*Int. J. Human-Computer Studies* (2002) **56,** 225–245
doi:10.1006/ijhc.2001.0523
Available online at http://www.idealibrary.com.on IDEAL®

# Preventing user errors by systematic analysis of deviations from the system task model

Fabio Paternò and Carmen Santoro
*CNUCE-C.N.R., Via G. Moruzzi 1, 56100, Ghezzano Pisa, Italy.*
*emails: f.paterno@cnuce.cnr.it; c.santoro@cnuce.cnr.it*

*(Received 7 February 2001 and accepted in revised form 18 December 2001)*

Interactive safety-critical applications have specific requirements that cannot be completely captured by traditional evaluation techniques. In this paper, we discuss how to perform a systematic inspection-based analysis to improve both usability and safety aspects of an application. The analysis considers a system prototype and the related task model and aims to evaluate what could happen when interactions and behaviours occur differently from what the system design assumes. We also provide a description and discussion of an application of this method to a case study in the air traffic control domain.



KEYWORDS: model-based evaluation; safety and usability; task models; inspection-based evaluation

## 1. Introduction

A number of usability evaluation techniques have been developed in recent years. They can be classified according to many dimensions: for example, the level of refinement of the user interface considered or the amount of user involvement in the evaluation. More generally, usability evaluation methods can be classified as: model-based approaches, where a significant model related to the interactive application is used to drive the evaluation; inspection-based assessment, where some expert evaluates the system, or some representation of it, according to a set of criteria; and empirical testing where direct use of the system is considered.

The research area of model-based design and evaluation of interactive applications (Puerta, 1997; Paternò, 1999) aims at identifying models able to support design, development, and evaluation of interactive applications. Such models highlight important aspects that should be taken into account by designers. Various types of models, such as user, context and task models have proved to be useful in the design and development of interactive applications. Task models describe activities that have to be performed so that user's goals are attained. A goal is a desired modification of the state of an application. The use of task analysis and modelling has long been applied in the design of interactive applications. However, less attention has been paid to their use to support systematic usability evaluation. To this end, it is important to have task

1071-5819/02/020225+21 $35.00/0 

models described by flexible and expressive notations with precise semantics able to represent the different ways to perform tasks and the many possible temporal and semantic relationships among them. This allows designers to develop systematic methods able to indicate how to use the information contained in the task model for supporting the design and evaluation of the user interface.

There are various approaches that aim to specify tasks. They differ in aspects such as the type of formalism they use, the type of knowledge they capture, and how they support the design and development of interactive systems. In this paper, we consider task models that have been represented using the ConcurTaskTrees notation (Paternò, 1999). In ConcurTaskTrees, activities are described at different abstraction levels in a hierarchical manner, represented graphically in a tree-like format (see Figure 1 for an example). In contrast to previous approaches, such as Hierarchical Task Analysis, ConcurTaskTrees provides a rich set of operators, with precise meaning, able to describe many possible temporal relationships (concurrency, interruption, disabling, iteration and so on). This allows designers to obtain concise representations describing many possible evolutions over a user session. The notation also provides the possibility of using icons or geometrical shapes to indicate how the performance of the tasks is allocated. For each task it is possible to provide additional information including the objects manipulated (for both the user interface and application objects) and attributes such as frequency. Automatic tools are needed to make the development and analysis of such task models easier and more efficient. A tool to specify task models in ConcurTaskTrees (CTT) and analyse their content is publicly available at *http://giove.cnuce.cnr.it/ctte.html*.

Task models can also be useful in supporting design and evaluation of interactive safety-critical applications. The main feature of these systems is that they control a real-world entity and have to fulfil a number of requirements while preventing the controlled entity from reaching hazardous states (states where there is actual or potential danger to people or the environment). There are many examples of safety-critical systems in real life (air traffic control, railway systems, industry control systems, etc.). In this field specific issues arise. For instance, in safety-critical domains, sometimes user actions cannot be undone (for example, if an irreversible physical process has been activated), so the issue of *user errors* and how to design the user interface so as to avoid them, acquires a special importance. In fact, many studies have shown that accidents often are caused by a human error whose likelihood may be increased by poor design.

The goal of this paper is to discuss how task models can be used in an inspection-based usability evaluation for interactive safety-critical applications. In order to show

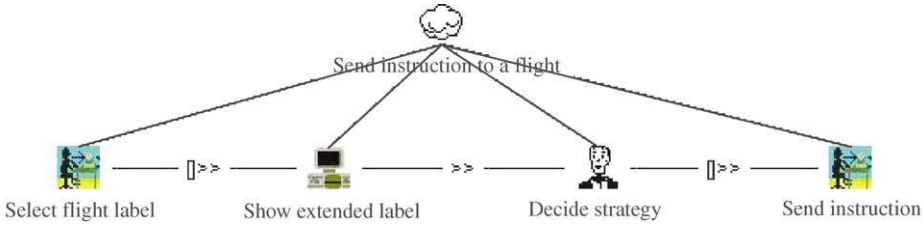


FIGURE 1. An example of high-level task.

how such evaluation works, we consider a case study in the air traffic control application (ATC) domain. However, it is worth emphasizing that the objective of this paper is to present an innovative *method* for analysing safety-critical systems, rather than the design of a new ATC system complying with current ICAO standards.

To this end, we first introduce our approach and a method (an early version was introduced in Paternò & Santoro, 2001) to describe how to use information contained in task models to support an exhaustive inspection-based evaluation. We then extensively discuss a case study where the method was applied with a multidisciplinary team to the air traffic control domain. Finally, we discuss this experience in terms of results and lessons learnt, providing some concluding remarks and indication for further work.

## 2. Task models and usability evaluation

Task models can be useful in various phases of the design cycle. They can play an important role in the requirements elicitation and specification phase [see for example GTA (van der Veer, Lenting & Bergevoet, 1996)], for example, by requiring precise definition of temporal relationships between the different activities that should be performed, avoiding any ambiguities. In addition, task models can be used to support the design of interactive applications: a number of criteria have been identified on how to use the information contained in CTT task models to drive the design of the user interface (Paterno, Santoro & Sabbatino, 2000). Task models satisfy the need to maintain concise and precise documentation of the system under consideration.

They can also be useful in the evaluation phase. The work done so far in model-based evaluation has mainly aimed to support performance evaluation (such as GOMS approaches) to predict task completion times or usability evaluation through the automatic analysis of user interactions (Ivory & Hearst, 1999). An example of the latter techniques can be found in Lecerof and Paternò (1998), where the use of task models in remote usability evaluation is described: the possible activities supported by the application and described by the task model are used to analyse real user behaviour as revealed by automatically recorded log files of user interactions with a graphical interface. This type of evaluation is useful for assessing final versions of applications. However, we think that task models may also help support evaluation in the early phases of the design cycle. Indeed, inspection-based methods are often applied to evaluate prototypes in order to give suggestions for obtaining improved versions. They are less expensive than empirical testing and one advantage is that their application at least decreases the number of usability problems that can be detected by the final empirical testing.

A number of inspection-based evaluation methods have been proposed. In heuristic evaluation (Nielsen, 1993), a set of general evaluation criteria (such as visibility of the state of the system, consistency, avoid providing useless information, etc.) are considered and evaluators have to check whether they have been correctly applied. This method heavily depends on the ability of the evaluator and many software engineers may have problems to understand how to apply such general criteria to their specific cases. In cognitive walkthrough (Wharton, Rieman, Lewis & Polson, 1994), the

evaluators have to identify a sequence of tasks to perform and for each of them four questions are asked. *While this method is clear and can be applied with limited effort, in our opinion it has a limitation: it tends to concentrate the attention of the evaluator on whether or not the user will be able to perform the right interactions. Little attention is paid on what happens if users perform errors (interactions that are not useful for performing the current task) and on the consequences of such errors*. It is easy to understand how crucial this aspect is in interactive safety-critical applications where the consequences of human errors may even threaten human life (see for example, the accident at Linate airport in October 2001 where 118 people died because of human error).

Few works have addressed this type of problem. Reason (1990) introduced a first systematic analysis concerning human error, including the simple slips/mistakes distinction. Human reliability analysis (Hollnagel, 1993) stems from the need to quantify the effects of human error on the risks associated with a system and typically involves estimating the probability of the occurrence of errors, which quickly runs into the problem of acquiring the data necessary for reliable quantification. Practical experience shows that different methods in this area often yield different numerical results, and even the same method may give different results when used by different analysts. Leveson (1995) introduced a set of guidelines for the design of safety-critical applications, but little attention was paid to the user interface component. Some guidelines for safe user interactions design are proposed, but they are too general to be used systematically when designing the user interface. Then, research moved on to finding systematic methods for supporting design and evaluation in this area. THEA (Fields, Harrison & Wright, 1997) uses a scenario-based approach to analyse possible issues. While our approach involves end-users in the evaluation exercise, THEA supports a judgment study performed by experts. Formal methods (Palanque, Bastide & Paternò 1997) have also been considered for this purpose and have shown to be useful in analysing limited parts of these applications. In Galliers, Sutcliffe and Minocha (1999) the authors propose an analysis that supports re-designing a user interface to avoid the occurrence of errors or to at least reduce their effects. The analysis is supported by a probabilistic model that uses Bayesian Belief Nets. Johnson has developed a number of techniques—see for example, Johnson and Botting (1999)—for analysing accident reports, which can be useful to better understand the human errors that have caused real accidents.

The contribution of our method resides in the help that it provides to designers in order to systematically analyse what happens if there are deviations in task performance with respect to what was originally planned during the system design. It indicates a set of predefined classes of deviations that are identified by *guidewords* (MOD, 1996). A guideword is a word or phrase that expresses and defines a specific type of *deviation*. These types of deviations have been found useful for stimulating discussion as part of an inspection process about possible *causes* and *consequences* of deviations during user interactions. Mechanisms that aid the *detection* or *indication* of any hazards are also examined and the results are recorded. In our approach, we build upon the HAZOP family of techniques, originally developed for investigating hazard and operability issues in the chemical process industry. While the basic HAZOP techniques have been used for analysing software-based systems (McDermid &

Pumfrey, 1994) and safety-critical systems (Burns & Pitblado 1993), we note a lack of proposals to introduce them in structured methods for user interface analysis, design and evaluation with the support of the task models.

## 3. The method

In the analysis, we consider the system task model: how the design of the system to evaluate assumes that tasks should be performed. The goal is to identify the possible deviations from this plan. Interpreting the guidewords in relation to a task allows the analyst to systematically generate ways in which the task could potentially deviate from the expected behaviour during its performance. This serves 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 from hazardous states.

The method is composed of three steps:

(1) *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 but other notations for task modelling with similar operators could still be suitable.
(2) *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.
(3) *Analysis of deviations in high-level tasks*: these tasks allow the designer to identify a *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.

It is important that the analysis of deviations be carried out by interdisciplinary groups where such deviations are considered from different viewpoints and backgrounds in order to carry out a complete analysis. The analysis follows a bottom-up approach (first basic tasks, and then high-level tasks are considered) 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:

(1) *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. None 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 task has not been performed* or because it has been performed but, for some reason, *its results are lost*. For example, if we consider the task of selecting an aircraft on a graphical screen, it cannot be performed because the system does not provide correct presentation of the aircraft, or because the information is correctly presented but the controller does not realize the need for the selection, or because the controller mentally selects the aircraft but then forgets to perform the action on the screen because of being interrupted by other activities.

(2) *Other than*, the tasks considered have been performed differently from the designer's intentions specified in the task model; in this case, we can distinguish three sub-cases: less, more or different. Each of these sub-cases 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.

(3) *Ill-timed*, the tasks considered have been performed at the wrong time. Here we can distinguish at least when the performance occurs *early or late* with respect to the planned activity.

The choice of these guidewords is based on the observation that a task is an activity that usually requires some initial information, then its performance occurs and lastly such performance generates some result. A deviation can occur in any of the parts of a task.

In addition, for each task analysed, it is possible to store in one table the result of the analysis in terms of the following information.

(1) *Task*, indicating the task currently analysed.
(2) *Guideword*, indicating the type of deviation considered.
(3) *Explanation*, explaining how the deviation has been interpreted for that task or group of tasks.
(4) *Causes*, indicating the potential causes for the deviation considered and which cognitive faults might have generated the deviation.
(5) *Consequences*, indicating the possible effects of the occurrence of the deviation in the system.
(6) *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 on the system.
(7) *Recommendation*, providing suggestions for an improved design able to better cope with the considered deviation.

Tables can be used as documentation of a system and its design rationale, giving exhaustive explanation of cases when 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.

The process of performing the evaluation by interpreting every guideword for every task of the prototype on the one hand addresses completeness issues; on the other hand has the drawback of taking time and effort. However, what has to be emphasized on this subject is that we are dealing with safety-critical systems, so the costs are amply justified if there is a substantial gain in terms of safety.

We think it is useful to classify the explanation in terms of which phase of the interaction cycle, according to the stages of Norman's model (Norman, 1988), can generate the problem.

(1) *Intention*, the user intended to perform the wrong task for the current goal. An example of intention problem is that while controllers aim to solve an air traffic conflict to obtain a safer state, they decide to increase the level of an aircraft but in this manner they create a new conflict.
(2) *Action*, the task the user intended to perform was correct but the actions identified to support it were wrong. An example of action error is when the controller wants to graphically edit a path to send to an aircraft by selecting points that are not selectable.
(3) *Execution*, the performance of an action was wrong, for example the controller selects a wrong aircraft because of a slip.
(4) *Perception*, the user has difficulties in perceiving or correctly perceiving the information that is provided by the application. A perception problem is when the user takes a long time to find the user interface element necessary to perform the next task.
(5) *Interpretation*, the user misinterpreted the information provided by the application. An interpretation problem is when there is a "More Info" button but the user misunderstands for what topic more information is available.
(6) *Evaluation*, in this case the controller has perceived and correctly interpreted the information but it is wrongly evaluated, for example the controller detects an air traffic conflict that does not exist.

Many types of strategies can be followed when a deviation occurs: these range from techniques aiming to prevent abnormal situations, to others that allow such situations to arise, but inform the user of their occurrence in order to either mitigate or aid in recovering from potential problems (if any). To better explain how the method works, we consider an example of both basic task and high-level task.

Table 1 presents an example of the analysis of a basic task in an air traffic control case study where controllers can also interact with graphical representations of the current airport traffic. The task considered is *Check deviation* (the user checks whether an aircraft is following the assigned path in the airport). The class of deviation considered is *None*. First, we have to identify the information required to perform the task. In this case, it is the state of the traffic in the airport and the path associated with the aircraft under consideration. If we consider the *No input* case, we can note that it can have different causes, both generating perception problems: either a system fault has occurred or the controller is distracted by other activities. This shows that the method is able to consider both system and user errors. In any case, the consequence is that the controller has no updated view of the air traffic.

TABLE 1
*Example of analysis of task deviations*

| Task: Check deviation<br>Explanation | Causes | Consequences | Protections | Guideword: None<br>Recommendations |
|---|---|---|---|---|
| *No input*<br>The controller has no information concerning the current traffic | System failure<br>*Perception* problem | The controller has no updated view of the air traffic | The controller looks at the window to check the air traffic | Duplication of communication system |
| | Controller is distracted or interrupted by other activities<br>*Perception* problem | | Pilot calls controller to check the path | Provide an automatic warning message only when an aircraft is deviating from the assigned path. |
| *No performance*<br>The controller has the information but he does not check it carefully | Controller is distracted or overconfident<br>*Interpretation* problem | | Red line in the case of runway incursion | |
| *No output*<br>The controller finds a deviation but immediately forgets it | Controller is interrupted by other activities<br>*Intention* problem | | | |

The protection in the current system changes accordingly, if the system supporting the graphical representation is not functioning, then the controller can only look through the window in order to check the state of the traffic (according to current ICAO procedures). If the controller is distracted then pilots, especially if they perceive that something abnormal is occurring, may contact the controller. The recommendations for an improved system change depending on the case considered: duplication policy should be followed against system faults whereas warning messages should be provided in the case of aircraft deviating from the assigned path. Slightly different possibilities are considered in the other sub-cases. For instance, in the *No performance* case we consider when the information is available but for some reason the task is not performed: the controller is distracted or overconfident, so he does not correctly interpret the information. In the *No output* case, we have that the task is performed but its results are lost, e.g. the controllers find a deviation but they forget it because they are unexpectedly interrupted by another activity.

When not considering basic tasks, the deviations should be applied in a slightly different manner although we can still use similar tables to store the results of the analysis. In fact, in these cases, the interpretation and the application of each guideword to a higher-level task has to be properly customized depending on the temporal relationships existing between its subtasks. For example, consider a higher-level task with its basic tasks connected by the sequential enabling operator ([]$\gg$ or $\gg$ depending on whether information is exchanged between the tasks or not), for example *Send instruction to a flight* task in Figure 1. Because of the temporal relationship that relates the first left subtask to the others, a lack of input for the first basic task means a lack of input for the whole parent task. Therefore, this case (*None/No input*) of the analysis of the parent task can be brought back to the correspondent case of its first left child. In this case, the *Select flight label* task cannot be performed because, for example, the controller is wrong at selecting the label. The *None/no performance* case means that no lower-level subtask (*Select flight label*, *Show extended label*, *Decide strategy* and *Send instruction*) has been performed. The *None/No output* case is when all the subtasks have been carried out, but no output has been produced at the end. Referring to the aforementioned *Send instruction to a flight* task in Figure 1, this case can be brought back to the *None/no output* case of the *Send instruction* task.

For the other guidewords, the reasoning changes accordingly. For example, the *Less* guideword applied to the *Send instruction to a flight* task means that one associated subtask has not been carried out. For example, the user forgets to perform the last subtask (*Send instruction*) after having performed the other subtasks (*Select flight label*, *Show extended label* and *Decide strategy)*. Another possibility is sending an instruction to a flight without having previously decided any appropriate strategy. The *More* case might arise because of an additional, unforeseen performance of one subtask (e.g. one task is executed more than once). The *Different* case occurs when a different temporal relationship is introduced within the subtasks (e.g. the subtasks are performed concurrently rather than sequentially) or wrong subtasks are performed. With regard to Figure 1, an example of a different temporal relationship would be for instance if the controller sends an instruction while still deciding a strategy (thus, *Decide strategy* and *Send instruction* become concurrent tasks instead of sequential ones). The analysis of *Early* and *Late* guidewords should consider the parent task as a whole, inquiring about the impact on the system of all the situations when a too early/late performance of the parent task occurs.

Of course, with different arrangement of temporal operators, the reasoning has to be appropriately customized. It is beyond the scope of this paper to provide the reader with an exhaustive analysis of all the possible cases treated. Instead, what has to be emphasized is that, depending on the specific decomposition of a higher-level task into lower-level tasks, the application of the guidewords has to be adapted on the basis of the particular temporal relationships specified.

We note that analysis of high-level tasks can imply changes in the task structure in terms of subtasks and possible temporal relationships and this can stimulate the identification of better solutions requiring changes in such structure.

A systematic analysis can require the consideration of a high number of cases, sometimes similar among them, thus only a selection of meaningful cases can be deeply analysed in order to limit the amount of time and effort spent.

## 4. Setting up an evaluation session

The evaluation exercises should be carefully organized. It is important to involve, besides the evaluators, at least some software developers and real end-users as well. For each stakeholder, it would be better to have more than one representative to get a good level of reliance of the results achieved. The participants should be introduced to the method so that they can understand the structure of the exercise and the reasons for it. The meaning of each requested information should be precisely explained, otherwise the entire process might be distorted. For example, the participants could associate a very restrictive meaning to each column, neglecting to mention cases that, on the contrary, are interesting as well. A classic example is the interpretation of the "*protection*" word. Protection sometimes is thought only in terms of the system-side (automatic or semi-automatic safeguards provided by the system). On the contrary, we refer to a wider denotation: not only all the automatic protections offered by the system, but also all the protections provided by the other users involved in the system. For example, protections are all the situations when human agents have sufficient information to be able to realize that something is going wrong in the system.

The evaluators should have the task model available in order to have a representation helping them to systematically identify what should be analysed. In order to simplify the exercise, it is not necessary that end-users follow the details of the notation used for specifying the task model. It is sufficient that the model is used by the evaluators who analyse it to decide the questions or issues to raise. In addition, in real case studies it may happen that the task modelling phase is carried out in a time different from when the deviation analysis exercise is carried out. Thus, it can happen that the participants do not completely agree on how some parts of the task models have been modelled because, e.g. aspects that are relevant have been neglected or other objections are raised, so it is important to reach a common agreement before starting the exercise.

During the exercise the prototype (at whatever level of refinement it is), should be available and the participants should spend some time to understand its main features. It can be useful to have an audio record of the session that can be considered later on to check some parts of the discussion.

During the session, evaluators should drive the discussion raising questions following the task model and the list of deviations. Often in the discussion, it is useful to explain which presentation elements of the user interface support the logical task considered. In addition, for each task it is important to clearly identify what information is required to perform it and what the result of its performance is. If users make a mistake in understanding the information requested for each task, their answers might be inappropriate. The worst thing is that those misunderstandings might not be so evident at the beginning: if discovered ahead in the discussion they could invalidate parts of the previously achieved results, inevitably wasting time.

Tables can take some effort to be filled in and in some cases, designers may consider providing only a meaningful subset of cases able to address all the main issues. The effort required to be exhaustive is justified especially when systematic documentation of the system and design rationale are required. This is particularly important in long projects in which different people have to consider the system design at different times, for example, when discussing how to satisfy new requirements.

## 5. A case study

We have applied the method to a prototype for the air traffic control in an aerodrome. The main purpose of the system is to support data-link communications to handle aircraft movement between the runways and the gates.

Data link is a technology allowing asynchronous exchanges of digital data coded according to a predefined syntax. It complements the current communication technology that is based mainly on radio communications and observations from the window of the control tower. The support of this new technology is particularly useful in the case of bad atmospheric conditions. Thus, controllers can interact with graphical user interfaces showing the traffic within an airport in real-time (see for example, Figure 2) and messages received from pilots. In addition, in this type of application controllers can compose messages by direct manipulation techniques.

In the airport, there are two main controllers: the ''ground'', who handles the aircraft on the taxiways until they reach either the beginning of the runway (for departing aircraft) or the assigned gate (for arriving aircraft), and the ''tower'', who decides on take-off and landing of aircraft. In the considered system, the controllers have enriched flight labels available on their user interface instead of traditional paper strips


FIGURE 2. The user interface of the prototype evaluated.

commonly used in current air traffic control centres. The enriched flight labels permanently show just essential information concerning flights (*standard* mode) but they can interactively enlarge to show additional information (*selected* mode) and shrink again. They are different depending on the specific controller. For instance, consider a departing flight: the related label shown in *standard* mode on the *ground*'s user interface includes information about the flight identifier, the category and the assigned runway [see Figure 3(a)]; the flight label available to the *tower* controller for the same flight differs over the third field, which shows the expected departure direction of the flight [see Figure 3(b)]. When a label is selected, an extended set of information is shown [*selected* mode—Figure 3(c)], including the path assigned to the flight (which can be both textually and graphically represented), the current velocity of the aircraft, and other useful data. In addition, by clicking with the right mouse button on the selected label a pop-up menu with the possible commands to send is visualized [Figure 3(d)] and such commands vary depending on the particular controller.

We undertook the exercise in different work settings: at first, just the evaluators were involved, mainly to understand which kind of questions/answers the approach could have arisen. The goal of this phase was to get a preliminary overview of how to carry out the exercise, trying to identify possible key areas on which to subsequently focus the attention and critical points to be clarified.

The next phase was to perform the exercise involving also the developers. It was a useful opportunity for the evaluators to further clarify some aspects, especially those related to the prototype's user interface design choices. In fact, software developers are expected to give precise contribution about the protections existing in the system as they are the more reliable source of information with respect to it.

In addition, it was also a way of validating the results obtained during the preliminary exercise when only the evaluators were involved. For example, if some issues had not arisen or the results obtained were different from what was obtained before, it was a good way to discuss and check them and annotate key points.

Then the real and complete evaluation was performed with a multidisciplinary team in which software developers, final users (air traffic controllers in this case) and experts in user interface design were involved. Having a multidisciplinary team is useful because

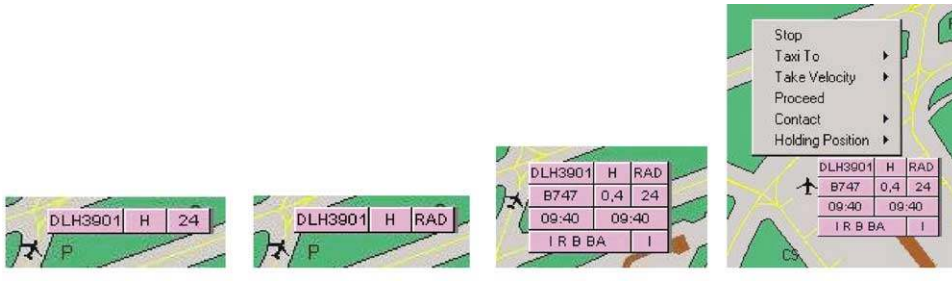


FIGURE 3. Flight information and commands depending on controller and mode. Enriched flight labels: (a) ground, standard mode; (b) tower, standard mode; (c) tower, selected mode; (d) tower, selected mode, available commands shown.

a problem is seen from all the different viewpoints and perspectives and everyone gives its own contribution based on his/her own knowledge, skills and expertise. Meeting the final users is important also because they help to understand the right importance of aspects tightly related to the domain's knowledge. In this way, some problems that could be overestimated are brought back into the correct terms and seen in the right perspective and, on the other hand, there is no risk of underestimating other aspects.

Figure 4 shows an excerpt of the task model used to carry out our evaluation exercise, whose meaning will be explained by referring to some of its tasks in the next sections. Here we would just like to recall that ||| is the concurrency operator, $\gg$ is the enabling operator, [] $\gg$ is the enabling with information exchange operator, and [] is the choice operator and that the temporal evolution is indicated from left to right (thus $T1 \gg T2$ means that $T2$ performance is enabled after $T1$).

During the exercises many interesting issues arose. In the next section, we give examples of some of them indicating the tasks and the deviations that raised the issues in order to allow the reader to understand as to how the method was useful to identify them. We also show how an analysis of the source of the problem can help identify new solutions for an improved design.

## 6. Sample issues detected

### 6.1. THE AUDIO WARNINGS PROBLEM (TASK: CHECK DEVIATION—GUIDEWORD: NONE)

The first issue is related to the controller's task of checking whether the current state of an aircraft conforms with the expected one, especially in terms of current/planned positions in the aerodrome area. We called this task "Check deviation", and if it is not

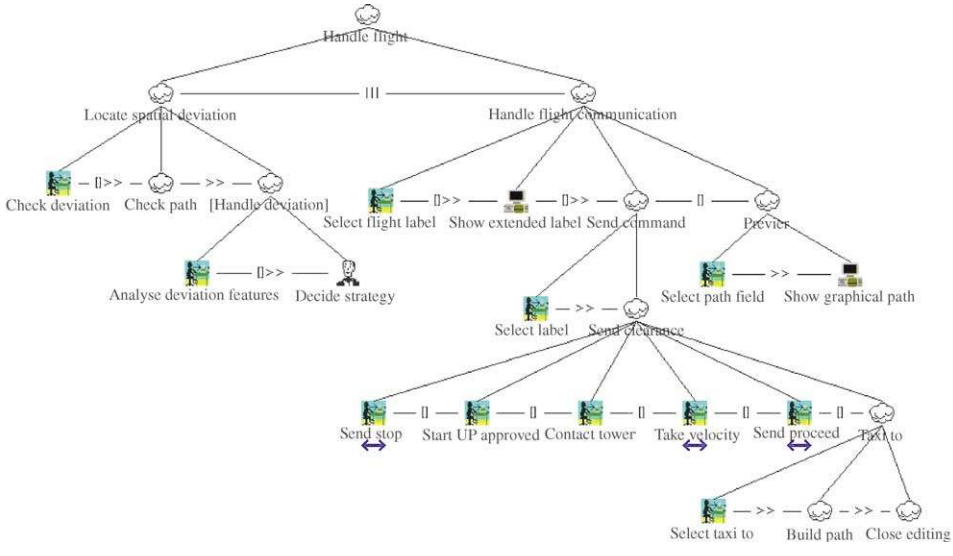


FIGURE 4. An excerpt of the task model used to perform the evaluation exercise.

performed by the controller (because of a high workload or a controller's memory fault or a system fault) the protections existing in the current prototype are operative only in truly hazardous situations, such as a runway incursion (see Figure 5). In this case, a red line connecting the aircraft involved is highlighted on the controller's user interface. However, such a safeguard is still insufficient: whenever it is really crucial to attract the controller's attention, a visual warning is not always the most appropriate technique, especially in an environment such as envisioned ATC centres, where the user must control a lot of visual tools and media even more than now. In fact, datalink is expected to play an important role in the transmission of routine messages (e.g. controller's downlinking of actual flight parameters), in order to achieve a substantial gain in voice channel availability. For this reason, most pilot–controller communications that now are carried out by means of voice in envisaged data-link equipped ATC centres will be carried out by means of datalink visual messages, leaving the *R*/*T* channel for the most time-critical ones. In this future scenario, with the controllers supposed to monitor even more visual tools than in the current system (but in quieter control rooms), audio warnings may be a useful improvement to the current prototype provided that they are sparingly used. This means that their use should be limited only to low probability


FIGURE 5. Example of runway incursion.

situations with serious consequences (e.g. a runway incursion), to reduce the probability of simultaneous occurrences of such warnings and to prevent at the same time the controllers from getting used to continuously hearing them. In addition, it is also important to allow controllers to disable such alarms (usually in high traffic situations), because as soon as the controllers become aware of the hazard, they want to concentrate on the problem without annoying (and useless) acoustic warnings still active in the control room. Controllers found such alarms particularly useful in low traffic situations when they tend to get distracted and overconfident. In fact, controllers' slips or errors do not usually occur when high workloads are put on them, except when they have little work to do and their attention level flags because they tend to trust too much the ability of the pilots to separate each other (e.g. in case of good atmospheric conditions).

It is worth noting that, when discussing the audio warnings and, more generally, the opportunity of using the aural channel for all situations where it is important to rapidly attract attention, the controllers seemed to have rather strong opinions against their use. During follow-up discussions on this point, we discovered that the controllers had just experienced a run of badly implemented audio warnings in the current system. An illustrative example was the system allowing pilots to notify controllers of some problems on board. In such a situation, the pilot must press the Transponder and a special code is inserted generating an audio alarm in the Control Tower. Once the controllers become aware of the problem, they should provide confirmation to the pilots by switching off the connected ground equipment in order to better concentrate on the problem. However, this was simply not possible because the pilot still had the device switched on, then the alarm kept on sounding in the Control Tower. In this case, the problem was not directly connected with the presence of an audio warning, but with the annoying sound that continued even after the controller understood that a hazardous situation occurred. This example highlighted that the problem was not the audio in itself but its particular (poor) implementation in the system.

### 6.2. THE NEED FOR DIFFERENT VIEWS OF THE AERODROME AREA (TASK: CHECK DEVIATION—GUIDEWORD: NO INPUT)

In the prototype, it was already possible for the controller to zoom in or out of the aerodrome map in order to have a more detailed view of some parts of the aerodrome. However, it is possible that some conflicts occur in the part of the airport currently out of the controller's view (*No input* case of the "Check Deviation" task) because the controller has just zoomed in a different part of the aerodrome map. In this situation, the possibility of having a button to return to the default view is not sufficient to make the controller aware of some hazardous situations in time. Thus, the suggestion was to have the global view of the aerodrome map (for evident safety goals) permanently displayed and, on request, to activate a separate window highlighting a specific part of the aerodrome (local view). For example, in Figure 6, one can see a possible implementation of our suggestion where there is a secondary window presented after the controller requested to zoom in on a part of the aerodrome which is rather crowded of aircraft.

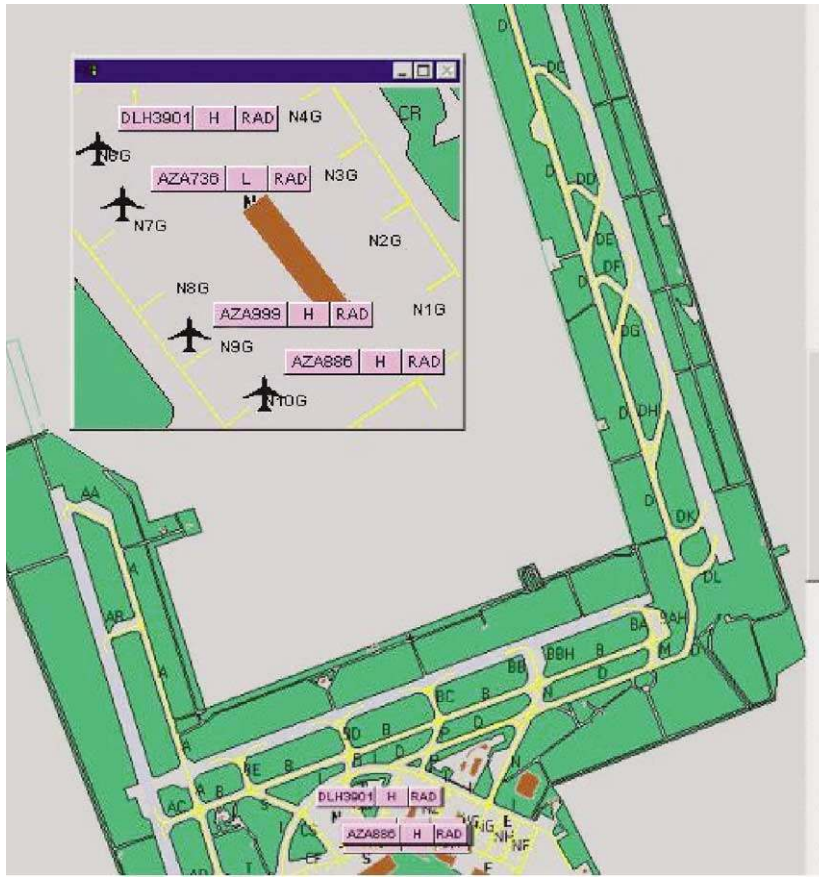


FIGURE 6. Possible solution to the zoom in problem.

### 6.3. PROVIDING LOGICAL ACKNOWLEDGEMENT OF DATALINK MESSAGES (TASK: SEND CLEARANCE—GUIDEWORD: NO PERFORMANCE)

Another deviation considered was associated to the task of *sending a clearance*, when a *No performance* deviation occurs. The causes can be either a system's fault or a distracted pilot. The protection against similar deviations in the current system can be afforded by detecting the lack of reaction from the pilot within a time interval after the controller has sent a clearance. However, this kind of protection is rather poor because it does not provide the controllers with the information about whether the clearance has really reached (or not) the pilot's system so that the controllers are able to act as soon as possible. Time being a very critical aspect in such systems, we suggested that automatic acknowledgement of the fact that the clearance has reached the pilot's system is very important in such a highly safety-critical system and should be introduced as an improvement in the prototype because waiting for the pilot's reaction could delay the overall activity too much. In addition, such feedback would improve the controller's degree of awareness of the current situation of the system. It is worth noting that such

feedback is just a logical acknowledgment (LACK) from a receiving system that the message has been successfully received and is operationally acceptable for further processing. There is no guarantee that the pilot has really read the clearance until the controller receives the WILCO message, which is the actual pilot's confirmation.

### 6.4. HIGHLIGHTING THE MOST SAFETY-CRITICAL ACTIONS (TASK: SEND STOP—GUIDEWORD: DIFFERENT PERFORMANCE)

At any time, the controller can choose among a set of clearances to send. For example, in the considered prototype by clicking with the right mouse button on the selected flight label a pop-up menu (see Figure 7) is visualized with the possible commands, depending on the controller (ground/tower) and on the flight type (arrival/departure). This kind of interaction technique shows the commands all in the same manner (items in a menu), although in case of error the consequences of activation of each command can considerably differ. For example, consider the situation when the controller detects that one aircraft is going to crash somewhere: in this case, a slip error occurring while selecting the "stop" element would be really dangerous. In the considered prototype the "stop" element is currently located in the best position as it is the first item of the menu, thus the fastest to be found with fewer possibilities to be erroneously confused with another one. Nevertheless, better highlighting of this command by appropriated graphical attributes (such as size and colour) would be even more effective.

### 6.5. THE PROBLEM OF STOPPING TAKE-OFF (TASK: SEND STOP—GUIDEWORD: ILL TIMED)

Another problem was with the task of stopping an aircraft during take-off. In the prototype, it is always possible to stop an aircraft taking-off because the related command (Stop) is always available on the user interface of the controller. However, in

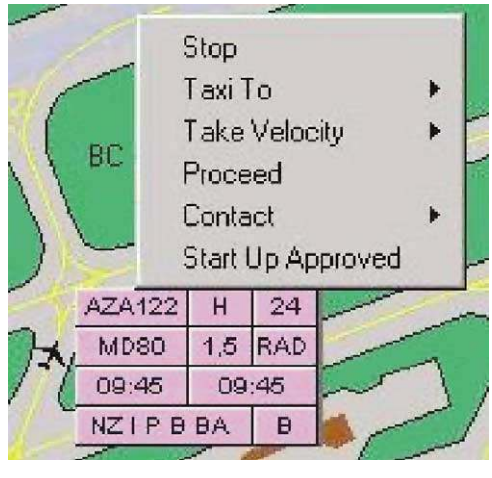


FIGURE 7. The set of possible commands available to the ground controller for a departing flight.

real situations, it is not always possible to do it because there are some situations when, for safety reasons, the take-off cannot be aborted anymore. According to this, the command "Stop" should no longer be enabled on the controller's user interface once the aircraft has started the take-off because trying to abort such an action at that moment is too late ("ill-timed" performance). Thus, in order to make the controller's activity easier it is advisable that in this menu only the items that are meaningful according to the current state of the aircraft are enabled from time to time. Thus, in the task model a different temporal relationship is required when composing this task with the tasks associated with sending other clearances (in Figure 4, we can see that the initial system task model just indicated a choice ([] operator) among these tasks, available at any time). This small example shows how this analysis can also be useful in revealing limitations in the system task model itself and provide suggestions for improving its structure.

### 6.6. IMPROVING THE DYNAMIC BUILDING OF THE PATH (TASK: BUILD PATH— DEVIATION: NO PERFORMANCE)

The deviation considered is still *No performance* and the problem was detected when controllers *graphically build a path*. As previously mentioned, within the prototype, a path to be sent to the pilot can be generated in two different ways: an automatic one, with which the controller takes advantage of the capability of the system to calculate the path according to some criteria; and a graphical, manual one which is supposed to be used by experienced controllers who need to have full control of the system in order to maximize its efficiency. The reason for this is that depending on contingent situations, neither sending the shortest path nor a "standard" path could be judged as the most appropriate solution by an experienced controller who can face this situation by building "*ad hoc*" paths. In the current prototype, there is a set of fixed points (nodes) in the aerodrome map: when the controllers have to graphically build a taxi route they make the path by selecting such points in the right order (the next point is the closest to the last selected one). If they select a wrong point then nothing happens and the path is not built. Causes of such an error can be an action slip or a wrong interpretation (the point selected is not next to the current one). In this case, the recommendation was to ease the controllers' task by allowing them to dynamically draw the path by dragging the cursor, without explicitly selecting all the intermediate points of the path.

### 6.7. PROVIDING REDUNDANT INFORMATION IN THE USER INTERFACE (TASK: SELECT FLIGHT LABEL—DEVIATION: DIFFERENT PERFORMANCE)

A further problem was detected for the *Select flight labels* task: an erroneous flight can be selected by controllers (*Different performance* deviation). In fact, the two controllers handle a different subset of aircraft depending on the current state of each aircraft, and the cause of the error could simply be the misinterpretation of such a state. In order to avoid a controller's trying to send clearance to an aircraft handled by the other controller, and thereby wasting time because in this case this action will produce no

effect (due to an automatic lock-out), the prototype provides each controller with a separate list of the flights under his/her control (see Figure 8).

However, this may not be sufficient to prevent the controller from repeatedly trying to send an instruction to an aircraft not under his/her control, especially because controllers often concentrate their attention on the aerodrome map. In fact, while speaking with the controllers we found that they prefer to have the information duplicated (redundancy) in more than one tool rather than to have a specific tool dedicated to a specific kind of information. In fact, in the latter case they might spend too much time looking for the right tool within the user interface. A possible application (and an improvement for the prototype as well) could be, for instance, to put information on the flight label of each aircraft about whether the aircraft is currently under the responsibility of the controller in question. In the considered prototype, such information was contained in a separate list that each controller has displayed on his/her user interface, so the controller has to look at it in order to understand if an aircraft is (or is not) under his/her responsibility. If this information was displayed on the flight label as well (e.g. by means of a different colour for each controller), the controllers could have a more immediate view of the traffic situation that he is currently managing. Thus, we suggested having redundant information in the associated flight label by means of different visual attributes for assigned flights.

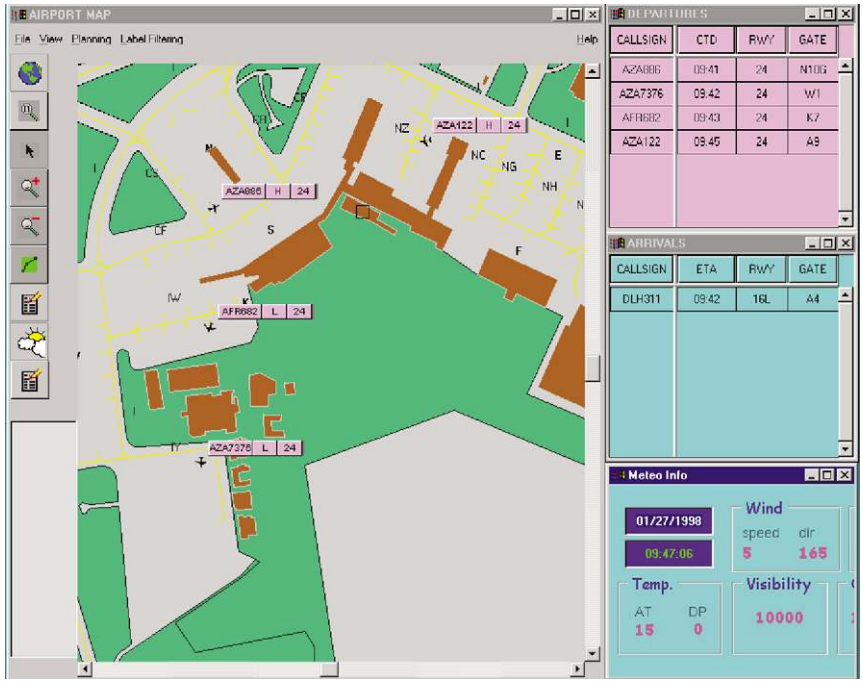


FIGURE 8. The part of the user interface indicating the controlled aircraft.

## 7. Conclusions

In this paper, we have discussed how task models and guidewords can support a systematic inspection-based usability evaluation for interactive safety-critical applications. The combination of basic and high-level tasks with the many types of deviations considered allows designers to address a broad set of issues. This can be useful in analysing how the system design supports unexpected deviations in task performance. Such an analysis is particularly important in interactive safety-critical systems where user error may even threaten human life. We have also discussed the application of this method to the evaluation of a prototype for air traffic control and showed how it can require addition of new protections or changes in the task structure.

Our experience has shown the effectiveness of the method, despite some social constraints that often occur in software development enterprises (software developers tend to defend every decision taken, users tend to digress in the teamwork, time pressure problems). The systematic analysis with multidisciplinary teams (including real users) has provided useful and thorough support in the detection of potential safety-critical interactions and has aided in improving the design.

Further work is planned to develop tool support able to help in the systematic identification of relevant issues and the recording of the results for subsequent analysis.

We gratefully acknowledge support from the European Commission and our colleagues in the (http://giove.cnuce.cnr.it/mefisto.html) MEFISTO project for useful discussions.

## References

Burns, D. J. & Pitblado, R. M. (1993). A modified HAZOP methodology for safety critical systems assessment. *Directions in Proceedings of the Safety-Critical Systems Symposium, Bristol.* Berlin: Springer.

Fields, R. E., Harrison, M. D. & Wright, P.C. (1997). *THEA: Human error analysis for requirements definition*. Technical Report YCS-97-294, Department of Computer Science, University of York. http://www.cs.york.ac.uk/~bob/papers.html

Galliers, J., Sutcliffe, A. & Minocha, S. (1999). An impact analysis method for safety-critical user interface design. *ACM Transactions on Computer–Human Interaction*, **6,** 341–369.

Hollnagel, R. (1993). *Human Reliability Analysis—Context and Control*. New York: Academic Press.

Ivory, M. & Hearst, M. (1999). *Comparing performance and usability evaluation: new methods for automated usability assessment*. Report available at http://www.cs.berkeley.edu/~ivory/research/web/papers/pe-ue.pdf.

Johnson, C. & Botting, R. (1999). Reason's model of organisational accidents in formalising accident reports. *Cognition, Technology and Work*, **1,** 107–118.

Lecerof, A. & Paternò, F. (1998). Automatic support for usability evaluation. *IEEE Transactions on Software Engineering*, **24,** 863–888.

Leveson, N.G. (1995). *Safeware: System Safety and Computers—A Guide to Preventing Accidents and Losses Caused by Technology*. Reading, MA: Addison-Wesley.

McDermid, J. A. & Pumfrey D. J. (1994). A development of hazard analysis to aid software design. *Proceedings of COMPASS '94*. New York: IEEE Press. ftp://ftp.cs.york.ac.uk/hise_reports/safety/develop.ps.Z

MOD (1996). *HAZOP studies on systems containg programmable electronics*. UK Ministry of Defence Interim Def Stan 00-58, Issue 1. Available from http://www.dstan.mod.uk/dstan_data/ix-00.htm

NIELSEN, J. (1993). *Usability Engineering*. Boston: Academic Press.

NORMAN, D. (1988). *The Psychology of Every Day Things.* New York: Basic Books.

PALANQUE, P., BASTIDE, R. & PATERNÒ, F. (1997). Formal specification as a tool for objective assessment of safety-critical interactive systems. *Proceedings INTERACT '97*, pp. 323–330. London: Chapman & Hall.

PATERNÒ, F. (1999). *Model-based Design and Evaluation of Interactive Applications*. Berlin: Springer Verlag, ISBN 1-85233-155-0.

PATERNÒ, F. & SANTORO C. (2001). User interface valuation when user errors may have safety-critical effects. *Proceedings INTERACT '2001*, pp. 270–277, Tokyo.

PATERNÒ, F., SANTORO, C. & SABBATINO, V. (2000). Using information in task models to support design of interactive safety-critical applications. *Proceedings AVI'2000*, pp. 120–127. ACM Press, May 2000, Palermo.

PUERTA, A. (1997). A model-based interface development environment, *IEEE Software*, **14,** 40–47.

REASON, J. (1990). *Human Error*. Cambridge: Cambridge University Press.

VAN DER VEER, G., LENTING, B. & BERGEVOET, B. (1996). GTA: groupware task analysis—modelling complexity. *Acta Psychologica*, **91,** 297–322.

WHARTON, C., RIEMAN, J., LEWIS, C. & POLSON, P. (1994). The cognitive walkthrough: a practitioner's guide. In J. NIELSEN & R.L. MACK Eds. *Usability Inspection Methods*. New York: John Wiley & Sons.

Paper accepted for publication by Associate Editor, Dr G Van der Veer