJoystickCommand.100.right,

mcMotorsCommand.fastSpeed.right,

mmObsLocChange.left.nonCriticalDist,

mcMotorsCommand.standStill.right>

This proves that $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE,\ ideal}}(\mathrm{REQ}^{\mathrm{M}}_{\mathrm{SAFE}}) \not\sqsubseteq_F \mathrm{SENSE}_{\mathrm{SAFE}}(\mathrm{REQ}_{\mathrm{SAFE}})$, i. e., that the perceived reality is not an implementation of the mental model.

Fig. 8. Counterexample by FDR proving the mode confusion due to imperfect vision.

forward right. The wheelchair accordingly moves in a right curve at full speed. The user then sees an obstacle on the path. It is a bit to the left of the middle of the path, and still at a non-critical distance. Suddenly, the wheelchair brakes to a stand-still. The cause for the braking action is a second, much closer obstacle on the path which is out of the user's vision. The user is confused. He thinks the wheelchair is in the user control mode, while in reality it is in the avoid mode. He cannot explain this behaviour as long as he assumes that his vision is perfect. Therefore the failures refinement check fails and produces this counter-example.

In our classification, such a mode confusion arises from an incorrect observation of the environment: the "wrong" obstacle is assumed to be the closest obstacle.

We resolved the mode confusion by updating the user's knowledge about his senses: we replaced $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE,\,ideal}}$ by $\mathrm{SENSE}^{\mathrm{M}}_{\mathrm{SAFE}}$. The latter mental model of the senses includes the restricted vision and is identical to the physical senses $\mathrm{SENSE}_{\mathrm{SAFE}}$.

The above mode confusion is very common in manned robotics: the human driver of the robot (here: the wheelchair) is not correctly aware of the obstacle situation in the surrounding of the robot. As a consequence, the user is surprised if the automation intervenes where there seems to be no reason for such an intervention. Or, vice versa, the user cannot track the automation's behaviour if it does *not* intervene while the user expects it should do so.

Proving the Absence of Further Mode Confusions

The automated analysis proved the absence of further mode confusions after we resolved the above problems as described. The model-checking tool investigated all traces of events theoretically possible and thereby conducted a mathematical proof by exhaustive enumeration. Of course, the proof holds only for the mental model of this specific user, and only as long as the actual mental model does not change. The expanded transition graphs to be explored during one of the refinement checks are in the order of 100,000 states and 300,000 transitions.

8 Summary

We present a rigorous way of modelling the user and the machine in a shared-control system. This enables us to propose precise definitions of "mode" and "mode confusion" for safety-critical systems. We then validate these definitions against the informal notions in the literature. Our definition is an improvement with two respects: First, it "sharpens" the relation between the two models involved. We demonstrate that a mathematically weaker relation is sufficient to avoid automation surprises. Instead of an equivalence relation, a specification/implementation relation is sufficient. Second, our definition adds precision to many details. For example, an abstraction step from states to modes is necessary, and one must consider a possibly incorrect observation of the environment by the user.

A new classification of mode confusions by cause is another result of our definitions. It leads to a number of design recommendations for shared-control systems which help to avoid mode confusion problems.

Our approach supports the automated detection of remaining mode confusion problems. A tool to model-check our specific specification/implementation relation exists. We demonstrated our approach practically by applying it to a wheelchair robot. Our rigorous modelling process already revealed a mode confusion problem. Our automated analysis detected two other mode confusion problems, which were new to us. We then could resolve these problems.

The automated detection is obviously restricted to a particular instance of a mental model of behaviour that has been extracted. The success of reducing mode confusion potential in this way therefore depends on the suitable selec-

tion of one or more typical users. It also requires that sufficiently accurate methods are available for extracting a mental model by a user interview or by user observation. As an exception, a mental model derived from training material will be relevant for most users of a safety-critical system without further effort.

If one had applied our recommendations already while building the automated wheelchair, it would have been easier to use. One should have kept an up-to-date requirements document, with a separate section on the safety-relevant behaviour. This could have prevented the above problem with the "halt" process. One also should have made the display show which part of the wheelchair is about to collide with an obstacle. This could have prevented confusion because of the user's insufficient lateral obstacle observation abilities.

Our work lends itself to extension into at least two directions. First, we can apply our recommendations while building a new system. Second, the notions of mode and mode confusion need not be restricted to safety-critical systems, e. g., aviation and robotics. Elsewhere [37,38], we investigate the telephony domain. There, we find that a considerable number of so-called feature interactions are also shared-control mode confusions, and we transfer and adapt the measures against mode confusions from safety-critical systems to telephony.

A Detailed Comparison of Our Definition Against Other's Notions of Mode Confusion

We now provide the detailed comparison for the discussion in Sect. 6. We check whether our definitions of mode and of mode confusion indeed cover the

informal notions in the literature.

Thimbleby [16] defines a mode to be a "mathematical function mapping commands to their meanings within the system". This is consistent with a mode being a potential future behaviour. Thimbleby does not deal with the mode confusion problem.

Doherty [15, pp. 118] finds that any treatment of mode must be based on a user-relevant abstraction, because there will be sequences of input actions which can distinguish between virtually all states. This avoids treating all states as separate modes. This argument supports our choice of a mandatory abstraction function. Doherty defines modes as partitions of the state-space. A mode formally relates a trace of input actions to an outcome. This again is consistent with a mode being a potential future behaviour. Doherty has no explicit notion of mode confusion.

Sarter & Woods [11] have no explicit definition of mode. Concerning mode confusion, they refer to Norman [28] and state that "a human user can commit an erroneous action by executing an intention in a way that is appropriate to one mode when the device is actually in another mode." This definition leaves open what "erroneous" and "inappropriate" mean. If we interpret them as "has an undesired outcome", we get close to our definition. One can argue that Sarter & Woods don't include situations with an unexpected but not undesired outcome. For example, the user might just not care about the different behaviour. We cover this aspect insofar as we first abstract the system to its safety-relevant behaviour. After that, all differing behaviour is undesired by definition.

Leveson et al. [14] explicitly view the system as a black box, exactly as us. For them, "a mode defines a mutually exclusive set of system behaviours."

The term "mutually exclusive" alludes to the partitioning of the state space, again. The term "set of system behaviours" is precisely equivalent to our definition.

Leveson et al. allow more than one human controller and more than one automated controller. Nevertheless, the idea is the same to have separate models that must be consistent. We focus on one of the human users only and put all other humans into the environment. Both views can be translated into each other.

These authors distinguish three kinds of modes: supervisory modes, component operating modes, and controlled-system operating modes. This distinction is a direct consequence of their different view on controllers. With our view, the three kinds collapse into two kinds: the modes of the technical system and the modes of the human's mental model of it.

Leveson et al. define that "mode confusion errors result from divergent controller models." This definition is stronger than ours. It requires equivalence between the models, that is, refinement in both directions. We require refinement in one direction only. The latter covers situations where the user does not know how the system will behave, but where the user knows that he/she does not know. Such a situation does not lead to an automation surprise. Leveson et al. are right that this is undesirable. But we prefer to distinguish insufficient knowledge of the user from actual confusion situations.

The definition of Leveson et al. is informal, it does not define precisely the term "divergent". Therefore, it is not clear whether the models must have the same set of traces only or also the same set of failures. Only the latter ensures that no model can refuse an event when the other cannot. We clearly opted for the second choice. Otherwise, a surprise can still happen.

Rushby [9,26] has implicit definitions of mode and mode confusion only. Rushby writes [26]: "Complex systems are often structured into 'modes' [...], and their behavior can change significantly across different modes. 'Mode confusion' arises when the system is in a different mode than assumed by its operator; this is a rich source of automation surprises, since the operator may interact with the system according to a mental model that is inappropriate for its actual mode." We do not see any contradiction to our rigorous definitions.

Rushby describes his model-checking approach [26]: "If we accept that automation surprises may be due to a mismatch between the actual behavior of a system and the operator's mental model of that behaviour, then one way to look for potential surprises is to construct explicit descriptions of the actual system behavior, and of a postulated mental model, and to compare them." This does not say yet how the models are compared. Otherwise, it matches our approach. Rushby's actual comparison of models is determined strongly by the tool he uses, Murϕ. Murϕ can check one model against a set of properties, but not two models against each other. Therefore, Rushby must encode the comparison indirectly. Buth [13] discusses this in detail. In the end, Rushby explicitly agrees that the two-model approach of FDR and CSP (as we use it) would have been better.

Degani & Heymann et al.: Degani et al. [12] "define a mode as a machine configuration that corresponds to a unique behavior." The term "unique behavior" matches our notion. The term "machine configuration" already reveals that they do not use a black-box view on the technical system. Together with Leveson et al., we think that a black-box view is necessary. Because of their white-box view, Heymann and Degani [31] propose a formal abstraction algorithm. This algorithm generates a (minimal) black-box de-

scription from an internal machine description that includes non-observable events. This is helpful if one does not have a (black-box) requirements document in the beginning. Nevertheless, we prefer to start with an explicit requirements document.

Degani et al. [12] distinguish clearly between physical and actually observed events. This matches the respective part of our definition. They point out that the user must be able to sense the input events that trigger a transition, and that "the user's job of integrating events, some of which are located in different displays, is not trivial". These recommendations are close to our recommendations to check that the user can both physically and psychologically observe all safety-relevant events.

Degani et al. [12] state a prerequisite for mode confusion: the user's inability to anticipate the future behavior of the machine leads directly to confusion and error. As with Leveson et al., this description includes non-surprise situations where the user knows that he/she does not now what will happen.

Degani and Heymann [18] add a formal verification algorithm. The models must "march in synchronization". This means automata equivalence, as with Leveson et al. They construct a "composite model" in a fashion similar to Rushby, and they state three correctness criteria for the composite model.

The check for equivalence is motivated by Degani's and Heymann's desire to construct a *minimal* safe mental model. We agree that a minimal safe mental model should be in an equivalence relation with the machine. However, we define correctness separately from minimality.

A prerequisite for their entire approach is that the machine is deterministic. This eliminates the difference between our failures refinement and the simpler traces refinement. Our approach can handle non-deterministic mod-

els that nevertheless do not imply a mode confusion.

Buth [13, pp. 183] writes: "Modes are identifiable and distinguishable states of a system which differ with regard to the effect of interactions." This is exactly the same as our notion. She continues: "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 may lead to potentially critical actions of the operator." This also is exactly our intuition.

Later (pp. 199), Buth uses failures refinement in CSP. We adopted this idea from her. A major part of her work is the comparison of Rushby's one-automaton approach to the two-automaton failures refinement approach, for checking the two models against each other. She finds that the failures refinement approach is better.

A difference to our approach is that Buth requires mutual failures refinement, i. e., equivalence. This is due to the notion of mode confusion that Rushby uses and which Buth investigates.

Buth does not consider senses, and she does not consider the task of abstraction formally. However, in one case she hides an event manually that the user cannot perceive. And she discusses abstraction, but only in the light of model checking and state space explosions, not with respect to safety-relevance (pp. 208).

Hourizi & Johnson [10,33] criticize the mode confusion detection efforts in the literature. They stress that the underlying problems are a (mode) confirmation bias and selective (mode-confirming) perception of the human user. These are covered by our definition, too. They manifest themselves as a lossy relation $\text{SENSE}_{\text{SAFE}}$ whose lossiness depends on the current mode.

We already stated that *any* imperfect relation $\text{SENSE}_{\text{SAFE}}$ is bound to

cause trouble. One should fix the perception first and then perform the formal verification of the rest. Therefore, it does not matter that it would indeed be rather difficult to obtain an explicit $\text{SENSE}_{\text{SAFE}}$ that is sufficiently precise in its mode-dependent lossiness.

References

[1] Charles E. Billings. "Aviation automation: the search for a human-centered approach". Human factors in transportation. Lawrence Erlbaum Associates Publishers, Mahwah, N.J. (1997).

[2] E. Palmer. "Oops, it didn't arm." – A case study of two automation surprises. In "Proc. of the 8th Int'l Symp. on Aviation Psychology" (1995).

[3] D. A. Norman. Some observations on mental models. In Dedre Gentner and Albert L. Stevens, editors, "Mental Models", pages 7–14. Lawrence Erlbaum Associates Inc., Hillsdale, NJ, USA (1983).

[4] J. J. Cañas, A. Antolí, and J. F. Quesada. The role of working memory on measuring mental models of physical systems. *Psicologíca* **22**, 25–42 (2001).

[5] John Rushby. Modeling the human in human factors. In Udo Voges, editor, "Computer Safety, Reliability and Security – 20th Int'l Conf., SafeComp 2001, Proc.", volume 2187 of "LNCS", pages 86–91, Budapest, Hungary (September 2001). Springer.

[6] J. Crow, D. Javaux, and J. Rushby. Models and mechanized methods that integrate human factors into automation design. In K. Abbott, J.-J. Speyer, and G. Boy, editors, "Proc. of the Int'l Conf. on Human-Computer Interaction in Aeronautics: HCI-Aero 2000", Toulouse, France (September 2000).

[7] R. W. Butler, S. P. Miller, J. N. Potts, and V. A. Carreño. A

formal methods approach to the analysis of mode confusion. In "Proc. of the 17th Digital Avionics Systems Conf.", Bellevue, Washington, USA (1998).

[8] José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering* **8**, 275–310 (2001).

[9] John Rushby. Analyzing cockpit interfaces using formal methods. In H. Bowman, editor, "Proc. of FM-Elsewhere", volume 43 of "Electronic Notes in Theoretical Computer Science", invited paper presented at Pisa, Italy, in October 2000 (2001). Elsevier.

[10] Rachid Hourizi and Peter Johnson. Beyond mode error: Supporting strategic knowledge structures to enhance cockpit safety. In "HCI 2001, People and Computers XIV", Lille, France (10–14 September 2001). Springer.

[11] Nadine B. Sarter and David D. Woods. How in the world did we ever get into that mode? Mode error and awareness in supervisory control. *Human Factors* **37**(1), 5–19 (March 1995).

[12] A. Degani, M. Shafto, and A. Kirlik. Modes in human-machine systems: Constructs, representation and classification. *Int'l Journal of Aviation Psychology* **9**(2), 125–138 (1999).

[13] Bettina Buth. "Formal and Semi-Formal Methods for the Analysis of Industrial Control Systems". Habilitation thesis, University of Bremen, Germany (2001).

[14] N. G. Leveson, L. D. Pinnel, S. D. Sandys, S. Koga, and J. D. Reese. Analyzing software specifications for mode confusion potential. In "Workshop on Human Error and System Development", Glasgow, UK (1997).

[15] Gavin Doherty. "A Pragmatic Approach to the Formal Specification of Interactive Systems". PhD thesis, University of of York, Dept. of Computer Science (October 1998).

[16] Harold Thimbleby. "User Interface Design". ACM Press, New York, USA (1990).

[17] Peter Wright, Bob Fields, and Michael Harrison. Deriving human-error tolerance requirements from tasks. In "Proc. of ICRE'94 – IEEE Int'l. Conf. on Requirements Engineering", Colorado, USA (1994).

[18] Michael Heymann and Asaf Degani. Constructing human-automation interfaces: A formal approach. In "Proc. of the Int'l Conf. on Human-Computer Interaction in Aeronautics: HCI-Aero 2002", pages 119–125, Cambridge, MA, USA (October 2002).

[19] M. Rodriguez, M. Zimmermann, M. Katahira, M. de Villepin, B. Ingram, and N. Leveson. Identifying mode confusion potential in software design. In "Proc. of the 19th Digital Avionics Systems Conf." [41].

[20] M. Zimmermann, M. Rodriguez, B. Ingram, M. Katahira, M. de Villepin, and N. Leveson. Making formal methods practical. In "Proc. of the 19th Digital Avionics Systems Conf." [41].

[21] M. Sage and C. W. Johnson. Formally verified, rapid prototyping for air traffic control. *Reliability Engineering & System Safety* **75**(2), 121–132 (February 2002).

[22] S. S. Vakil and R. J. Hansman, Jr. Approaches to mitigating complexity-driven issues in commercial autoflight systems. *Reliability Engineering & System Safety* **75**(2), 133–145 (February 2002).

[23] J. Rushby, J. Crow, and E. Palmer. An automated method to detect potential mode confusions. In "Proc. of the 18th Digital Avionics Systems Conf.", St. Louis, Montana, USA (1999). IEEE.

[24] G. Lüttgen and V. Carreño. Analyzing mode confusion via model checking. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, "SPIN' 99",

volume 1680 of "LNCS", pages 120–135. Springer (1999).

[25] Axel Lankenau. Avoiding mode confusion in service-robots. In M. Mokhtari, editor, "Integration of Assistive Technology in the Information Age, Proc. of the 7th Int'l Conf. on Rehabilitation Robotics", pages 162–167, Evry, France (April 2001). IOS Press.

[26] J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering & System Safety* **75**(2), 167–177 (February 2002).

[27] James Reason. "Human error". Cambridge University Press (1990).

[28] Donald A. Norman. "The Psychology of Everyday Things". Basic Books, New York (1988).

[29] Edward Bachelder and Nancy Leveson. Describing and probing complex system behavior: A graphical approach. In "Proceedings of the Aviation Safety Conference", Seattle, USA (September 2001). Society of Automotive Engineers, Inc. Paper number 01D-22.

[30] Asaf Degani and Michael Heymann. Formal verification of human-automation interaction. *Human Factors* **44**(1), 28–43 (2002).

[31] Michael Heymann and Asaf Degani. On the construction of human-automation interfaces by formal abstraction. In S. Koenig and R. Holte, editors, "SARA 2002", number 2371 in LNAI, pages 99–115. Springer (2002).

[32] D. Javaux. Explaining Sarter & Woods' classical results. The cognitive complexity of pilot-autopilot interaction on the Boeing 737-EFIS. In "Proc. of HESSD '98", pages 62–77 (1998).

[33] Rachid Hourizi and Peter Johnson. Unmasking mode error: A new application of task knowledge principles to the knowledge gaps in cockpit design.

In "Interact'01, The 18th IFIP Conf. on Human Computer Interaction", Tokyo, Japan (9–14 July 2001). Springer.

[34] C. A. R. Hoare. "Communicating Sequential Processes". Prentice-Hall (1985).

[35] A. W. Roscoe. "The Theory and Practice of Concurrency". Prentice-Hall (1997).

[36] Jan Bredereke and Axel Lankenau. A rigorous view of mode confusion. In Stuart Anderson, Sandro Bologna, and Massimo Felici, editors, "Computer Safety, Reliability and Security – 21st Int'l Conf., SafeComp 2002, Proc.", volume 2434 of "LNCS", pages 19–31, Catania, Italy (September 2002). Springer.

[37] Jan Bredereke. On preventing telephony feature interactions which are shared-control mode confusions. In Daniel Amyot and Luigi Logrippo, editors, "Feature Interactions in Telecommunications and Software Systems VII", pages 159–176, Amsterdam (June 2003). IOS Press.

[38] Jan Bredereke. "Maintaining Families of Rigorous Requirements for Embedded Software Systems". Habilitation thesis, University of Bremen, Germany (2004). *To appear.*

[39] Thomas Röfer and Axel Lankenau. Architecture and applications of the Bremen Autonomous Wheelchair. *Information Sciences* **126**(1-4), 1–20 (July 2000).

[40] Axel Lankenau and Thomas Röfer. The Bremen Autonomous Wheelchair – a versatile and safe mobility assistant. *IEEE Robotics and Automation Magazine, "Reinventing the Wheelchair"* **7**(1), 29–37 (March 2001).

[41] IEEE. "Proc. of the 19th Digital Avionics Systems Conf.", Philadelphia, PA, USA (7–13 October 2000).