Analysing the Impact of Deviations in Task Performance When a User Error May Have Safety-critical Consequences

Fabio Paternò & Carmen Santoro
Istituto CNUCE – C.N.R.
Via V. Alfieri 1, Loc. Ghezzano
Pisa, Italy
{fabio.paterno, carmelina.santoro}@cnuce.cnr.it

ABSTRACT
Interactive safety-critical applications have specific requirements in the design cycle that cannot be completely captured by traditional techniques. In this paper we present a proposal aiming to support designers in analysing the space of design solutions and improve both usability and safety of the system considered. This is achieved by focusing the attention of designers on what can happen when abnormal interactions and behaviours occur while users carry out their activities. The result of the analysis can be contained in tables that can be useful to represent design rationale and suggestions for improvements.

Keywords
Design of interactive safety-critical applications. Design rationale. Tasks and errors. Integration of usability and safety.

INTRODUCTION
As interactive safety-critical applications have unique requirements with respect to all other application domains, their design cycle is not well supported by traditional techniques. They differ from more common office applications because in some cases the effects of user interactions cannot be undone even if such wrong interactions may threaten human life. Thus, this type of application requires design solutions able to provide an integrated compliance to usability and safety requirements. Usability criteria may need to be revised in this context. For example, it may be acceptable to increase the number of user actions if it provides benefits in terms of greater safety. In other cases the choice of some direct manipulation techniques may be discarded as it increases the possibility of slips errors that can have safety-critical effects.

As a consequence, in this class of applications the design has to be carefully analysed and evaluated, and techniques that have been successfully applied in other domains may not be adequate. In particular, there is a strong need for systematic methods for analysing the design of a user interface and how it supports abnormal situations.

Another important aspect is how to represent the result of this analysis. In the design rationale field, the QOC (Questions, Options, Criteria) notation [5] has long been considered. However, we found that this approach provides little support in identifying the design issues and the related options, especially those that are relevant in safety-critical contexts. From the evaluation point of view, often inspection-based techniques, such as Cognitive Walkthrough [10], are used. However, such techniques mainly aim to understand whether users will be able to perform effectively the right interactions and to interpret the associated system feedback whereas little attention is paid to understand whether they may perform wrong interactions and what the effects of such wrong interactions are.

Scenario-based design and evaluation [1] have shown to be useful to address and highlight important issues. However, they need the use of other complementary techniques to overcome some limitations. Scenario-based approaches often consider a restricted number of particular sequences of tasks; different scenarios may generate conflicting recommendations, and, in safety-critical applications, designers tend to analyse scenarios that are already known to be problematic on the basis of their knowledge of previous accidents. This can be useful to obtain a design to prevent that incidents occur again but may not be sufficient to identify new possible problems. Thus, there is a need for methods able to give more systematic support, especially when more safety-critical contexts are considered. In particular, the goals of this work are:

- To help designers to thoroughly evaluate the design of applications where both usability and safety are main concerns;
- To provide documentation of design solutions and the associated analysis. This is useful especially when safety-critical applications are considered. Especially when they have to be modified by people that are different from those who designed them and consequently need to clearly understand what was the underlying design rationale.

In [3] the MECHA (Method for Evaluation of Cooperation, Hazards and Allocation) framework supporting the analysis and comparison of a set of design options was presented. Here we present a new proposal for one of the main components of such a framework, namely the analysis of deviations. We focus on such a systematic analysis in order to have a more refined method indicating how to use the information contained in task models formally represented and to consider a wider set of classes of deviations.

In the paper we first describe the method and then we show examples of application taken from a case study, a real prototype for the control of air traffic in an airport. Finally some concluding remarks and indications for future work are provided.

THE METHOD

The method helps designers analyse systematically what happens if there are deviations in task performance with respect to what was originally planned. It indicates a set of predefined classes of deviations that are identified by *guidewords*. A guideword [6] is a word or phrase that expresses and defines a specific type of *deviation*. Guidewords have been found to be useful for stimulating discussion as part of an inspection process about possible *causes* and *consequences* in deviations of user interactions. Mechanisms that aid the *detection* or *indication* of any hazards are also examined and the results are recorded.

Analysis of user deviations has already been considered in previous approaches (such as THEA [2] and CREWS-SAVRE [9]). Our method differs from these approaches for a number of reasons. We start from the task model aiming to address more thoroughly issues related to the performance of the tasks and their relationships instead of limiting to a small set of scenarios that could identify performance of only specific sequences of a small number of tasks. We consider a different set of types of deviations. Given these different elements, the method we use for analysing the deviations is rather different and can provide rather different results.

In the analysis, we consider different categories of tasks: tasks performed by the user or the system or an interaction between them. The basic idea is that we consider the tasks supported by the design that has to be evaluated and the possible deviations that can occur in the performance of such tasks. We consider deviations from how the design analysed assumes that tasks are performed. Interpreting the guidewords in relation to a task allows the analyst to systematically generate ways the task could potentially go wrong, as a starting point for further discussion and investigation. Such analysis should generate suggestions about how to guard against deviations as well as recommendations about user interface designs that might either reduce the likelihood of the deviation or support its detection and recovery.

The method is composed of three steps:

- *Development of the task model of the application considered*; this means that the design of the system is analysed in order to identify how it requires that tasks are performed. The purpose is to provide a description logically structured in a hierarchical manner of tasks that have to be performed, including their temporal relationships, the objects manipulated and the tasks' attributes. We use the ConcurTaskTrees notation [8] but other notations for task modelling (such as UAN [4]) could still be suitable.
- *Analysis of deviations related to the basic tasks*; the basic tasks are the leaves in the hierarchical task model, tasks that the designer deems should be considered as units.
- *Analysis of deviations in high-level tasks*, these tasks allow designer to identify *group of tasks* and consequently to analyse deviations that involve more than one basic task. Such deviations concern whether the appropriate tasks are performed and if such tasks are accomplished following a correct ordering.

The above steps are performed in the order they have been introduced. As a consequence we have a bottom-up approach that allows designers first to focus on concrete aspects and then to widen the analysis to consider more logical steps.

We have found useful to investigate the deviations associated with the following guidewords:

- *None*, the unit of analysis, either a task or a group of tasks, has not been performed or it has been performed but without producing any result;
- *Other than*, the tasks considered have been performed differently from the designer's intentions specified in the task model;
- *Ill-timed*, the tasks considered have been performed at the wrong time.

Such high-level guidewords identify classes of deviations rather than specific deviations. Thus, in our analysis, these guidewords are further refined. The meaning of the resulting types of deviations slightly changes depending on whether designers consider one basic task or a group of tasks.

A basic task is an elementary activity that cannot be further decomposed or its decomposition is not relevant for the current use of the task model. If *one single basic task* is considered then we use the guidewords in the following manner:

- *None*, it is decomposed into three types of deviations depending on why the task performance has not produced any result. It can be due to a *lack of initial information* necessary to perform a task or because the *performance has not occurred* or because it occurred but, for some reason, *its results are lost.* An example is when the user updates a system and for some reason the results of this updating are not presented on the screen.
- *Other than*, in this case we can distinguish when *less, more or different information* is used in the task performance with respect to what is planned. Each of this sub-case may be applied to the analysis of the input, performance or result of a task, thus identifying nine types of deviations (less input, less performance, less output, other than input, other than performance, other than output, more input, more performance, more output). An example of the less input sub-case is when the user sends a request without providing all the information necessary to perform it correctly. Thus, in this case the task produces some results but they are likely wrong results.
- *Ill Timed*, here we can distinguish at least when the performance occurs *too early or too late*.

When high-level tasks are analysed in some cases they should be considered as a single logical entity whereas in other cases it is better to focus on their sub-components. In addition, when groups of tasks are considered then the meaning of the deviations can be different. More precisely, we have:

- *None*. This case is similar to the corresponding case for basic tasks as it means that the considered group of tasks either is not performed at all or it does not produce any result. It can be decomposed in *no input* and *no performance* sub-cases. This analysis is better performed if the designer focuses on the parent task, in order to pay attention to the activity of the task as a whole (in case of no performance), or on the subtasks that are supposed to be performed at the beginning (in case of no input).
- *Other than*. In this case the interpretation of the sub-guidewords changes. *Less* means that less subtasks have been performed than required. *More* means that some additional, useless subtasks have been performed. *Other than* indicates that wrong temporal relationships have been applied to the sub-tasks. For example, in case of a group of tasks that should be executed sequentially, the designers want to understand what happens if they are performed concurrently, or one task is interrupted by another one or they have been performed as they were in alternative choice.
- *Ill timed*. It has the same meaning as for basic tasks. This analysis is better performed focusing on the parent task.

The systematic application of this method to a real case study can involve hundreds of tasks. Then, an exhaustive analysis of all the possibilities can take a considerable amount of time and effort. Thus, we found more practical to restrict the evaluation to an exhaustive analysis of the parts of the application that can have more impact on safety aspects. Interactive safety-critical applications usually consist in applications where some trained users control an external world interacting at real-time with it (examples are air traffic control systems, nuclear power plants, railways systems, and aeroplane cockpits). In this case safety-critical situations occur when for some reasons the controlled entity deviates its behaviour from the planned one and the user (the controller) has to react rapidly in order to avoid dangerous effects. A combination of user deviations with system deviations can move the system in highly hazardous states. Thus, in these cases the user interface should be organised so as to support carefully the users in avoiding them or to mitigate as much as possible the hazard of their consequences.

The result of the analysis is stored in tables that share the same structure. For each element analysed there is one table with a number of elements:

- *Task*, indicating the task currently analysed*;*
- *Guideword*, indicating the type of deviation considered;
- *Explanation*, explaining how the deviation has been interpreted for that task or group of tasks;
- *Causes*, indicating the potential causes for the deviation considered and which cognitive fault might have generated the deviation;
- *Consequences*, indicating the possible effects of the occurrence of the deviation in the system;
- *Protection*, describing the protections that have been implemented in the considered design in order to guard against either the occurrence or the effects of the deviation;
- *Recommendation*, providing suggestions for an improved design able to better cope with the considered deviation.

Such tables can be used as documentation of a system and its design rationale, giving exhaustive explanation of how an unexpected use of the system has been considered and which type of support has been provided in the system to handle such abnormal situations. Many types of strategies can be followed for this purpose; these range from

techniques aiming to prevent abnormal situations, to others that allow such situations, but inform the user of their occurrence in order to either mitigate their effects or aid in recovering from them.

Besides providing guidewords during the analysis process for the different types of deviations that may occur, it is also useful to identify guidewords for the possible causes of deviations. To this end we can consider the phases of the cycle of an interaction [7]. Thus, we can identify at least the following possible cognitive causes of deviations:

- *Intention*, the intention is in some sense faulty;
- *Action*, the actions are carried out incorrectly;
- *Perception*, deviation results from an incorrect perception;
- *Understanding*, the deviation results from a mis-understanding of the perceived information;
- *Evaluation*, the elements composing the deviation are correctly interpreted but they are erroneously evaluated
- *Memory*, the user does not remember some information useful for performing the task.

The technique proposed helps to identify potential hazards. However, a designer's decision about what to do about a potential failure will depend on a deeper assessment of the hazard, its causes and consequences.

We believe that this type of analysis is particularly effective when performed by a multidisciplinary team, that includes designers, software developers, and end users. Each contribution can enrich the discussion with specific expertise and knowledge, helping to identify possible deviations and the most effective design choices.

A CASE STUDY: DATA LINK COMMUNICATION TO CONTROL AIR TRAFFIC IN AIRPORTS

We have applied the method to a real prototype for the air traffic control in an aerodrome. The main purpose of the system is to support data-link communications to handle the air traffic between the runways and the gates. Data link is a technology allowing asynchronous exchanges of digital data coded according to a predefined syntax. This allows replacing or complementing current communication technology that is based mainly on radio communication. This support provides controllers with graphical user interfaces showing the traffic within an airport in real-time and messages received from pilots. In addition, in this type of applications controllers can compose messages by direct manipulation techniques.

This system replaces traditional paper strips with enriched flight labels. There are labels that can interactively enlarge to show additional information concerning the flight and shrink again showing just essential information on the flight. In Figure 1 we can see an example for the control of air traffic in the Rome Fiumicino airport. It is part of the user interface of the "ground" controller, who is in charge of handling the aircraft on the taxiways until they reach the beginning of the runway (for departing aircraft) or the assigned gate (for arriving aircraft).

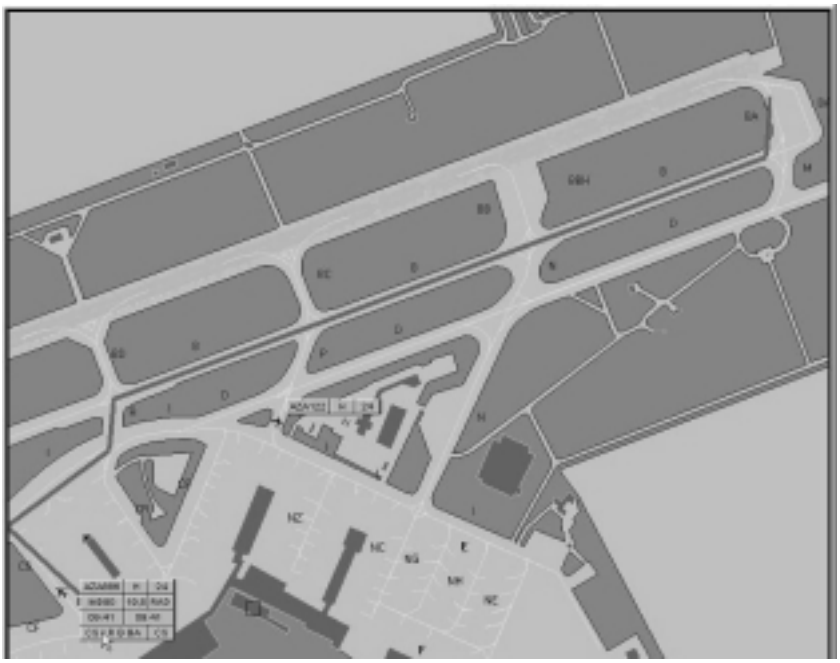

Figure 1: A part of the user interface of the ground controller.

In the airport area represented in Figure 1 there are two aircraft. Enriched flight labels give information concerning flights. They have a standard set of information (see flight AZA122) including the flight identifier, the category, and the assigned runway. When a label is selected, an extended set of information is given (see flight AZA886). The path assigned to a flight can be graphically represented (for example, see the path associated with the flight AZA886 in Figure 1).

Figure 2 shows a small excerpt of the task model of the application. We consider it as a small example to introduce the method. Here, we consider the *Handle spatial deviation* task. It is a task that can be performed multiple times (this is indicated by the * symbol). It is decomposed into *Check deviation, Check path, and Handle deviation* tasks. They are sequential tasks with information passing ([]>> operator) from one to the next. The last two are optional tasks. The squared brackets around their name indicate this. This means that their performance is not required to complete the overall *Handle spatial deviation* task. *Check path* is decomposed into a sequence of tasks: *Select flight label*, *Show Extended Labe*l, *Select path field*, *Show graphical path*, *Check deviation*. Some are interactive tasks, two are application tasks (those with the computer icon). *Handle deviation* is decomposed into *Analyse deviation features* and *Decide strategy*, which is a user task because it basically involves the cognitive activity of deciding a solution for the problem.

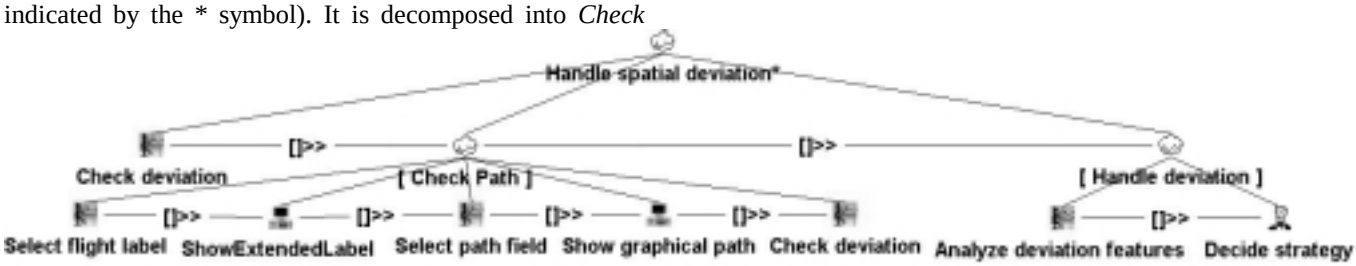

Figure 2: An excerpt of the task model of a ground controller.

EXAMPLE OF APPLICATION OF THE METHOD

To show how the method works we consider both an example of single task and an example of group of tasks taken from the excerpt of the task model introduced in the previous section.

Analysis of deviations in a basic task

In this case we analyse the *Check Deviation* task. In order to apply the *None* guideword we need to consider what the input and output for the performance of such a task are. The input is the information concerning the air traffic and the controller's knowledge of the path that the aircraft should follow to get to the beginning of the runway. The output is whether the position is on the correct path or there is a deviation. As you can see in the following tables, it is not necessary to consider always all the sub-cases because some of them may not be meaningful for specific tasks. For example, in the *other than* category of the Check deviation task, the *more* sub-case is not particularly significant because in this context to have either more input for detecting deviation task or assuming that it provides more output does not seem particularly meaningful. Likewise, checking deviation too early does not seem particularly meaningful because it is never "too early" for checking the safety of the current situation of traffic.

In other cases, two types of deviation are tightly connected, such as different input and different output in the Check deviation task. Indeed, if controllers have a wrong belief of what the assigned path is then they can make wrong decisions in checking deviations.

There are also cases that substantially coincide. For instance, some examples of wrong performance and wrong output of task can be considered in similar manner in their analysis.

Table 1, 2 and 3 show the main elements of the analysis for the considered task and they are structured so as to be self-explanatory.

Analysis of deviations of a group of tasks associated with a high-level task

When a group of tasks are considered for the analysis of deviations we can use tables similar to those previously seen. Instead of a single task we will have to indicate the high-level task, its subtasks and the related temporal relationships. For example, we can consider *Handle Spatial Deviation* that is decomposed into *Check Deviation*, *Check Path*, *Handle Deviation*. They are three sequential tasks. The last two are also optional tasks.

In the analysis of this group of tasks, we have not shown the discussion related to some cases that provide elements similar to those raised by the *Check deviation* task (for example in the less and different input sub-cases). In Table 4 we have focused on those cases that are more related to the relationships among the subtasks composing the high-level activity considered.

Task: Check Deviation		**Guideword: None**		
Explanation	**Causes**	**Consequences**	**Protections**	**Recommendations**
No input The controller has not information concerning the current and planned traffic	A temporary fault occurred in the system *Perception* problem	The controller has not the necessary information and so he cannot check whether a certain deviation occurs. If the deviation has occurred, it can evolve without possibility to e handled by the controller.	The controller expects this information on UI: lack of it could alert him (e.g. he looks at the tower window)	Provide redundant devices to compensate for lack of information (e.g.: another display available to the controller)
	The controller is distracted or interrupted by other activities *Perception problem*		Pilot could suspects that something got wrong and calls the controller to check the path	Provide an automatic warning message to highlight that lack of information has occurred
No performance The controller has the information but he does not check it, that is he does not compare current and planned positions of flights	Controller is either distracted or over confident *Interpretation* problem The controller forgets to execute the activity (*Memory* problem)	The controller cannot know whether the flight is following the right path or a deviated one In case of deviation: a) the controller is not able to locate *where* the deviation has occurred (and other aircraft potentially in conflict) b) the controller cannot react appropriately c) the deviation goes ahead: the hazardous situation might get worst.	A suspicious pilot calls controller to check the path	Provide an automatic mechanism of conformance monitoring on the system that warns the controllers just in case of a deviation.
No output The controller has not retained in his memory general information about the current/planned situation of flight.	Controller is interrupted by other activities Controller forgets information			

Table 1. Table for "None" guideword

Task: Check Deviation		**Guideword: Other than**		
Explanation	**Causes**	**Consequences**	**Protections**	**Recommendations**
Less Input The controller does not remember the planned path of the flight	The controller does not hold in his memory the planned path (*Memory* problem)			Display automatically information about planned positions only when a deviation occurs using techniques that become more and more intrusive as more the deviation goes ahead without any intervention of the controller.
The controller considers only a part of the current traffic	The controller does not pay attention, or it is distracted by other activities and does not look carefully at all the information *Perception* problem			

	Temporary fault of the application (only a portion of the information is displayed on the UI) *Perception* problem		The controller might realise by himself that some data are missing	Provide redundant mechanisms to compensate for lack of information
Different **Output/ performance** The controller has the correct input but performs check erroneously	Cognitive slip because of work overloading *Understanding Problem*	He could not detect any deviation (though it exists). He might detect deviation (that not exists).		Provide an automatic mechanism of conformance monitoring on the system that warns the controllers just in case of a deviation.
Different **Input** The controller believes that a different path has been assigned or he has wrong information of the current situation of traffic	Fault of the system *Perception* problem Controller is distracted or over confident *Interpretation* problem	He might not detect deviation (when they exist) He might detect deviation (that not exists).	The controller might cross-check the information (wrongly) displayed on the UI with that gathered by looking out of the Control Tower window.	Provide redundant mechanisms to compensate for errors in the system.

Table 2. Table for "Other than" guideword

Task: Check Deviation		**Guideword: Ill-timed**		
Explanation	**Causes**	**Consequences**	**Protections**	**Recommendations**
Too late The controller checks deviation after a long time it occurs *Interpretation* or *Understanding* problem	Controller is distracted or over confident	The controller gets aware of a possible hazardous situation when the situation has become more problematic In case of deviation the controller has less time to find a strategy and solve the problem. The controller might have to guess different strategies than usual ones to solve problems, then he can further lose time and the might tend to make errors	In case of deviation, other stakeholders might have already detected it and called the controller (e.g. the pilot)	The warning associated with the automatic detection of deviation should become more and more intrusive if there is no reaction from the controller

Table 3. Table for "ill timed" guideword

Task: Handle Spatial Deviation			**Guideword: Other than**	
Subtasks: *Check Deviation* []>>[*Check Path*] []>>[*Handle Deviation*]				
Explanations	**Causes**	**Consequences**	**Protections**	**Recommendations**
Less performance There is a deviation but the controller does not handle it appropriately and completely	Controller thinks that it has no impact on safety. The controller deems to have more urgent things to do The controller is distracted by other events	The deviation evolves with possible dangerous consequences.		Provide interaction techniques able to appropriately highlight all the relevant information concerning the deviation
Less performance The controller checks the path of a flight without first analysing possible deviation	The controller performs a check on a flight that has no deviation at all *Interpretation* problem	The controller performs a useless action wasting his time		Provide interaction techniques that help controllers doing activities in the correct order
Less output The controller devises an uncompleted strategy	Controller considers only a part of the elements that define the deviation *Evaluation* Problem	The controller may not able to correctly solve the deviation and he may think to have solved the problem whereas the problem is still holding and thus can get worse		Automatic mechanisms to check completeness of conflict solutions
More performance The controller checks multiple times a flight	The controller does not feel sure of the previous decision and wants to check again the data	The controller spends more time to get more confident on his understanding of the current situation of traffic The controller can get further relevant information that he previously did not notice		Provide tools able to show possible strategies and offer to the controller the possibility and the means able to evaluate each suggested strategy, in order to allow the controller to compare different strategies only once and not to force him to repeat the process more than one time.
Different temporal orders Concurrent instead of sequential	The controller is over confident about his capacity of handling a deviation and performs concurrently the different activities	The solution is taken without considering all the possible aspects and interrelationships		Provide in the UI interaction techniques that help the controllers to carry out activities in the right order
Or Handle deviation interrupts	The controller deems that the collected information is enough to appropriately handle the deviation	The controller can have a partial view of all the data involved in the deviation		Provide some mechanisms that warn the controller when a part of an activity has been abruptly interrupted and possible critical information missing

Table 4. Table for "Other than" guideword

CONCLUSIONS

This paper describes a method supporting systematic analysis of task performance taking into account a wide set of possible deviations that can have an effect upon either a single basic tasks or groups of tasks.

We have shown examples of use of such a method taken from a case study in the air traffic control domain. The systematic analysis requires some effort that is particularly motivated when safety-critical applications are considered. The tables containing the result of the analysis can give additional benefits: they can be useful to represent design rationale and suggestions for further improvements.

We plan to develop some tool support to help designers in this systematic analysis.

ACKNOWLEDGMENTS

MEFISTO is an Esprit Reactive Long Term Research Project (http://giove.cnuce.cnr.it/mefisto.html).

REFERENCES

[1] J.Carroll (ed.), Scenaro-Baed Design: Envisioning Work and Technology in System Development. John Wiley and Sons, 1995

[2] Fields, R.E., Harrison, M.D. and Wright, P.C. (1997). *THEA: Human Error Analysis for Requirements Definition.* University of York, Dept. of Computer Science, Technical Report YCS-97-294. http://www.cs.york.ac.uk/~bob/papers.html

[3] R. Fields, F.Paternò, C.Santoro, S.Tahmassebi, Comparing design options for allocating communication media in cooperative safety-critical contexts: a method and a case study. To appear on ATOCHI, 1999.

[4] R.Hartson, P.Gray, "Temporal Aspects of Tasks in the User Action Notation", Human Computer Interaction, Vol.7, pp.1-45, 1992.

[5] MacLean A., Young R.M., Bellotti V.M.E. and Moran T.P. (1991) Questions, options, and criteria: elements of design space analysis. *Human-Computer Interaction*, 6(3-4), pp.201-250.

[6] MOD HAZOP Studies on Systems Containing Programmable Electronics. UK Ministry of Defence, Interim Def Stan 00-58 Issue 1, (1996).

[7] Norman, D., *The Psychology of Everyday Things*, Basic Books, 1988.

[8] F. Paterno', "Model-Based Design and Evaluation of Interactive Applications". Springer Verlag, 1999.

[9] A.Sutcliffe, N.Maiden, S.Minocha, D.Manuel, "Supporting Scenario-Based Requirement Engineering", IEEE Transactions on Software Engineering, Vol.24, N.12, December 1998, pp.1072-1088.

[10] Wharton, C., Rieman, J., Lewis, C. and Polson, P. (1994) "The Cognitive Walkthrough: A Practitioner's Guide". In *Usability Inspection Methods*, Nielsen J. and Mack R.L. (eds.), John Wiley & Sons.