



The MEFISTO Project

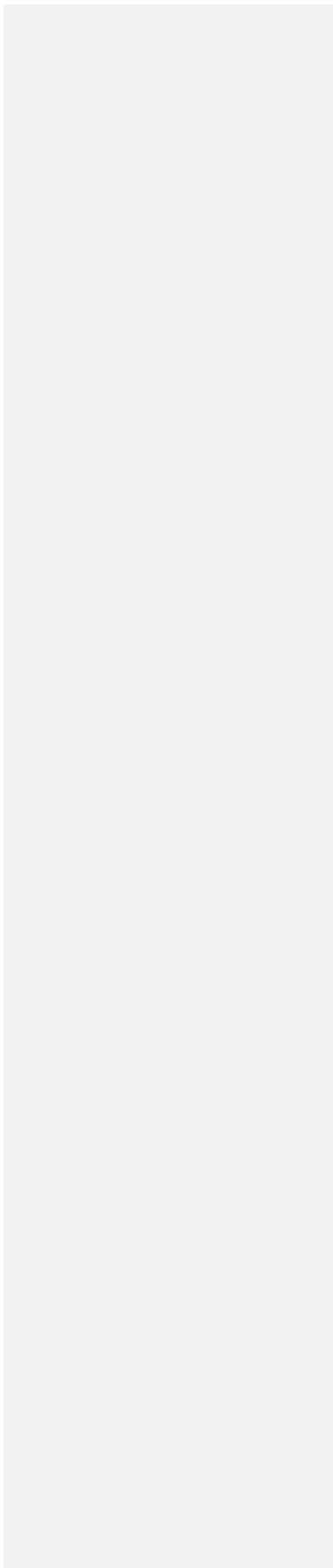
ESPRIT Reactive LTR 24963 Project

Title of Document: Task models and task-based design
Author(s): Paternò, F., Santoro, C.
Affiliation(s): CNUCE-CNR
Date of Document: Oct 7, 1999
Mefisto Project Document: D 3
Distribution: PROJECT
Keyword List: Task models, interactive safety-critical applications, user interface design
Version: For MEFISTO reviewers

MEFISTO Partners:

CNUCE, Pisa, Italy
Alenia, Rome, Italy
Dept. of Computer Science, University of York, United Kingdom
DRA, Malvern, United Kingdom
Université Toulouse 1, Toulouse, France
CENA/Sofréavia, Toulouse, France

Associates Partners: University of Siena, Italy — ENAV, Rome, Italy



Title: Task models and task-based design	Id Number: D2.1
-------------------------------------------------	------------------------

Abstract

This deliverable describes the work developed in the project concerning with the use of task models. It discusses the reasons for their use and related issues, provides some excerpts of task models related to a case study considered in the project and examples of task-based design. It also includes a discussion on how to apply model checking techniques to task models.

Table of Contents

1. INTRODUCTION.....	3
2. MOTIVATION AND APPROACH.....	4
2.1 TASK MODELLING.....	4
2.1.1 <i>Overview.....</i>	5
2.2 THE MECHA FRAMEWORK.....	5
2.2.1 <i>Criteria for comparison.....</i>	6
2.2.2 <i>Implications for individual tasks and task allocation.....</i>	6
2.2.3 <i>Implications for hazards and deviations.....</i>	7
2.2.4 <i>Co-ordination of activities and mutual awareness.....</i>	8
2.3 FROM INFORMAL SCENARIOS TO TASK MODELS.....	9
3. TASK MODELLING.....	11
3.1 AERODROME CASE STUDY: THE CURRENT SYSTEM.....	11
3.1.1 <i>Ground Controller.....</i>	12
3.1.1.1 First Level tasks.....	12
3.1.1.2 Arrange strip.....	12
3.1.1.3 Update current mental picture.....	13
3.1.1.4 Handle Traffic.....	14
3.1.2 <i>Tower Controller.....</i>	15
3.1.2.1 First Level tasks.....	15
3.1.2.2 Arrange strip.....	16
3.1.2.3 Update current mental picture.....	16
3.1.2.4 Handle Traffic.....	16
3.1.3 <i>Cooperations.....</i>	18
3.2 AERODROME CASE STUDY: THE ENVISIONED SYSTEM.....	19
3.2.1 <i>Ground Controller.....</i>	19
3.2.2 <i>Tower Controller.....</i>	20
4. FROM TASK MODEL TO DESIGN.....	23
4.1 TASK-BASED APPROACH.....	23
4.2 THE AERODROME CASE STUDY.....	23
4.2.2 <i>Ground Controller.....</i>	23
4.2.2.1 <i>Build path task.....</i>	23
4.2.2.2 <i>Build picture current state task (flight labels for ground).....</i>	32
4.2.2.3 <i>The guidance task.....</i>	39
4.2.2.4 <i>Deviations and warnings.....</i>	40
4.2.3 <i>Tower Controller.....</i>	42
4.2.3.1 <i>Correlation between different views.....</i>	42
4.2.3.2 <i>Flight labels (standard) for Tower control.....</i>	45
4.3 TOWARDS GUIDELINES FOR SAFETY-CRITICAL SYSTEMS.....	47
5. REASONING ABOUT TASK MODELS.....	49
5.1 INTEGRATING MODEL-CHECKING IN USER INTERFACE DESIGN.....	50
5.2 FROM CONCURTASKTREES TO LOTOS.....	51
5.3 USER INTERFACES PROPERTIES.....	52
5.3.1.1 <i>Warning message for time-out expired.....</i>	52
5.3.1.2 <i>Controllers' mutual awareness.....</i>	53
5.3.1.3 <i>Controllers' coordination.....</i>	54
5.3.1.4 <i>Controlled sharing.....</i>	55
6. CONCLUSIONS.....	56

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

7. REFERENCES.....57

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

1. Introduction

This deliverable describes the work developed in the project concerning the use of task models in supporting the analysis, specification and design of interactive safety-critical applications.

It is structured into four sections. Section 2 is dedicated to discuss methodological aspects. In particular, we consider motivation for task modelling and the MECHA framework whose aim is to indicate the main aspects that should be considered in designing this type of applications and in evaluating different design options. We also discuss an approach to support the development of task models from informal material in order to facilitate such a development.

The next Section is dedicated to show and discuss excerpts extracted from the models that we have developed for the aerodrome case studies (one of the two MEFISTO case studies) for sake of brevity we do not report all the work developed which is described in detail in [WP3-3].

Then, we move on how to use information contained in the task model to support the design of the user interface, still considering the aerodrome case study. The criteria that have been used can be applied also to other applications that have similar requirements so that they can be considered the core for a set of guidelines to support design of interactive safety-critical applications.

Finally, we describe how the work for applying formal reasoning to task models specified in ConcurTaskTrees has evolved.

We conclude with some general remarks and indications for future work.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

2. Motivation and Approach

The use of task models for supporting the various phases of the design cycle belongs to a more general research area, that concerning model-based approaches, aiming 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.

This Section reminds the reader the approach that we follow in task modelling, discusses the MECHA framework that highlights a set of aspects that are important in analysing and evaluating design options in an interactive-safety critical context and discusses possible support for designers in developing such task models.

2.1 Task modelling

The use of task analysis and modelling has been applied for a long time in the HCI (Human-Computer Interaction) field. However, there is still a lack of engineering approaches to the use of task models. An engineering approach should require at least:

- use of 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;
- 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;
- availability of automatic tools able to make the development and analysis of such task models more efficient.

The HCI group at CNUCE is involved in two European projects (MEFISTO and GUITARE) where task models are considered from very different perspectives. The reason for such a large difference stems from the different application areas considered (interactive safety-critical application, with particular attention to air traffic control, in MEFISTO; enterprise resource planning in GUITARE). Thus, they require two different, yet complementary, approaches: in MEFISTO we want to understand to what extent the use of rigorous techniques developed in the formal methods areas can help in the design work and we pay particular attention to the possible user deviations from the expected behaviour that can have an impact on safety whereas in GUITARE we want to obtain a set of tools able to support automatic generation of user interface from task models, objective reachable because the application domain considered in that project imposes the use of some well-defined guidelines of the user interface that limit the space of the possible solutions.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

2.1.1 Overview

The task models developed have been represented using the ConcurTaskTrees notation [P99]. The main features of this notation were already introduced in D1.2 so we just recall some basic concepts.

The purpose of a task models specified in ConcurTaskTrees is to provide a description of how the activities should be performed in order to reach the user's goals. Such activities are described at different abstraction levels in a hierarchical manner which is graphically represented in a tree-like format even though a task can appear in different places of this structure. In contrast to previous approaches, ConcurTaskTrees provides a rich set of operators with a precise semantics to describe the temporal relationships among such tasks. The notation gives also the possibility to use icons or geometrical shapes to indicate how the performance of the tasks is allocated: only user, only application, interaction, abstract (which means that there are subtasks allocated differently). Finally, for each task it is possible to provide additional information including the objects (both user interface and application objects) that are manipulated to perform it.

2.2 The MECHA Framework

The goal of the MECHA framework and the related method, that we have developed together with Bob Fields (University of York), is to support the analysis and comparison of a set of design options offering a bridge between a social view of collaborative activity (as that in the ATC domain considered in the project) and the work of designers of real systems who require systematic methods able to evaluate design choices.

More specifically, it considers different options, the current and the envisioned systems where new technology is supposed available, and the options differ in terms of *media allocation which means decisions about the access that actors in a system have to different communication media*. This implies decisions concerning how the tasks are performed and allocated, and on the choice of the artefacts and representations that are appropriate to support such tasks. These differences are highlighted also by describing scenarios that allow the analyst to focus on a specific sequence of tasks. The scenarios are introduced in the technical context of the current system and subsequently, are modified in the other cases considered.

The comparison of the design possibilities will be guided by a collection of criteria that involve usability aspects (such as task efficiency, and mutual awareness) and safety aspects (such analysis of users' deviations and their impact).

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

2.2.1 Criteria for comparison

A collection of criteria that can be used in making comparisons between the different design and task allocation options has to be identified (the selection of those criteria depends basically on the specific features of the considered domain). The aim of introducing such criteria is not to provide specific measurable parameters that can distinguish in a quantitative way between the options, but instead to suggest criteria that form a framework in which we may explore what the differences between the options are.

The reasons for this more qualitative approach is that evaluation of interactive systems is more economically carried out earlier in the development lifecycle, where re-design in response to identified problems is more feasible, as several authors (eg., [JK96]) have pointed out. Besides, the scope of MECHA is broader than a number of other HCI evaluation techniques (such as Heuristic Evaluation [N93]), that focus on specific aspects of a user interface design, since it deals also with cooperation and hazards thus allowing designers to obtain a global evaluations of the impact of using different communication technologies.

The process of comparing competing design alternatives will be a two phase one. In the first phase, it is envisaged how the performance of a sequence of tasks will be “played out” given a particular configuration of technology, task, and responsibilities. In the second phase, we begin to ask questions that allow us to make hypotheses about some of the problems with the allocation of tasks and functions that might arise. In order to carry out this second, evaluative phase, we assess the technology and its usage according to three sets of criteria. The enquiry will be scoped and contextualised by considering the tasks, actions and artefacts demanded by a particular scenario.

The MECHA method has already been applied to aspects of the en-route case study [FPST99] in collaboration with Sophie Tahmassebi (CENA), in order to evaluate how different arrangements of the media and artefacts supposed available can affect the system as a whole with regard to three main criteria:

1. Implications for individual task and task allocation
2. Hazards and deviations
3. Coordination of activities.

2.2.2 Implications for individual tasks and task allocation

We can identify three main types of difference between the current system and “augmented” systems where data-link is available:

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

- *Change of task allocation between the human and the machine:* for example, in datalink environment, the update of the ground system (containing flight information) is no longer performed manually by the controller, but in an automatic way by the system.
- *Change of task allocation between human operators:* because both controllers can communicate with pilots as well, by means of datalink functionality.
- *Change of objects manipulated by task and change of representations used to support tasks:* for example, in the new system the information contained in flight paper strips can be electronically provided.

Furthermore, a number of factors relating to the way tasks are carried out must be considered when making comparisons between the design options. For instance, technological changes can have the effect of transforming interaction tasks into vigilance and monitoring tasks at which people are often less effective (cf. [H88]). Similarly, design and task allocation decisions can have a significant impact on the workload of individuals and the range of responses to workload demands that are available to participants.

2.2.3 Implications for hazards and deviations

Our collection of criteria are particularly important for interactive safety-critical systems, and involve studying the different failure and hazard characteristics of different design options. We use an inspection technique to go systematically through the actions that are required from participants, and consider ways in which failures might arise during a scenario, what the effect of failures might be, and what safeguards and defences exist in the system. Since an objective of the current work is to explore the impact of different arrangements of communication technology, a special emphasis will be placed on communicative actions.

The particular questions we seek answers to are: what are the potential hazards that can arise as consequences of deviations, failures in communication, or erroneous actions in the scenario? Are there factors that tend to encourage miscommunication, erroneous action, or faulty assessments? What recommendations concerning the user interface design can be provided to mitigate possible hazardous states and their effects?

This type of analysis will be performed with the help of *guidewords* (see [MOD96], [L97], [BP93] for related techniques). A *guideword* is a word or phrase that expresses and defines a specific type of *deviation*. Guidewords have been found to be a useful tool to stimulate 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. We have found it useful to investigate the deviations associated with the following guidewords:

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

- *None*, the task has not been performed or it has been performed but it has not produced any result;
- *Other than*, the task has been performed using the wrong data or producing wrong data;
- *Ill-timed*, the task has been performed at the wrong time.

In an analysis, these guidewords can be further refined. For example, *Other than* could be further refined into *Less*, *More*, or *Different* indicating situations where less, more or different information has been used in the tasks. Likewise *Ill-timed* can be refined into *Early* or *Late* implying that the task is performed too early or too late.

The basic idea is that for each option we consider the main tasks and the possible deviations that can occur in the performance of the task. Interpreting the guidewords in relation to a task allows the analyst systematically to generate ways the task could potentially go wrong, as a starting point for further discussion and investigation. This analysis may generate suggestions for how to guard against such deviations and recommendations about user interface designs that might either reduce the likelihood of the deviation, or support detection and recovery.

This analysis of possible deviations can give better results if applied by multidisciplinary teams, such as the MEFISTO team, that involve software developers, user interface designers, application domain experts and end users.

2.2.4 Co-ordination of activities and mutual awareness

The concept of “articulation work” and the means by which the activities of individuals are coordinated are a complex topic. For current purposes, we focus on one aspect, namely, the way in which technology changes (such as the introduction of datalink) have an impact on the kinds of coordination that are necessary and possible. More specifically, the two questions we will be asking about the design alternatives are:

- What coordinations are needed so that the tasks of the two controllers are brought into step? The answer to this question will typically be dependent on the particular roles, responsibilities, and tasks of the individuals involved that can be represented in ConcurTaskTrees.
- How such coordinations will be supported by the available mechanisms? The answers to this question are likely to be dependent on the detail of the technologies and artefacts that mediate the tasks of individuals and communications between them.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

2.3 From informal scenarios to task models

In the MECHA framework we have seen the main design dimensions when interactive safety-critical applications are considered. We have seen that task models play an important role. One of the main problem in the use of task models, especially for people who are not expert in this activity is how to start their development starting from scratch. People may get confused in trying to identify the tasks that should be included in the model and their relationships.

To support this initial phase it can be useful to consider that when approaching the design of a new application or the re-design of an existing application, designers have often a lot of informal information available: documentation concerning existing applications, notes from meetings with users, requirements provided by customers, and so on. They have to refine this material to identify the task structure underlying the existing application to analyse or that corresponding to the new application to design.

Scenarios are a well known technique in the HCI field [S95] often used during the initial informal analysis phase. They provide informal descriptions of a specific use in a specific context of an application. A careful identification of a meaningful scenario allows designers to obtain a description of most of the activities that should be considered in a task model. The main difference between a task model and a scenario is that a scenario indicates only one specific sequence of occurrences of the possible activities while the task model should indicate all the possible activities and the related temporal relationships.

Given their limited scope and the simple structure underlying them, it is sometimes easier to start thinking in terms of specific scenarios rather than more general models of activities. Then, there is the need to obtain task models to reach a more general and precise description of the possible activities. To this end some tool support can be provided [PM99].

We start with an informal description of a scenario. The scenario should be selected so as to include performance of most of the main activities involved by the application considered. It can be either the description of a specific use of an existing system or an envisioned use of a new application to design depending on what the designer's goal is. Next the designer can load such a description in the environment provided by our tool and select the roles (1 in Figure 2.1), words related to activities (such as detection of conflict, paper strip's update, sends clearance, see for example 3 in Figure 2.1) and add them to the list of tasks. The names of such tasks can be edited in order to make them more general. The designer can also interactively indicate how to allocate the performance of the task: to the user (if only internal cognitive actions are required), to the application, to a user interaction (if the performance consists in user interactions with some device). This is specified by selecting the icon associated with the task allocation chosen. In the scenario's description it is also possible to select the objects (5 in figure 2.1) and indicate to what tasks they are associated. One task can manipulate multiple

objects during its performance and one object can be manipulated by multiple tasks. In this way designers have an environment allowing them to rapidly identify tasks, objects and their relationships.

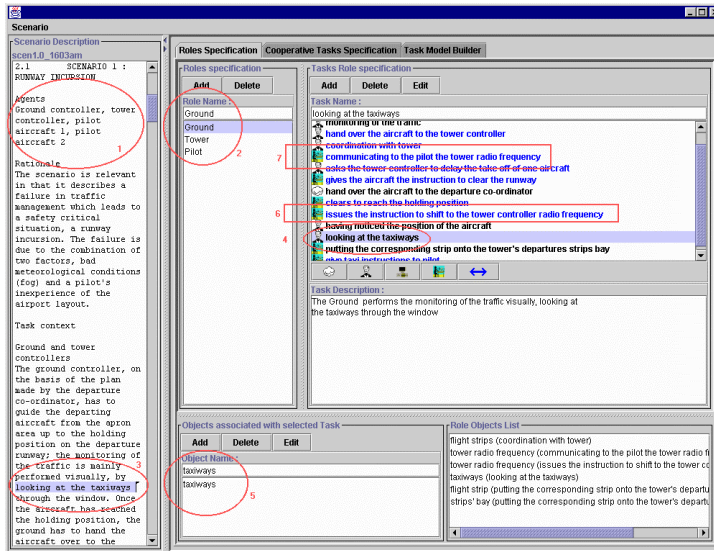


Figure 2.1: Moving from informal scenario to task model.

The next step is to identify the structure of the task model. We split this activity into two steps: identify the hierarchical structure among tasks and define their temporal relationships. The input for this phase is the list of tasks identified with the scenario support. This list is not definitive. It can be further modified, for example to add new tasks whose purpose is to logically group a set of identified tasks that are semantically connected and share some temporal relationship.

In our tool designers can activate an environment which has the list of tasks identified as input and allows designers to indicate a logical hierarchy among such tasks: from the list of identified tasks on the left side we can select a task and indicate its parent task on the right side.

We thus obtain a hierarchical task model that can be further edited by the existing ConcurTaskTrees editor (<http://girove.cnuce.cnr.it/ctte.html>). The difference is that now designers have not to start by scratch but they have available the hierarchy of tasks and most of the objects manipulated by such tasks have already been identified. Thus only the temporal relationships among the tasks have to be specified with the support of this editor.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

3. Task Modelling

This Section describes some excerpts of the task models for the aerodrome case study specified in ConcurTaskTrees. More specifically, the task models of the current system are completely described in order to introduce the reader to problems and issues connected to the considered environment, whereas for the task models of the envisioned system the description has been kept with at a higher level because lower tasks will be introduced and discussed later on when we explain how to derive information from the task model for the design of the user interface. We provide examples taken from the analysis of both the current and the envisioned system. In both cases we consider two different roles: the ground and the tower controller. The ground controller is mainly in charge of handling the air traffic within the airport whereas the tower handles the landing and take-off phases.

Eliminato: 1

3.1 Aerodrome Case Study: The Current System

The development of the task models of the current system has been carried out with the support of information gathered in different ways. We have visited various times the control tower of the Fiumicino airport in Rome (photo in Figure 3.1 was taken during one of these visits), followed by interviews of controllers. We have had various meeting with the Alenia team that has a long experience in development applications for air traffic control and it is involved in the development of the new prototype.



Figure 3.1: The Fiumicino control tower.

3.1.1 Ground Controller

3.1.1.1 First Level tasks

At the highest level of the GND's task model three main tasks are recognisable: *Arrange strip*, *Update current mental picture* and *Handle traffic* (see Figure 3.2): they respectively refer to tasks about handling the paper strips, maintaining and updating the picture of the current/future traffic situation and driving the traffic under his/her responsibility.

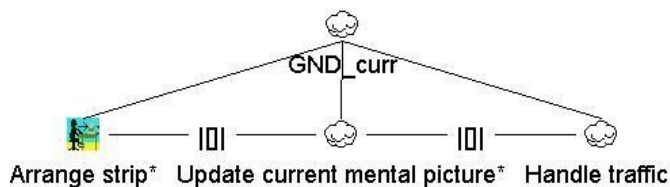


Figure 3.2: First Level Tasks - (Current System/Ground controller)

For each of these tasks we provide a separate description in a dedicate paragraph. Thus, by proceeding from left to right in the analysis of the task model of Figure 3.2, the first task to examine is *Arrange strip*.

3.1.1.2 Arrange strip

The *Arrange strip* task deals with the Ground's activity of continuously receiving paper flight strips of departing flights (*Arrange dep strip*) and arriving flights (*Arrange arr strip*), putting them in the right position into the respective strip bay (see Figure 3.3).

As the Ground controller receives the strips of arriving flights from the Tower controller, *Take arr strip* task is marked as a task part of a cooperative task (double arrow below the task name), whereas no co-operation is indicated in the *Take dep strip* task as s/he receives them from the Apron controller—whose activities we are not interested to model.

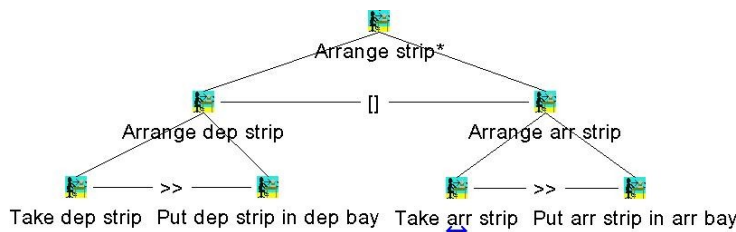


Figure 3.3: *Arrange strip* task (Ground controller)

3.1.1.3 Update current mental picture

Concurrently with the previous task, controllers have to build and continuously refresh their mental picture of the traffic situation (*Update current mental picture*, see Figure 3.4) collecting and monitoring traffic data (*Collect data*) gathered by looking at taxiways out of the control tower window or on the radar if present (*Check taxiways*, *Check radar* tasks). This information is used in order to check (*Check conformance*) if the *current* traffic situation conforms with the *planned* situation reported in the flight strips (*Read strip*).

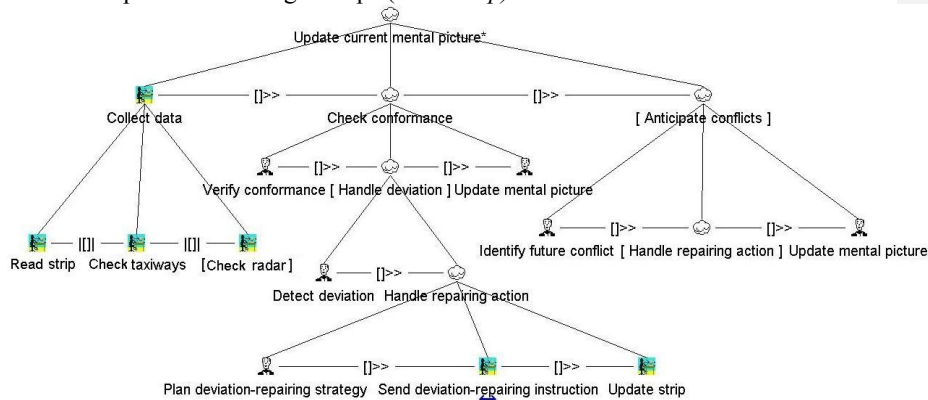


Figure 3.4: *Update current mental picture* task - (Ground controller)

If there is no conformance (*Handle deviation*), it means that some deviation has been detected (*Detect deviation*) requiring that repairing actions are undertaken by the controller (*Handle repairing action*) who has to think about a possible solution, to

send an instruction to a pilot, and to update consequently his/her mental picture and the flight strip.

If conformance exists, the (optional) *Handle deviation* task is not performed, however, the GND mentally records the gathered snapshot refreshing the previous mental picture with this information (*Update mental picture*). The next step is to analyse whether —according to the current and the expected situation of traffic— conflicts are possible to happen (*Anticipate conflicts*), deciding if some actions are necessary (or not). Anyhow, the controller keeps in mind the collected information refreshing his/her picture of current state.

3.1.1.4 Handle Traffic

The Ground has to manage both the path's requests from pilots (*Manage traffic* task) and the hazardous situations that could occur in the system (*Control hazardous situations* task).

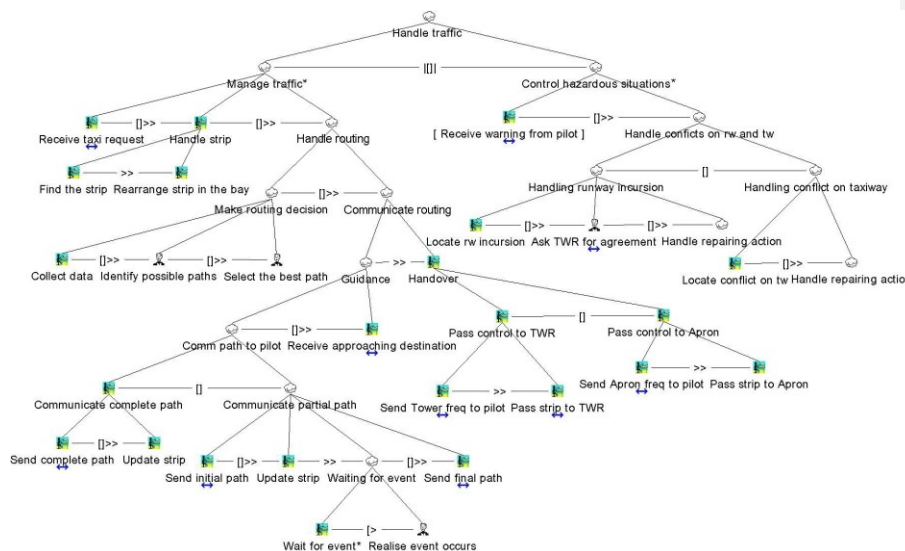


Figure 3.5: *Handle traffic* task - (Ground controller)

As far as it concerns the first task, once the controller receives the pilot's request by listening to the radio (*Receive taxi request*), the GND searches (*Handle strip*) the associated strip (possibly modifying its location in the bay) in order to be in a position of deciding how to answer to the pilot.

In order to make a decision on how to manage the received request in the current situation of traffic (*Make routing decision*) the GND has to collect appropriate data (*Collect data*) to identify at first the possible paths (*Identify possible paths*) and then selecting the best path to communicate to the pilot (*Select the best path*).

The action of actually sending the clearance to the pilot (*Communicate routing task*) that is usually the task of communicating path to pilot (*Comm path to pilot*), can be carried out by the Ground controller in two different ways, depending on the current situation of traffic:

- sending once the entire path to pilot (*Communicate complete path*), or
- giving at first a partial path and afterwards —when some conditions are verified— completing the path (*Communicate partial path*).

When the controller receives the pilot’s message acknowledging that the destination has been reached (*Receive approaching destination*), s/he performs the hand-over with the next controller (*Handover task*), that is to pass the control of the a/c either to the Apron controller or to the Tower controller.

As far as it concerns the *Control hazardous situations* task, the GND can receive warning messages from pilot (*Receive warning from pilot*) or s/he can realise by himself that some conflicts are occurring on runways or taxiways : in the first case s/he has to ask TWR for agreement (being runways under the responsibility of TWR) before sending repairing actions, in the latter case s/he autonomously sends emergency instructions to pilot.

3.1.2 Tower Controller

3.1.2.1 First Level tasks

If we analyse the task model of the Tower controller we note that it presents relevant similarities with the Ground’s despite the fact that they manage aircraft in two different phases (from both temporal and spatial viewpoints). The reason is that they perform mainly the same general tasks (arrange paper strips, communicate with pilots, supervise the system, solve possible hazardous situations, ...), although the specific objects they manage are quite different. For example, both controllers receive requests from pilots however, while the requests for the Ground are requests to have a path to get to the holding position from the departure’s gate (or from the end of the runway to the arrival’s gate), the requests directed to the Tower are requests to get the clearance to take-off or to approach the airport.

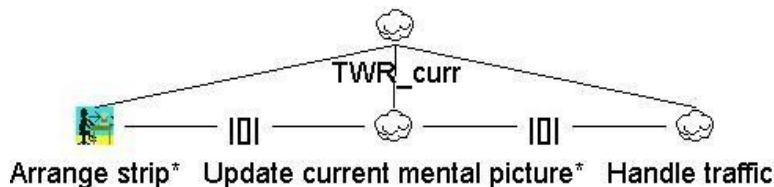


Figure 3.6: First level tasks - (Tower controller)

3.1.2.2 Arrange strip

The Tower’s *Arrange strip* task is similar to the Ground’s respective task. The only difference is that now the task part of a cooperative task is *Take dep strip* because the Tower receives the strip of departing flights from the Ground controller, whereas s/he receives the other ones from the Arrival Co-ordinator —whose activities we are not interested to model, then no link to a co-operative task is specified.

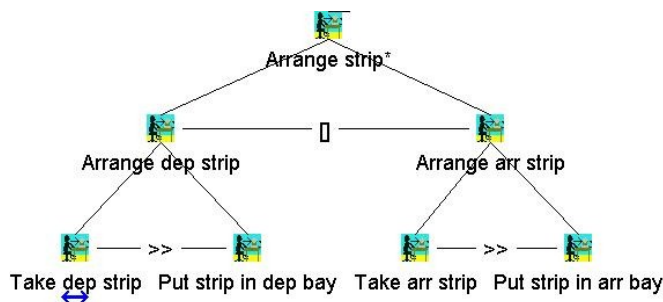


Figure 3.7: *Arrange strip* task - (Tower controller)

3.1.2.3 Update current mental picture

The Tower’s *Update current mental picture* task is quite similar to the Ground’s respective, keeping in mind that, differently from the Ground controller (whose responsibility is mainly on the taxiways), the Tower has to build the picture of the runways’ current/future state.

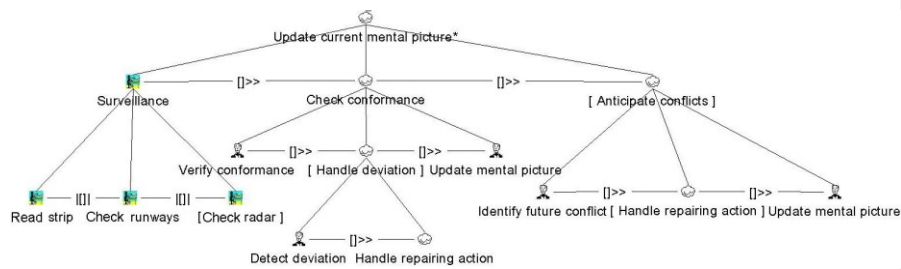


Figure 3.8: *Update current mental picture* task - (Tower controller)

3.1.2.4 Handle Traffic

Again, *Handle Traffic* task is along the lines of the respective Ground’s task, with the most evident differences at lower levels.

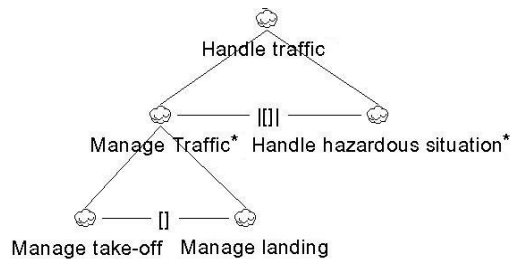


Figure 3.9a: The initial part of the *Handle Traffic* task.

It is decomposed into the *Manage Traffic* task and the *Handle hazardous situation* task. The first one concerns with driving a departing/arriving flight (respectively *Manage take-off* and *Manage landing* tasks), the second one is about the problem of coping with some hazardous situation (*Handle hazardous situation*).

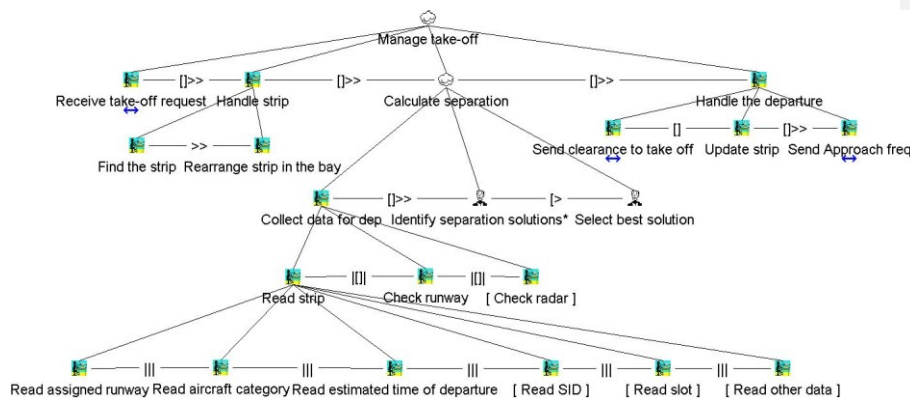


Figure 3.9b: *Manage take-off* task.

More specifically, when a departure has to be managed the controller receives the request from the pilot (*Receive take-off request*) listening to the radio and searches the associated strip in the strips' rack (*Handle strip*). Once s/he has the strip, s/he starts to collect data (*Collect data for dep* task) about the flight from all the available sources of information (looking out at the runways, at the strip, and on the radar) and starts to identify possible solutions in order to manage the request and to ensure that minimal separation is maintained between flights. When s/he finds the best solution s/he communicate it to the departing pilot (*Handle the departure*), sending the clearance to take-off (*Send clearance to take off*) and passing to the pilot the frequency to contact the Approach controller (*Send Approach freq*).

On the other hand, if s/he has to manage an arriving flight (*Manage landing*, Figure 3.9c) after s/he has received the request to land and s/he has found the strip, s/he is

in a position to decide how to answer to the request of the pilot (*Handle the arrival*), which is sending to the pilot the clearance to land (*Clear to land*) or sending to the pilot the order of “go around”, which means that the pilot has to wait for landing safely on the assigned runway.

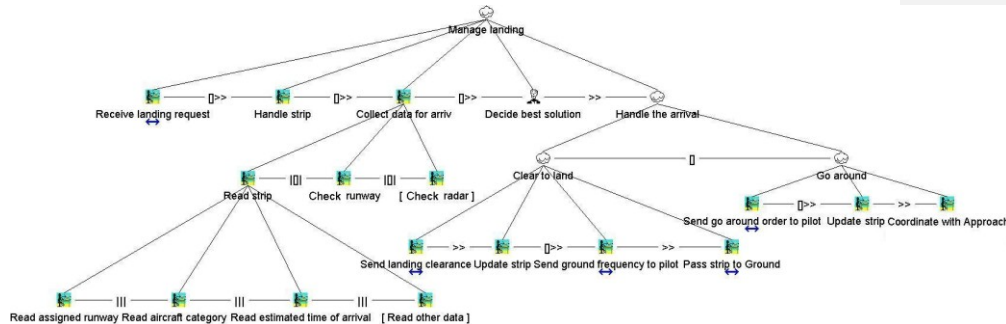


Figure 3.9c: Manage landing task.

3.1.3 Cooperations

ConcurTaskTrees gives designers also the possibility to model explicitly cooperations. A cooperative task is task that implies actions from two or more users. In figure 3.10a there is an excerpt of the specification concerning the communication between the ground controller and a pilot to identify the path to follow. As you can see the leaves of the cooperative part are tasks performed by one user (whose role and related cardinality are explicitly indicated).

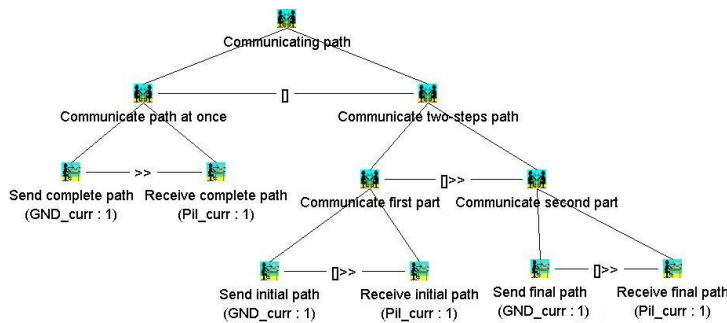


Figure 3.10a: Communicating path cooperative task.

The leaves of cooperative tasks are tasks performed by a single user that appear also in the task model of the related role. In the role-related task model such tasks are annotated by a double-arrow to highlight that they are a point of connection with the cooperative part.

Before analysing the task models of the envisioned system just to give an idea of the current level of refinement in specification work that has been carried out for

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

the aerodrome case study, in the Figure 3.10b we show a summary of statistical information (number of tasks, number of occurrences for each operator, ...) for the task model of the two controllers the cooperations that occur in the current system (and, for completeness, also for the role of the pilot).

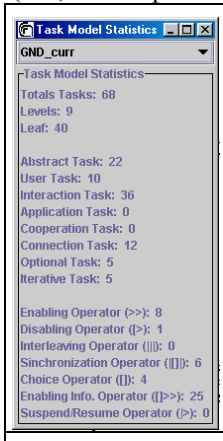
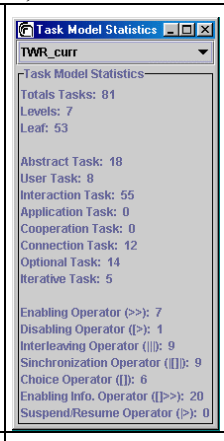
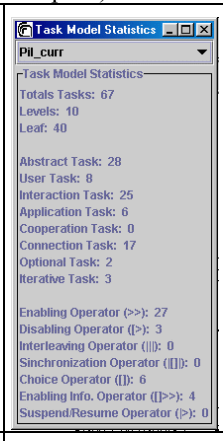
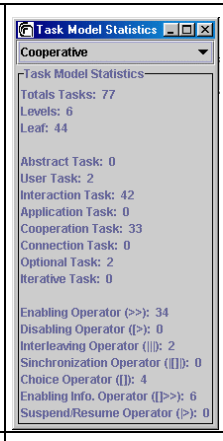
 <p>GND_curr Task Model Statistics Totals Tasks: 68 Levels: 9 Leaf: 40 Abstract Task: 22 User Task: 10 Interaction Task: 36 Application Task: 0 Cooperation Task: 0 Connection Task: 12 Optional Task: 5 Iterative Task: 5 Enabling Operator (>): 8 Disabling Operator (>): 1 Interleaving Operator (): 0 Synchronization Operator (): 6 Choice Operator (): 4 Enabling Info. Operator (>): 25 Suspend/Resume Operator (>): 0</p>	 <p>TWR_curr Task Model Statistics Totals Tasks: 81 Levels: 7 Leaf: 53 Abstract Task: 18 User Task: 8 Interaction Task: 55 Application Task: 0 Cooperation Task: 0 Connection Task: 12 Optional Task: 14 Iterative Task: 5 Enabling Operator (>): 7 Disabling Operator (>): 1 Interleaving Operator (): 9 Synchronization Operator (): 9 Choice Operator (): 6 Enabling Info. Operator (>): 20 Suspend/Resume Operator (>): 0</p>	 <p>Pil_curr Task Model Statistics Totals Tasks: 67 Levels: 10 Leaf: 40 Abstract Task: 28 User Task: 8 Interaction Task: 25 Application Task: 6 Cooperation Task: 0 Connection Task: 17 Optional Task: 2 Iterative Task: 3 Enabling Operator (>): 27 Disabling Operator (>): 3 Interleaving Operator (): 0 Synchronization Operator (): 0 Choice Operator (): 6 Enabling Info. Operator (>): 4 Suspend/Resume Operator (>): 0</p>	 <p>Cooperative Task Model Statistics Totals Tasks: 77 Levels: 6 Leaf: 44 Abstract Task: 0 User Task: 2 Interaction Task: 42 Application Task: 0 Cooperation Task: 33 Connection Task: 0 Optional Task: 2 Iterative Task: 0 Enabling Operator (>): 34 Disabling Operator (>): 0 Interleaving Operator (): 2 Synchronization Operator (): 0 Choice Operator (): 4 Enabling Info. Operator (>): 6 Suspend/Resume Operator (>): 0</p>
Ground controller	Tower controller	Pilot	Cooperative tree

Figure 3.10b: Statistic information about task models in the current system.

3.2 Aerodrome Case Study: The Envisioned System

About the envisioned system, in the following sections we give the description of the task models of the envisioned system for the highest levels, leaving the analysis of more detailed specifications to the next section where some excerpts of task models will be used to derive suggestions for the design of the user interface. The basic requirements for the envisioned system are the use of data-link to support the communication with aircraft and the use of enriched flight labels to provide information concerning each flight.

3.2.1 Ground Controller

In the envisioned system, the Ground controller has to build and maintain up-to-date the picture of the current state (*Build picture current state*), looking at the taxiways out of the Control Tower window and looking at the tools supposed available on the radar display in the new environment (enriched flight labels, aerodrome map). The task of both verifying conformance between the current situation of the traffic and the future situation and to anticipate possible conflicts in the near future is mainly up to the system in the forms of automatic tools which signal to the controller when some deviation is detected or is near to happen.

As far as it concerns managing properly the traffic (*Handle traffic*), the controller receives continuously the data-link requests that arrive and, from time to time decides which one is the (taxi) request that has to be managed. Once s/he selects it, s/he has to build and send a path to the pilot (*Build path*), driving and controlling the flight until the pilot gets to the destination point (e.g. holding position for departing pilots) and finally the controller passes the responsibility of the flight to the next controller (*Handover*). The *Control hazardous situation* task manages all the situations when some deviation has been detected or some conflict occurs both on the taxiways and the runway (eg: runway incursion): in the following sections we show how to manage a space-based deviation, expanding the associated subtask (*Handle spatial deviation*).

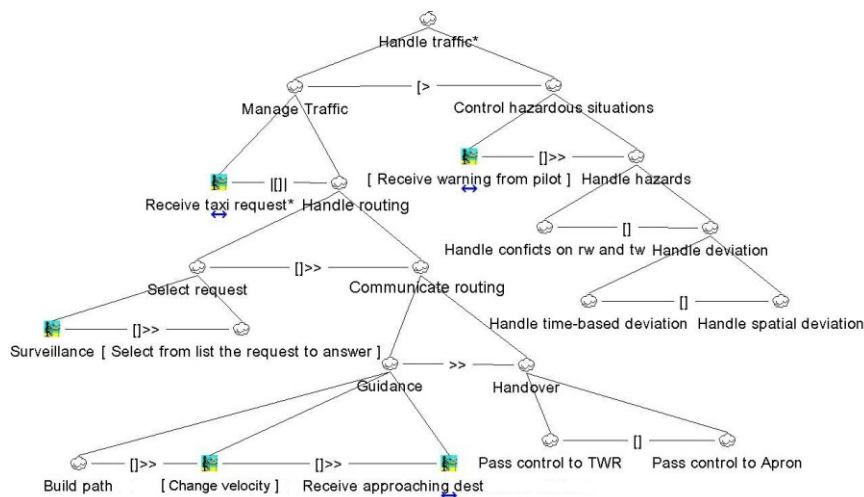


Figure 3.11: The *Handle Traffic* task in the new system.

3.2.2 Tower Controller

In the envisioned system, the Tower controller has to build and maintain an up-to-date picture of the current state (*Build picture current state*), looking at the runways out of the Control Tower window and looking at the tools supposed available on the radar display in the new environment (enriched flight labels, aerodrome map). Also in this case the task of both verifying conformance between the current situation of the traffic and the future situation and to anticipate possible conflicts in the near future is mainly up to the system in the forms of automatic tools which signal to the controller when some deviation is detected or is near to happen.

As far as it concerns managing properly the traffic (*Handle Traffic*), the controller receives continuously (look at the iterative operator on the *Receive pilot request*

task) the data-link requests that arrive and, from time to time decides which one is the request that has to be managed. If it is a request from a departing pilot (*Handle the departure*), the controller has to collect data for the departing flight and find the best answer for the pilot, alternatively, if the request is from a pilot who asks for landing, (*Handle the arrival*) the controller has to decide if there are all the safe conditions necessary to make the flight land and then passes the responsibility of the flight to the ground controller, who drives the flight to the apron. Finally, the *Handle hazardous situation* task manages all the situations when some deviation has been detected or some conflict occurs on the runway (eg: conflicts, emergencies).

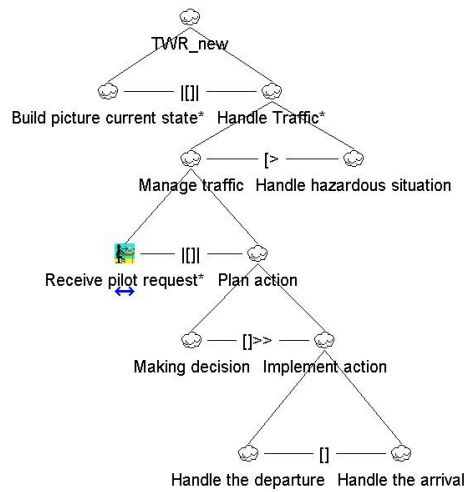


Figure 3.12: Main activities of tower controller in the new system.

As before, we show a summary about statistic data on task models of the envisioned system (see Figure 3.13 below).

Title: Task models and task-based design

Id Number: D 2.1

GND_new	Twr_new	PIL_new	Cooperative
<p>Task Model Statistics</p> <p>Totals Tasks: 89 Levels: 12 Leaf: 49</p> <p>Abstract Task: 43 User Task: 6 Interaction Task: 28 Application Task: 12 Cooperation Task: 0 Connection Task: 10 Optional Task: 11 Iterative Task: 5</p> <p>Enabling Operator (>): 6 Disabling Operator (!>): 3 Interleaving Operator (!): 3 Synchronization Operator (!): 6 Choice Operator (!): 7 Enabling Info. Operator (! >): 26 Suspend/Resume Operator (!>): 0</p>	<p>Task Model Statistics</p> <p>Totals Tasks: 72 Levels: 9 Leaf: 43</p> <p>Abstract Task: 35 User Task: 3 Interaction Task: 30 Application Task: 4 Cooperation Task: 0 Connection Task: 11 Optional Task: 8 Iterative Task: 3</p> <p>Enabling Operator (>): 12 Disabling Operator (!>): 1 Interleaving Operator (!): 3 Synchronization Operator (!): 4 Choice Operator (!): 5 Enabling Info. Operator (! >): 17 Suspend/Resume Operator (!>): 0</p>	<p>Task Model Statistics</p> <p>Totals Tasks: 63 Levels: 8 Leaf: 38</p> <p>Abstract Task: 25 User Task: 8 Interaction Task: 25 Application Task: 5 Cooperation Task: 0 Connection Task: 18 Optional Task: 2 Iterative Task: 2</p> <p>Enabling Operator (>): 29 Disabling Operator (!>): 2 Interleaving Operator (!): 1 Synchronization Operator (!): 0 Choice Operator (!): 5 Enabling Info. Operator (! >): 0 Suspend/Resume Operator (!>): 0</p>	<p>Task Model Statistics</p> <p>Totals Tasks: 73 Levels: 5 Leaf: 44</p> <p>Abstract Task: 0 User Task: 2 Interaction Task: 40 Application Task: 0 Cooperation Task: 31 Connection Task: 0 Optional Task: 5 Iterative Task: 0</p> <p>Enabling Operator (>): 39 Disabling Operator (!>): 0 Interleaving Operator (!): 3 Synchronization Operator (!): 0 Choice Operator (!): 2 Enabling Info. Operator (! >): 0 Suspend/Resume Operator (!>): 0</p>
Ground controller	Tower controller	Pilot	Cooperative tree

Figure 3.13: Statistic information about task models in the envisioned system

4. From Task Model to Design

4.1 Task-based Approach

The approach is based on a top-down visit of the task model and for each level it considers two main aspects:

- *the analysis of the operators between tasks*, which gives information about the temporal constraints to implement in the design of the user interface in terms of relationships between the different dialogues associated to the tasks;
- *the analysis of the task*, in terms of type and category of task, type and cardinality of data manipulated by task, which provide information useful to design the user interface;

In the previous sections we show the specification of the task models in both the current and the new system, giving for the new system the specification of only the highest levels. In the following sections some excerpts of the task models in the new system will be resumed and analysed more deeply in order to highlight how to derive information from them for the design of the user interface.

4.2 The Aerodrome Case Study

4.2.2 Ground Controller

The first task that we analyse is the *Build path* task of the Ground controller, which describes all the activities necessary to answer to a taxi request from a pilot.

4.2.2.1 *Build path* task

(a) General description

At its highest level the *Build Path* task is decomposed into two main subtasks (see Figure 4.1):

- *Build path with automatic suggestions*, which describes the activities performed by the controller to build a suitable path exploiting automatic tools (in this case the system really drives the controller to get the best solution showing the set of possible solutions);
- *Build path without automatic suggestions*, when the controller is more self-confident and builds directly the path (in this case the role of the system is mainly to support controllers by helping their decision-making process).

As far as it concerns the subtask *Build path with automatic suggestions*, it is composed by an iterative subtask (*Ordering paths*) that describes all the activities

necessary to get the set of possible solutions ordered by some specified criterion: the controller selects the parameter by which s/he wants the solutions are ordered (for example, if s/he selects the parameter “Length”, the system will show to the controller all the possible paths ordered by this parameter, at first the paths that have the minimum length and then the others in an increasing order) and the system calculates the paths ordered by this parameter and shows the ordered list to the controller (*Show ordered paths*). This process can repeat more than one time (the task is iterative) because the controller can change mind and select another criterion, however finally the controller chooses the “best” path and after having (optionally) activated a preview of all the information about this path, s/he is able to actually send the path or cancel the whole process and restart it over again (in the task model it is modelled by a recursive instantiation of the *Build path* task).

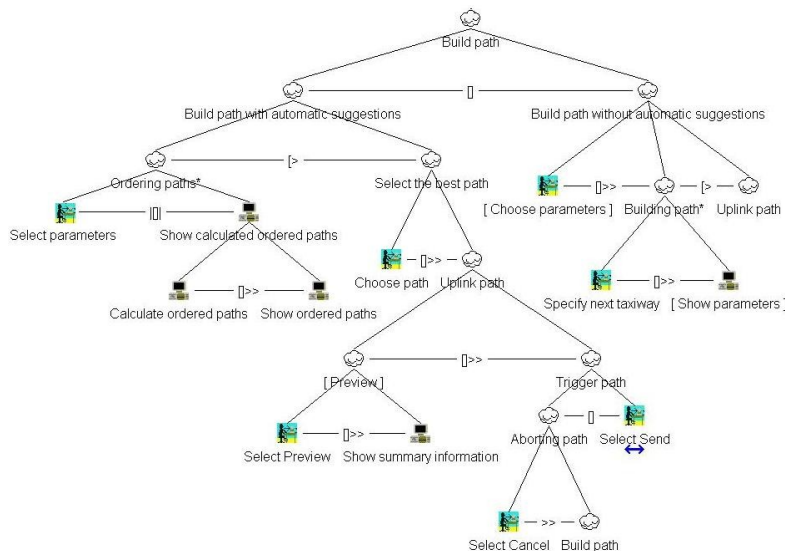


Figure 4.1: The *Build Path* task model

The subtask *Build path without automatic suggestions* allows the controller to build the path specifying directly the taxiways and optionally specifying whether and which parameter s/he is interested to know as s/he gradually builds the path. For example, if the controller has selected the parameter “Length”, as s/he gradually selects the taxiways the system shows from time to time how long the distance is, in order to help the controller to decide about the appropriateness of the path.

(b) The analysis

- Analysis of the operators between tasks

In this part of the analysis we focus mainly on the operators, thus in Figure 10 we give a schematic representation of the *Build path* task model, where the expansion of some subtasks has been neglected (a grey triangle has been put instead of them) and —for sake of brevity— task names have been substituted by letters. The aimed goal is to focus on a (portion of) task model little enough not to bore the reader with too many details but sufficient to explain which information the operators give to the designer. In fact as you can see from the picture, almost all the CTT operators appear in the selected task model (Enabling, Disabling, Interleaving, Option, Iteration, Choice) so the discussion could be easily generalised and re-applied to other task models. The indications in the right part of the below figure allow to derive the “simplified” task model shown in the left part of the picture from the task model in Figure 4.2.

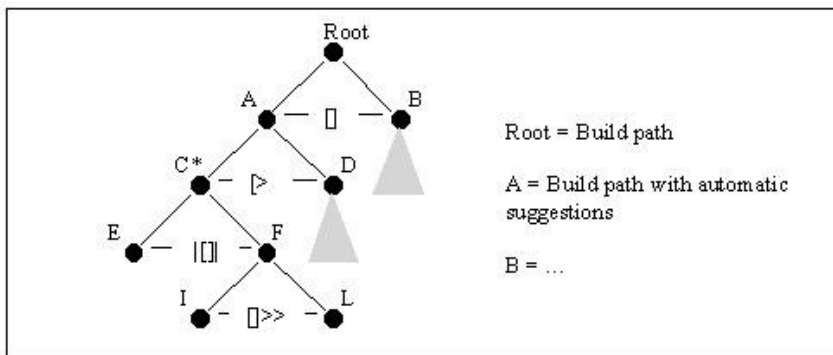


Figure 4.2: A “simplified” representation of *Build path* task model

Referring to this “simplified” task model, the Choice operator “[]” at the highest level means that two possibilities are available: some suitable interaction technique should be provided to the user in order to highlight that initially s/he can choose from two possibilities, clearly displaying to him/her what they are, and to activate the dialogue associated to each branch of the task model. Using an intuitive graphical language we express the information obtained up to now with the picture in Figure 4.3:

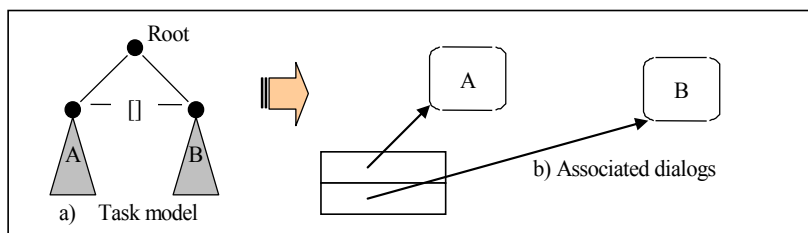


Figure 4.3: Implementing choice among tasks.

In the part a) of Figure 4.3 the task model has been shown highlighting only operators and subtasks at the level that is currently considered (the root level), neglecting for the moment the further specification of each subtask; in the part (b) of the picture it has been shown what the task model means in terms of structure of dialogues and presentations, which is—in the considered case—that two dialogues (whose structure has been temporarily left unspecified) have to be designed—one associated to the execution of the subtask A and another one for subtask B. For the moment, the choice of the interaction technique most suitable to *activate* the two different dialogs can be put off (by using two items in a menu—as stylised in the picture—is the most intuitive example).

Back to the task model and going ahead in the visit of the task model, the analysis goes down across the subtask A, whose decomposition has been shown in the part (a) of the below Figure 4.4:

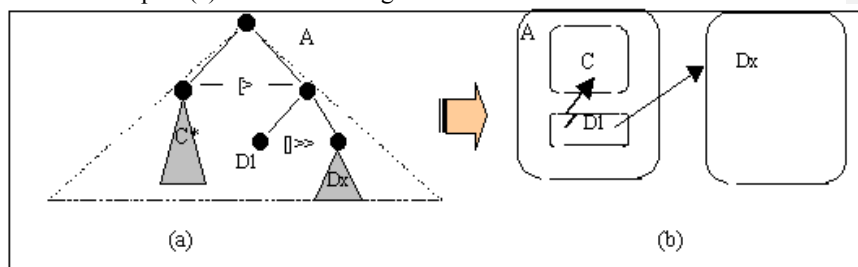


Figure 4.4: Implementing disabling operator between tasks.

What are the information associated to the task model in Figure 4.4? The presentation associated to A should be structured in such a way that the activity of the iterative task C could be performed more than one time until the first action of the disabling task D is activated (note that task D is decomposed into D1 $[] >> Dx$ and the first subtask of task D, D1, should be always available to the user during the whole performance of subtask C—in the shape of some interaction technique, e.g. a button as in the picture or something like that). When finally D is started, the presentation should convey to the user the information captured by the task model, which is that, once the D1 subtask has been started, the C subtask is no longer available. This effect could be achieved activating another separate dialogue associated to Dx.

Down again into the C task, its structure can be viewed as that in the part (a) of the following Figure 4.5:

The meaning associated to the $[[[]]]$ operator is that the activities are concurrently interconnected each other as they exchange information and in addition, as the parent task can be performed more than one time, it is useful to show the presentations associated to them grouped in the same dialogue (see part b) in order to highlight how they exchange information each other.

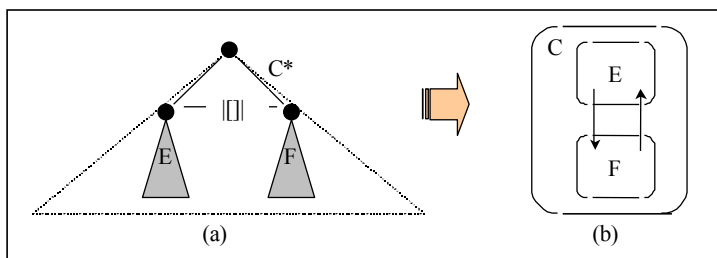


Figure 4.5: supporting concurrent tasks exchanging information.

If we summarise the results achieved up to now, we can say that —as far as it concerns the temporal constraints between dialogs associated to subtasks of task A— the presentation should be structured as in the following picture (using the same intuitive representation that we used before):

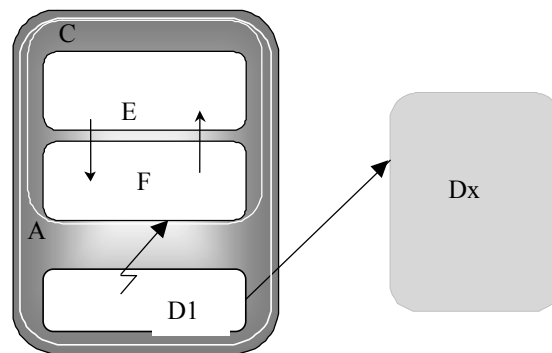


Figure 4.6: Implementing the example

Title: Task models and task-based design

Id Number: D 2.1

➤ **Analysis of the task**

Now we focus on the presentation techniques associated to each parent task and which we left undefined. We need more information about tasks: the goal, the type and category of task, objects manipulated by tasks with their type and cardinality,

As we previously saw, the dialogue associated to the *Ordering paths* task (C in Figure 4.4) is composed of two logical sub-dialogs, the first one dedicated to the selection of the parameters, and the second one where the set of paths are displayed according to the chosen criterion.

As far as it concerns the first dialog, it is associated to a Selection task (the user has to select item(s) from a predefined set of elements), so the decision about its presentation has to consider this in terms of the possible choices that should be provided, e.g. the cardinality of this set, how they should be provided to the user (single choice, multiple choice). For instance, the possible parameters which the controller could be interested to know are the calculated length of the path, the number of foreseen runway crossings, how much time the travel will take (supposing a standard velocity on the taxiways), and so on.

The user interface should allow the user to select a list of parameters from a defined set of parameters (allowing a multiple choice) and within this set to allow to mark which one (single choice) has to be used as the ordering criterion. A possible implementation is shown in the left part of window in Figure 4.7: controllers can mark which parameters are relevant to them (a “√” is associated to each selected parameter) but only one parameter can be selected as the ordering criterion (which is highlighted by a different presentation technique in the user interface). There is also an Edit button that allows the controller to change the criteria available.

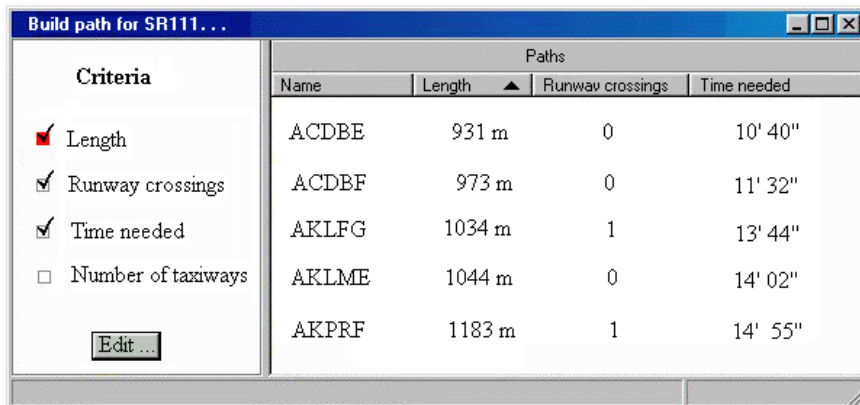


Figure 4.7: selection of a path.

Then, we consider the task of providing to the user (the controller) with the set of the possible solutions from which the controller will choose the “best” solution.

Title: Task models and task-based design

Id Number: D 2.1

The type of the activity that the controller performs is to decide which is the “best” solution, so the user interface should enhance all the interaction techniques that highlight the Comparison of different elements of the same set. In the picture this is modelled by using an ordered list of elements that share the same structure, so it is easy to compare different values referring to the same parameter.

Other considerations have to be done about the type and cardinality of data being the ultimate choice about the specific presentation that have to be used (especially that exploiting multimedia features) dependent on the integrate consideration of all those aspects. For example, being the path a spatial data, a good way to present it is by using some graphical technique, but this can be confusing when there are many paths that have to be displayed at the same time. In this case, a good solution could be to provide more than one type of presentation, for example an immediate textual presentation of the path, and the possibility to have on request a graphical presentation too (for example selecting a textual representation of a path, then the path is graphically highlighted in a separate window as shown in Figure 4.8).

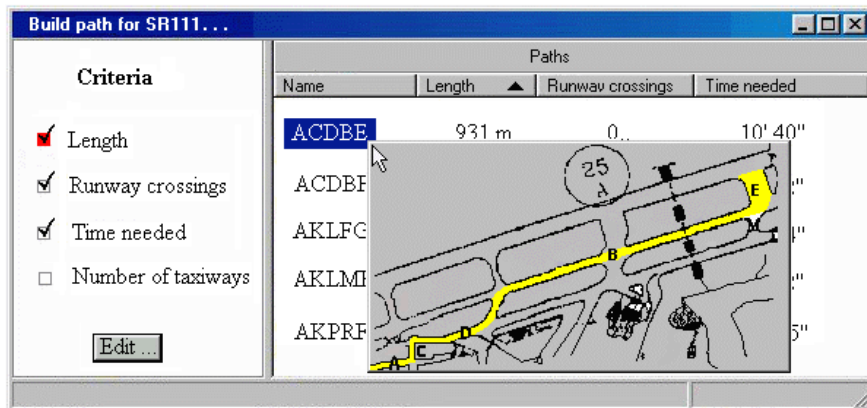


Figure 4.8: Interactive selection and graphical representation of path.

As we derive from the task model the activity of managing parameters and display accordingly the solutions could be performed more than one time, however, when the controller finally selects the path (for example by double-clicking on it as shown in Figure 4.9), before sending actually it a preview should be made available to show a summary (*Show summary information* task) of the main characteristics of the selected path.

Title: Task models and task-based design

Id Number: D 2.1

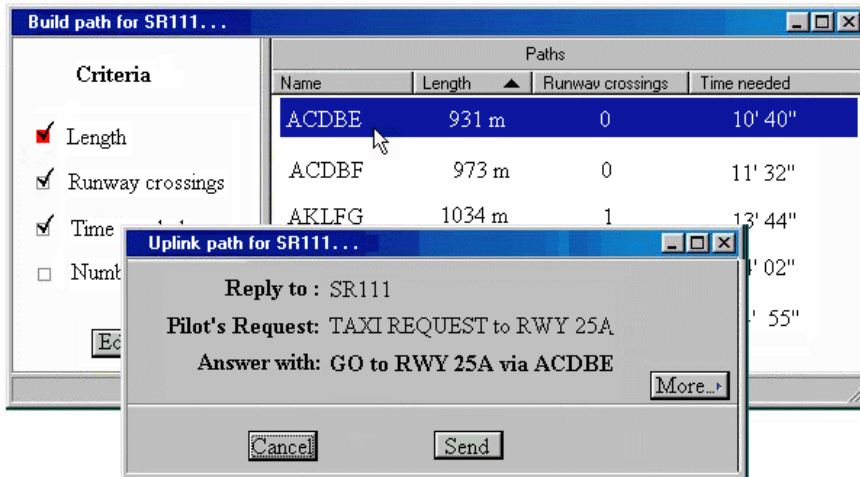


Figure 4.9: Automatic generation of a command to send to a pilot.

In Figure 4.9 we show a possible implementation of the ideas that have been expressed: the controller by double-clicking on a path activates a window where a possible answer for the pilot has been already composed. S/he can decide to send this path to the pilot or not, or to view other information on this path ("More..." button). As this figure has just explanatory goals, we suppose for sake of simplicity that only one predefined format of answer exists to reply a taxi request ("GO to RWY <rw> via <path>") even though it would be more realistic to provide the controller with the possibility of selecting other formats and/or specifying additional options on his/her answer in order to make them more flexible. For instance, once the controller has decided for a particular path, s/he could decide to specify other options on when the pilot should start to execute the order.

Further improvements could be thought: for example the possibility to specify some range of values that the calculated set of solutions has to satisfy for a selected parameter (this solutions could be pretty useful when the calculated set of solutions is very large, so the controller needs to refine the selected criterion) or even better, to allow the controller to specify more than one criterion to be optimised: sometimes it might occurs the case that two (or more) solutions come up to share the same value for a selected criterion, and in this case the controller should be able to select an additional criterion to have a further classification order within the first one in order to support controllers in their decision-making process to identify the "best" solution. In this way the controller could define the criteria in successive steps of refinement, that is at first s/he selects only one criterion, view the set of

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

proposed solutions and then (when it makes sense) s/he can select the other criterion to do a more complete evaluation of the solutions proposed.

Now we pay attention on how the controller could be supported by the user interface when s/he decides to build autonomously the path. In this case more freedom should be given to the controller although the system should still support controllers in building the path, allowing them to evaluate the effects of their decisions: the task of sending a path is really safety-critical so it is really important to avoid that meaningless solutions could be sent to the pilot.

If we analyse the task model associated to the task *Build path without automatic suggestion* (see again Figure 4.1) we can note that the first, optional task is just to give controllers the possibility to choose some parameter that they want to know while they build the path, and then to specify the path.

Suggestions on how the controller can build the path could be derived by the type of the data. For example, from the point of view of the domain objects, paths are spatial objects, so a graphical presentation should be the best because it is immediate to convey such information (in the next section we show an example of a graphical composition of the path when we discuss the flight labels). However, the taxiways composing the paths are named by strings, so other techniques could be used in order to allow an expert controller to directly enter the name without using graphical presentations. In addition, the total number and the names of taxiways are statically known, so interaction techniques allowing the controller to choose within a predefined set of possible values could be useful (e.g. a menu), because the possibility of directly enter a string is really prone-to-error mechanism. Thus, a menu is an efficient mechanism to avoid typing errors and it is efficient with a limited set of elements, so we can suggest that, in order to avoid typing errors, it could be used providing as selectable choices all the possible taxiways in the aerodrome (as long as they are a small/medium number).

However, the considered system is a safety-critical system, so we cannot limit attention only to syntactical errors: all the “semantic” errors that could be detected with the information contained in the system should be avoided providing additional checking mechanism. For example, in order to decide which options offer in the user interface, the design should consider the functional dependencies that exist in the set of data avoiding the possibility that the user can choose no-sense options and offering on the user interface only the actual subset of sensible choices: for instance, if the flight is at the gate G1 and from the gate G1 the taxiways that could be followed are only A, B and C taxiways, the interface should allow the controller only to select from this set and not from the all possible taxiways of the aerodrome. The same holds when the controller enters the string of the path: some checking mechanism has to be provided by the system in order to check and discover some enter-data errors, not only syntactical but also supporting and/or implementing functional dependencies between data: for instance, if taxiway A is

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

not directly connected to the taxiway B the user interface should not allow to enter “AB” string as portion of path.

4.2.2.2 *Build picture current state* task (flight labels for ground)

a) General issues

A key point to investigate is the information provided by flight labels. The idea behind the flight labels is that they should be a sort of “mini-strip”, collecting all the information controllers could be interested to know on a flight under their responsibility. However, showing all the information permanently on the radar screen is not recommendable for the safety of the system because the aerodrome is really crowded of vehicles and aircraft. It becomes crucial to decide different levels of priorities in terms of presentation of data, by selecting from time to time the information that are important for the controller (and that should be made permanently available, e.g. standard label) from other information that should presented after an explicit action of the controller (e.g. selected/extended label).

1. As the *standard label* is supposed to be displayed permanently on the radar, a radar displaying the situation of an aerodrome can have many flights to present, safety issues claim to reduce as much as possible the information to display permanently. Thus, the information presented in the standard label should be overall the “most important”: this refers firstly to how frequent are tasks that manipulate (read and/or modify) specific data in a specific period of time or flight phase, then the opportunity to offer them soon available on the radar. Secondly, in terms of how much information they give to the controller in order to easily maintain the overall picture of all the flights in terms of both their current state and future intentions of the pilots just looking on the aerodrome map.
2. As far as it concerns the *selected label*, the different ways of activating it imply quite a different intention of the controller. When the controller activates a selected label (by moving the mouse on the standard label), behind this interaction there is the implicit request to know precise information about a *specific* flight (the controller has selected a specific flight); so differently from the selected label, where the information enhanced the possibility to get the best “overview” of the system in the minimum possible space, the information in the selected label should allow the controller to interact with the flight.
3. The *extended* label is the third format of the flight labels that are supposed to exist in the new environment. In a stripless environment (as that supposed for the envisaged system) they are expected to contain all the information regarding a specific flight. Thus, from the extended label the controller should obtain at least all the information that in the current system s/he derived from paper strip (for example, the history of the datalink messages exchanged with the pilot should be made available as in the paper strip the list of instructions/annotations

were available). Safety assessment suggest the opportunity to display the extended label in a separate window and to make it movable in order not to hide large areas of the screen.

b) The standard label

From the task model we can see that the two controllers are interested to different information to perform their ordinary activities: for example, the Ground's activity is mainly dedicated to drive and follow the flights during their taxiing phase, thus s/he is interested to data concerning the current and expected positions of the flights on the taxiways (see task model in Figure 4.10). On the other hand when the TWR wants to get information about an aircraft under his/her responsibility s/he is mainly interested to check the current state of the runway and/or to know the aircraft category and SID in order to calculate separation, information not so relevant for the GND. Thus, the findings are that only a part of information is really meaningful for each controller at some time, thus, the UI has to be designed in such a way to appropriately filter the information distinguishing data actually meaningful for each controller.

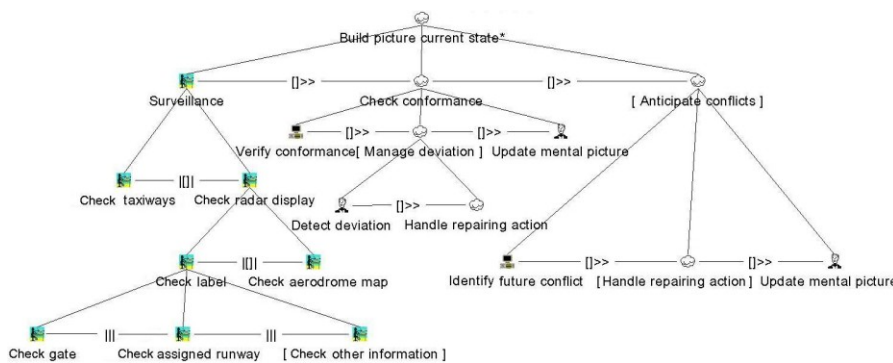


Figure 4.10: Build picture current state task (Ground – new system).

In addition, the system is really time-dependent (e.g. the flight's position in the aerodrome continuously changes), thus the associated information displayed in the labels should accordingly change too in order to show only the information that is really useful to the controller from time to time, filtering it from the information that has become meaningless (or that could be easily derived by other sources) because some conditions changed, and to provide the controller with information continuously kept up-to-date.

The same happened in the current system too: e.g. when the Ground controller gives the path to a flight s/he updates the paper strip in order to keep track of the order just sent and to be able to check the flight afterwards. Thereon, from time to time the controller is interested to just a part of the annotation, being the *current* taxiway the border line between the useful information (future positions in the path) and the useless information (past positions in the path). As you can see there are cases

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

where artefacts and associated interaction techniques are different between current and envisaged system (e.g. paper strips are supposed not to be available in the new system), but the goals remain unchanged.

In the new system the challenge becomes to make easier this activity exploiting as much as possible the capabilities of the new (electronic) environment. Thus, once the controller has given the path to the flight, the information about the destination point (the runway) has been further refined and detailed by the information about the path, which has to substitute the previous one because it details much more the intentions of the flight in the next future instead of informing only about the foreseen target.

Thus, for example, once the flight has sent its request for taxiing and is waiting at the gate for the path, displaying the information about gate in the standard label could be pretty redundant if the controller is aware of the current state of the flight (departing flight, waiting for a path) and see its icon on the radar near the representation of the gate. On the other hand, the same does not hold for the information about the flight's assigned runway, because s/he could not derive it by looking at the radar and indeed this information is crucial just for the activity that s/he has to perform: building and sending the path.

In addition, when it is possible (within the same controller's UI) to distinguish these data depending on different states or flight phases, the presentation should take into account this aspect, varying dynamically the data presented and possible operations/actions that could be performed by each controller on these data.

In the following figures we show a possible way for implementing the standard label of the Ground controller while the aircraft is under the Ground's responsibility. In Figure 4.11 the flight SR111 is at the gate, waiting for the path from the Ground controller. The path is between an origin point and a destination point: just looking at the radar screen the controller is able to know the first one (the gate), information about the second one should be explicitly provided by the system in the standard label because it is the first data that the controller look for in the system. For this reason the standard label (at this phase) shows information about the intention of the flight that is to reach the runway 27L.

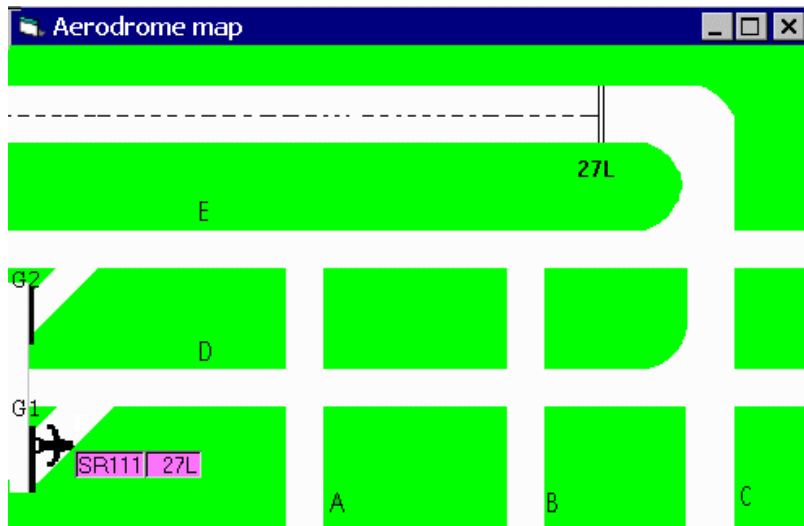


Figure 4.11: Start of an interaction to communicate a path.

In Figure 4.12 it is shown how the Ground controller can select and build the path (*Build path without automatic suggestions* task in Figure 4.1): s/he clicks on the different taxiways that s/he wants the flight to cover (*Specify next taxiway* task), and the system consequently draws step by step the path in a graphical way. In addition, other information can be displayed associated to the portion of the path that has been selected up to now: for example, a useful information can be to show from time to time how much time should take the travel between the origin point and the current last point selected on the path, in order to facilitate the controller to match the expected departure time of the aircraft or alternatively how good is the match with the slot time (if present) or with the expected time of departure.

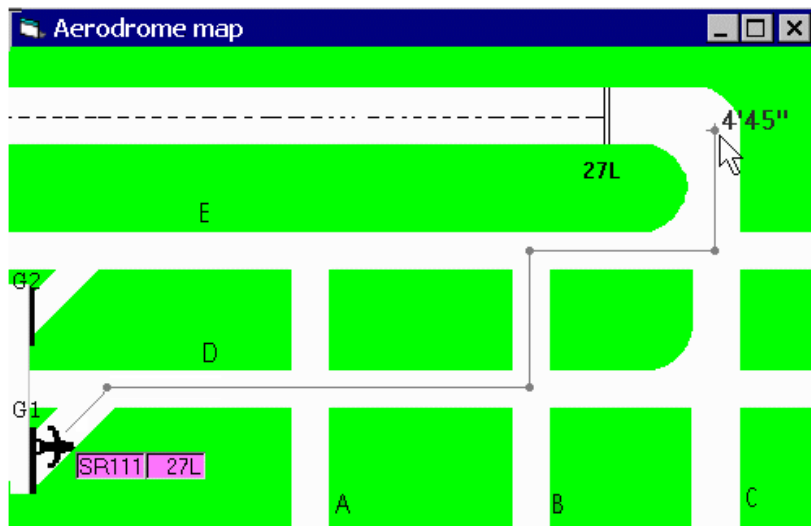


Figure 4.12: Build path without automatic suggestion path.

Of course, the controller should be always able to change mind and select another path, but, finally, when he decides that the selected path is the best path for the flight, s/he should be allowed to send the selected path in the most immediate way. For example, by double-clicking on the last point selected the system automatically changes the information contained in the flight label: instead of the “27L” the system sets automatically the selected path (DBEC in the Figure 4.13), showing it in a different colour in order to highlight to the controller that the data has changed.

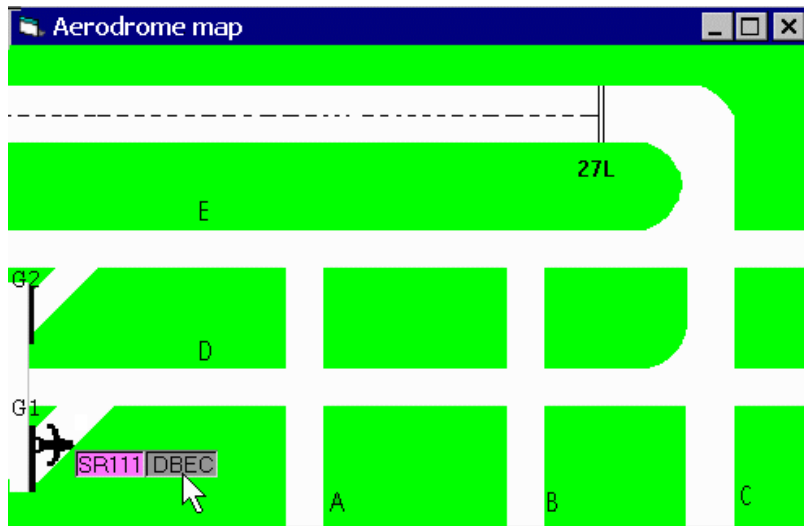


Figure 4.13: Update flight label.

When the selected path has been actually sent to the aircraft, the changed state of the aircraft is reflected on the standard label which now displays the received path from the controller. More importantly, as it is shown in Figure 4.14a, the information on the standard label is continuously maintained up-to-date, as the field of the path contained in the standard label changes depending on the current position of the aircraft in the aerodrome: as the past positions of the aircraft are no longer useful for the controller, the flight label shows only the portion of the path that the flight has still to cover, so in the part a of the picture only the next taxiways (BEC) is shown. We have to note that it is only one possibility: another possibility is – in order not to clutter the screen with too much information- to show only the name of the next taxiway.

The controller can optionally enable the displaying of the path in a graphical way and in this case the label shows in a different colour also the current taxiway, in order to have conformance between the displayed graphical information and the textual information on the label (see Figure 4.14b).

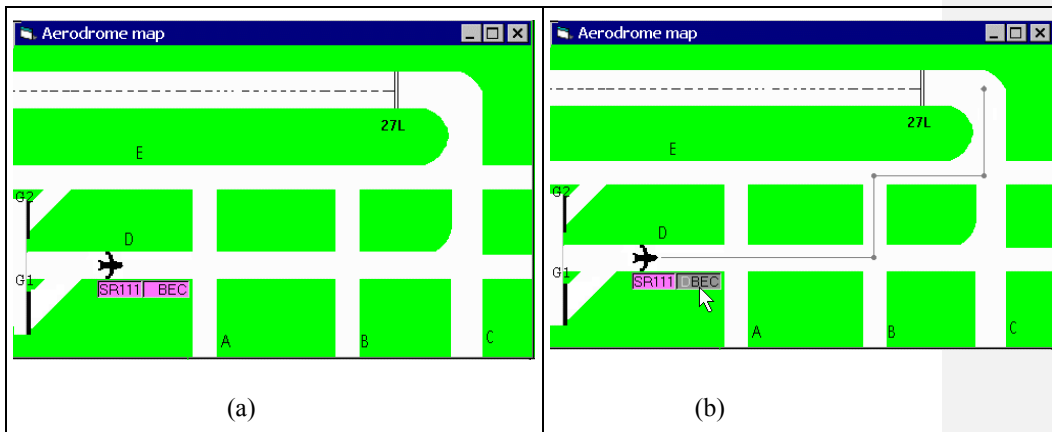


Figure 4.14: An interaction with an enriched flight label.

Of course, this is just an example of how to implement labels for the GND controller and other possibilities can be envisaged, but the most general guideline is that the labels should:

- present different information depending on the tasks that the specific controller want to perform, as they are interested to different data (in the following sections we show a suggestion for a TWR's label different from the previous one);
- present (when possible) dynamic information that changes depending on the changed state of the aircraft.

In this way, relevant information for the Ground controller (such as the path) are permanently shown on the radar display, allowing the ground controller to have them always soon available.

In addition, the interface should also provide the controller a graphical presentation of the path displaying just the current position of the aircraft and the portion of the path that it has still to cover (as we previously saw, the task model gives the rationale not to show the taxiways that have already been covered). With regard to the specific interaction technique to use, the choice of permanently displaying on the radar a graphical presentation of the path (as it happens for the textual presentation) reveals soon unacceptable because of the high number of aircraft that are in their taxiing phase at any moment, and the consequent confused layout deriving by displaying at the same time a lot of intersecting lines. A possible solution is to display the graphical path only after explicit interaction of the controller (e.g., by pressing the mouse on the associated field in the standard label, as in Figure 4.14b).

4.2.2.3 The guidance task

The task model gives useful information to derive temporal constraints that have to be implemented in the design of the user interface and to derive which are the actions that should be made available to the controller and which not.

In the previous sections we saw how the controller builds the path (with automatic suggestions or without them) and then s/he sends it to the pilot. However, at this point his activity is not yet finished, because s/he has to continuously check if the aircraft is following the expected path (although in the new environment we suppose available automatic tools which perform monitoring tasks) within the expected interval of time, in order to match as its best the expected time of departure and minimise the possibility of delay. In order to do it, the controller—once s/he has sent the path to the pilot and the pilot starts to move the aircraft on the taxiways—should have the possibility to send instructions to pilot to change the velocity of the aircraft during all the period of taxiing phase, depending on some environmental conditions and/or his/her strategic decisions.

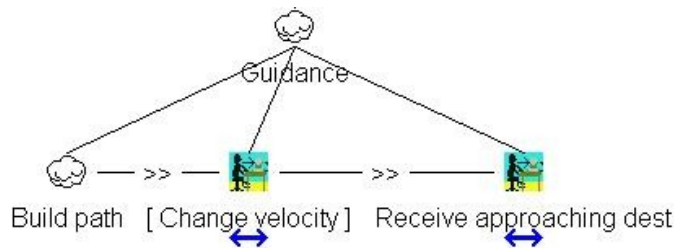


Figure 4.15: Guidance task.

Thus, although we assume that every aircraft is supposed to move on the taxiways at a standard (default) velocity, we allow that that speed can be modified by an explicit instruction of the controller in order to optimise the managing of the traffic. If we analyse the Figure 4.15 what we have modelled in the task model is that the activity to modify the velocity of the aircraft can be activated on the controller’s interface only after the path has been sent to the aircraft (Enabling operator) in order to avoid that an instruction to “speed up/slow down” an aircraft could be unintentionally sent to the pilot without any previous path. This kind of temporal constraints should be always implemented when necessary, in order to ensure the maximum level of safety in the system and the minimum room for the human slips and/or errors.

What we want to capture with the task model is that *before* having sent the path to the pilot the user interface should not allow the controller to change the velocity,

whereas *after* having sent the path the “Ground speed” field can be modified by the controller.

In both cases, all the interactions that are currently possible should be made available to the controller with a suitable interaction technique. For instance in the first case should be appropriately highlighted to the controller that no interaction is possible with the “Ground speed” field of the selected flight label in order to prevent him/her from starting at all any interaction (see Figure 4.16a where no interaction is possible as the scrolling/editing are disabled), whereas in the second case the changed state of the field should be highlighted (see Figure 4.16b, where the scrolling/editing is enabled) and a suitable technique should be provided to the controller. The ultimate choice about this technique should involve first of all considerations about type of data, cardinality and range of values that the flight parameter “Ground speed” is supposed to span and, in addition safety assessments about whether and how to limit the available interactions (for example safety considerations about the specific environmental conditions of the aerodrome could suggest to further limit the interval of the possible values assumable by the Ground speed field).

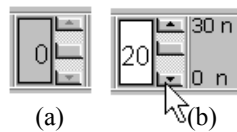


Figure 4.16: The interaction technique to support change of speed.

4.2.2.4 *Deviations and warnings*

The analysis of deviations allows designers to better analyse how the UI should warn the controller when the system detects some possible hazardous situations. Three main guidelines should be followed:

1. *different levels of warnings/alarms should be distinguished* depending on the different levels of urgency of situations that could arise in the system (a “more” serious hazardous situation requires different presentation techniques with respect to a “less” serious situation).
2. *different media* should be used to convey different information to the controller, exploiting the different nature of the media used;
3. *avoid to overload* the controller too much (e.g. too many different warnings coming from different sources for the same problem) and too often (select the actual hazardous situations in order not to make the controller to get accustomed to see/hear warnings in the system and not to noise them unnecessarily).

Therefore, when a classification of the possible hazardous situations has been carried out (for example, a runway incursion is one of the most serious situations) the UI should exploit in appropriate way its foreseen *multimedia* (audio/visual)

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

possibilities: for example, in case of highly serious situation an *audio* warning is the most appropriate way to be sure to capture the controller's attention as soon as possible (the controller could look at some other tools in the system, or out of the window so a visual alarm could not be immediately received), and the attributes of the audio signal could be calibrated in such a way to increase its effect depending on the importance (for example, the "volume" attribute of the alarm could be set at a level that is much bigger as much more hazardous the situation is (to be sure that controller is aware of it also in noisy situations), the "tone" attribute could be calibrated in such a way to associate high-pitched sounds to reinforce the information about the urgency of the warning), and so on. Thus the audio channel should convey information about how urgent and how "catastrophic" possible consequences of the hazardous situation are. On the other hand, once the controller's attention has been captured, the visual channel (exploiting its non-transient nature) should convey additional information about the *source* of the hazardous situation and (when possible), showing possible *suggestions* about the admissible solutions to the problem itself, although the controller remains the ultimate responsible for the final decision and action.

As far as it concerns handling the possible hazardous situations occurring on the taxiways (*Control hazardous situation* task), we have to say that the possible deviations of the controlled environment can be classified depending on two main criteria: there are deviations that are time-based (an a/c is in the right position but at the wrong time) and a deviation space-based (an a/c is in the wrong position).

With regard to the space-based deviation, as we show in Figure 4.17, the UI should highlight both the current position of the aircraft and the position where the aircraft was supposed to be, back until where deviation starts to occur: in this way the controller is able to locate the deviation and its extent, and to focus his/her attention on the most safety-critical areas (which are those wrongly covered by the aircraft). As in almost the hazardous situations the time is the most safety-critical factor, we suppose that the system is in the best position to calculate the optimum solution to solve the deviation in the minimum interval of time so, the first action of the controller is to ask the system for the best solution (*Ask for best solution* task in Figure 4.17): if the solution is okay for him/her, s/he has only to send it, alternatively s/he should be able to build an alternative path by him/herself.

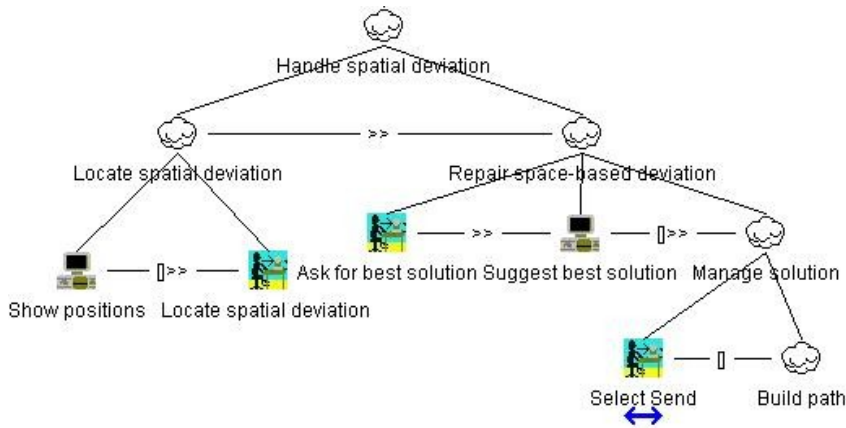


Figure 4.17: Handle spatial deviation task.

In Figure 4.18 we have summarised a possible solution on how to handle a spatial deviation: the “abnormal” nature of the situation is highlighted by means of a message that appears on the standard label and with an appropriate colour in order to highlight its urgency and to attract the attention of the controller. The user interface highlights the deviation and allows the controller to get quickly a possible solution (possibly shown in a graphical way) that the system has automatically calculated to re-integrate safely the aircraft in the traffic flow (see Figure 4.18)

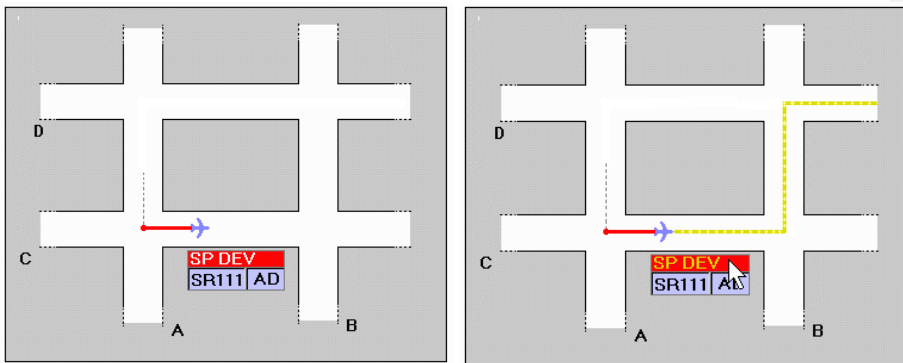


Figure 4.18: Example of spatial deviation.

4.2.3 Tower Controller

4.2.3.1 Correlation between different views

In the current system, when the Tower controller receives a request from a pilot, s/he has to make a decision about the best answer for this request. In order to do it, s/he finds the associated paper strip in the bay to collect as much as possible information and appropriately decide how to cope with the request in the current situation of traffic. In the new system, the tasks that the Tower controller has to perform are the same, most of the differences are on the tools and artefacts that are supposed available and from which the controller finds the flight data: e.g. for departures they are mainly the Departure Manager, the enriched flight labels shown near the flight’s representation in the aerodrome map, and the list of data-link messages received from pilots.

In fact, each of these tools provides the Tower controller with a specific and often partial “view” of the same referred object (the aircraft): for example, the Departure Manager displays the planned order of departures, showing the foreseen departure optimum time for each flight, the list of received data-link messages informs the controller about the future intentions of the pilots in terms of their requests, whereas the icon of the aircraft in the aerodrome map gives quick knowledge of the current position of the aircraft, with the standard flight label showing permanently immediate information about the flight.

When the Tower wants to collect information about the flight, s/he has to perform a sort of “merging” operation between those different sources of information in order to have the most complete picture of the current/planned state of the aircraft. This situation is described specifically in the task model of Figure 4.19.

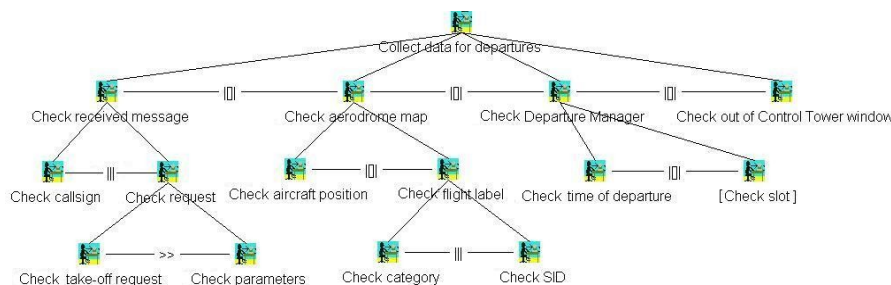


Figure 4.19: *Collect data for departures* task (Tower – new system).

From the task model you can see that there is the “[[]]” operator, meaning that there is not a pre-defined order with which those actions have to be performed (e.g. the controller could look at the Departure Manager and then in the aerodrome map, but the vice versa is also possible), being the key point the fact that, whatever the followed order is, all the information in the different provided tools are necessary in order to build the complete picture of current/future state of the aircraft. In addition, it is worth noting that once the controller has fixed the information about a specific aircraft in one of the three considered tools, s/he should perform other two

“retrieving” tasks in order to identify the related information within the other two sources of information.

For example, once the Tower controller has selected a specific request from a flight within the list of received data-link messages s/he should still perform a search within the aerodrome map in order to retrieve the icon of the associated flight, and another search within the Departure Manager message in order to know what the planned optimum departure time has been scheduled for the flight. In the new (electronic) environment it is possible to highly reduce the requested workload for performing those task by providing the controller with some automatic link that highlights automatically the correlation among the different “views” of information referring to the same object.

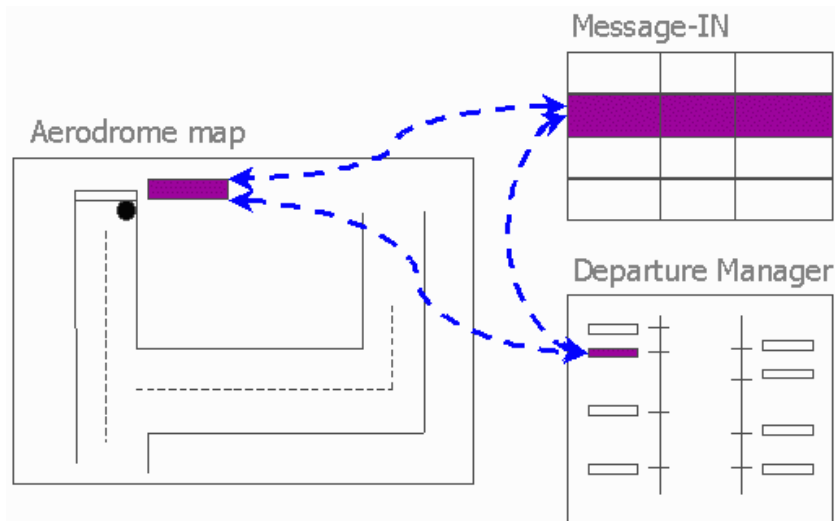


Figure 4.20: Different representation related to the same object.

In order to summarise the guideline just explained, we can say that, once different views of information are supposed spread between different tools the UI can be improved by supplying:

1. Possibility to highlight the same information in the different views (e.g. when it is selected in one of them for the Tower controller the link can be between the Departure manager, aerodrome map, Message-in list. In this way it should be soon clear that for example the aircraft A in position (x,y) within the aerodrome map has made a request to take-off and within the departure manager has been scheduled to take-off at a specific time T;
2. The different views have to be consistent each other (for instance an inconsistent situation for the Tower controller could be that an aircraft that has made a request to take-off is not displayed in the Departure Manager);

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

3. The different views take into account conditions that change in a dynamic way depending on the time
4. From each view it should be possible to activate the operations that are logically enabled from time to time (e.g. give the possibility to send a clearance from the label, from the message in the list of messages, from the icon in the Departure manager)

The above suggestion could be made more general and adapted appropriately to the different cases and phases of the flight being the aforementioned example just a specific case of use. The general guideline is that some automatic link should be provided in order to easily find information that are strictly connected to each other: e.g. for the Tower controller we can say that a correlation should exist also for the arrival flights (they are not displayed in the Departure Manager) between the icon in the aerodrome map and the list of received messages, and the same matter can be applied to the Ground controller too.

4.2.3.2 Flight labels (standard) for Tower control

As far as it concerns the Tower controller, we know that his/her main task is to manage take-off and landing, giving to them the clearance to takeoff/landing in such a way to ensure that adequate separation has been provided between consecutive flights. First of all, the problem of information that has to be provided in the standard label arises. As it occurred for the Ground controller, the data that have to be displayed in the standard label are the information the controller should have soon available.

Focussing on the departing flights, the Tower controller receives them in a queue that has been predisposed by the Ground controller on the basis of the expected optimum departure time and this sequence should match as much as possible the scheduled order. However, in situations of heavy traffic, sometimes occurs that a queue of several flights ends up to accumulate in the proximity of the holding position and (if necessary) the Tower controller should be able to perform some modifications to this queue in order to manage the maximum number of flights in a specific interval of time.

First of all, in order to decide which information has to be shown in the standard label (the label permanently displayed on the screen), we have to do a compromise between the need of offering soon available the information to which the controller is mainly interested, and the necessity not to clutter the screen in a region of the display where we suppose to have a high concentration of aircraft (being near the holding position of a runway).

We know from the task models (Figure 4.19) that the information which the Tower controller is mainly interested concerns the foreseen departure time and the information about the assigned runway, the category of the aircraft and the SID

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

(according on such information s/he is able to ensure adequate separation between consecutive departing flights on the same runway that even share the same SID). As far as it concerns the information about the assigned runway, when the Tower receives from the flight the request to take-off, the aircraft's icon is probably close to the assigned runway, so the information about the runway can be easily derived by looking at the aerodrome's map (anyway the information about the runway should be displayed in the selected label). About the other information (category and SID), they should be displayed in the standard label, because they are both necessary to the Tower in order to decide separation between consecutive flights. As far as it concerns the most suitable way to present that information, again this choice is mainly driven from the type of task and cardinality of data that have to be manipulated: with regard to the aircraft category, as the cardinality of the possible categories is low (only three different values) and the type of data that have to be shown is a quantity (Light, Medium, Heavy) we can present it by means of some graphical presentation such in the following Figure 4.21:

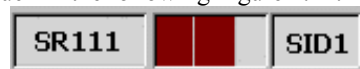


Figure 4.21: Flight label representation.

For example, in the above figure the three different categories of aircraft have been modelled by means of three different areas which are progressively filled up as the category increases (e.g. the *Heavy* category is when all the rectangles are coloured). The low cardinality allows users to easily discern between category (whereas the same presentation would not have been equally suitable if the cardinality was around ten values). In addition, we have to note that this kind of presentation could make easier some comparison tasks (between categories of different aircraft) because of its intuitive graphical representation. Considerations about the opportunity of reducing the occupied space in the standard label could suggest to change the orientation of the rectangles (from horizontal to vertical, as in the Figure 4.22).



Figure 4.22: Another flight label representation.

A similar reasoning about type and cardinality can be applied to decide the presentation of the SID: the most simple presentation is the textual but considerations about the type of task (spatial) and the cardinality of the SID suggests that a more intuitive presentation (as shown in Figure 4.22) could be used.

4.3 Towards guidelines for safety-critical systems

In this section we summarise the main guidelines that we previously indicated for the design of user interface for safety-critical systems starting from the analysis of the task models.

1. *Taking into account the task type.* For example, spatial tasks (tasks that allow users to provide or manipulate spatial information) should be supported by graphical presentations to improve the immediacy with which convey such information to the user and avoid that the users can perform errors while they handle those data (see the task of building a path for the Ground controller).
2. Whenever a task manipulates numerical/quantitative data, provide presentations (using graphical attributes) that enhance the performance of typical activities connected with those data e.g.: comparison). For the ultimate choice about the best presentation consider also the cardinality of data to present (See the presentation of the category of the aircraft for the Tower controller).
3. *Implementing a user interface dialogue that is consistent with the temporal relationships indicated in the ConcurTaskTrees model* reduce the possibility to introduce errors, which is important especially for tasks with high level of safety-criticality. (See the presentation supporting Change velocity task for the Ground).
4. In order to reduce the amount of information to provide permanently to the user, *define different levels of priority amongst data*; limit the permanently displayed data only to those that are necessary to get the overall picture of the current and future situation and give the possibility to get additional information only after an explicit action of the user; allow to get presentation with different level of refinement; maintain the information always up-to-date and allow to read/modify different information depending on different activities that have to be performed on a specific object from time to time. (See the presentation of enriched flight labels).
5. If it is possible to identify different types of users that perform different activities and are interested to different data, design *different user interfaces for each of them* in order not to provide them with meaningless information and no-sense interactions. (See the different labels for the two controllers).
6. When there are different tools that offer different, partial “views” for the same object, provide the user with an automatic link that allows him/her to get an *immediate correlation between the different views* obtaining the most complete picture of the object starting from each of these tools. While each view is more suitable for a specific task it is useful that they are able to support also those tasks that are primarily performed by the other views. (See the relationships among different tools for the tower controller in the new environment).

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

7. *Change the allocation from the human to the machine* for tasks (especially routine tasks) that force users to distract their attention from the most safety-critical activities. (See updating automatically flight data).
8. Provide *different levels of warnings/alarms* for each of the different deviations that could occur in the system depending on the impact on the safety of the system. Exploit different nature of media to convey different information to the user about how to cope with hazards (e.g. audio media to attract attention, visual media to suggest solutions). Avoid to overload the user with too many alarms/warnings, avoid the user makes accustomed to them. (See handling spatial deviation).

5. Reasoning about Task Models

One of the advantages of using a formal approach is the possibility to rigorously reason about properties of the specification. This can be carried out by *model checking*: the specification represents the model against which properties can be checked, and it is usually based on the analysis whether the transitions in the states of the specification satisfy the properties given.

Formal verification has been successfully used in hardware design where it is important to check that some properties are satisfied before implementing the specification into hardware. The HCI field is more challenging for verification methods and tools, since the specification of human-computer dialogues may be more complex than the hardware specifications. The main problems in applying model checking techniques to the design of user interfaces are:

- the identification of relevant user interface properties to check
- the development of a model of the User Interface System which is meaningful and, at the same time, avoids the introduction of many low level details which would increase the complexity of the model without adding important information for the design of the user interface.

There are various motivations to carry out model checking, in our case we found particular important the following:

- *it is possible to test aspects of an application even if they have not been completely implemented;*
- *user testing can be rather expensive*, especially in fields as what we consider Air Traffic Control, the users (controllers in our case) are highly specialised personnel whose time has high costs. We are not proposing that they should not be involved but we are indicating that model checking can decrease the need of empirical testing, even if it is always useful to have it;
- *exhaustive analysis*, the advantage of model checking is that the space of the states reachable by the specification is completely analysed. In user testing we just consider one of the possible traces of actions whereas there can exist a huge number of such traces and even an extended empirical testing can miss some of them. This lack of completeness in empirical testing can have dangerous effects especially in safety-critical contexts. However there is another difference between model checking and empirical testing: in the former case a model of the application is considered whereas in the latter case the focus is on the concrete implementation of the application (or part of it). This means that the model should be a meaningful approximation of the application in order to have a useful analysis of it. This opens an interesting issues that is on the one hand to have models sufficiently detailed to support a meaningful analysis and evaluation and on the other hand to have models that can be dealt with automatically so as to avoid an explosion of possible states.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

In [P97] there is a discussion on how to approach the verification of user interface properties and examples of a first set of general properties is given. Other approaches to the same type of problems can be found in [AMW95], [ASDR98]. Here we want to extend that approach to analyse multi-users interactive applications and address real case studies where the effort of using a formal approach is justified by the safety-critical context.

5.1 Integrating model-checking in user interface design

The part of our method concerning the use of formal methods is represented in Figure 5.1, where the processes are indicated with circles and the results with rectangles. After a first phase gathering information on the application domain, and an informal task analysis, designers should develop a task model which forces them to clarify many aspects related to possible tasks and their relationships.

An additional reason for introducing ConcurTaskTrees was that after first experiences with LOTOS [PF92] we realised the need for a new extension that allowed designers to avoid useless complicated expressions even for specifying small behaviours and to focus on more important aspects.

The ConcurTaskTrees specification can be used for two purposes: to drive the development of a software prototype consistent with the indicated requirements as we indicated in Section 4 and to transform it into a LOTOS specification. We have implemented a transformation tool where each task specification is translated into a corresponding LOTOS process. The motivation for this transformation is that there are various model checking tools able to accept LOTOS specifications as input, such as the CADP package [G97]) that transforms it into a finite state automata or Labelled Transition System (LTS).

On the other hand, the informal information initially gathered is also used to identify the relevant properties of the user interface, which can be both general formal properties, such as mutual awareness, and other informal properties that are specific of the considered application domain (for example, in the ATC application domain, the request that a controller's voice clearance has to be received by all the pilots currently in the sector).

Checking that the formal specification satisfy the relevant properties for the possible dialogues it is useful to understand whether the design developed can support usability and safety aspects.

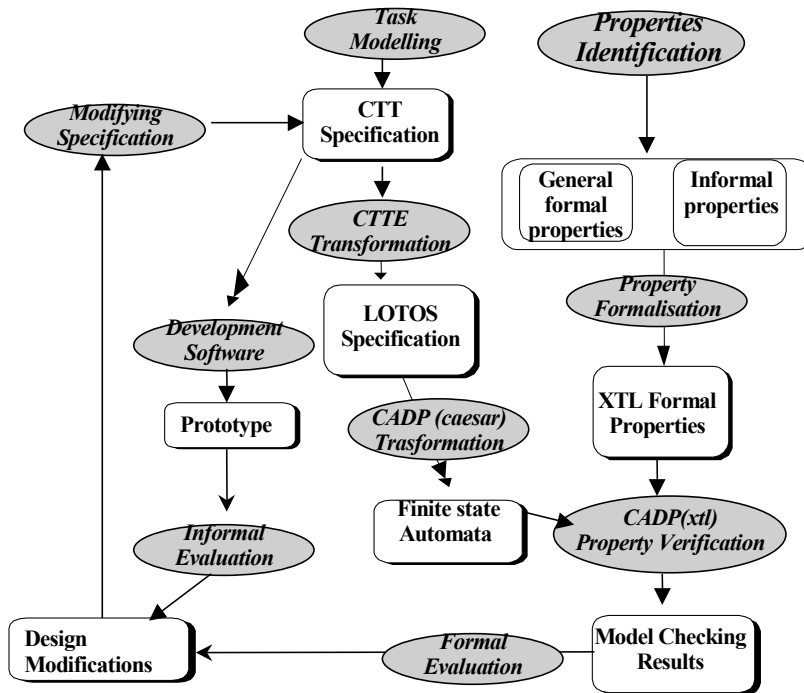


Figure 5.1: A Graphical Overview of the proposed Approach

After having formalised the identified properties with a formal language, these properties are checked against the LTS specification to verify which properties are valid in the system. Properties of the user interface are expressed as temporal logic formulae by the designer and model-checked against the model describing the Interactive System software derived in previous steps.

The verification is performed by a general purpose automatic tool for formal verification and the results of the model checking are used for formal and informal evaluations that can lead to modify the ConcurTaskTrees specification thus re-starting the process.

5.2 From ConcurTaskTrees to LOTOS

The first important aspect to consider is that the translation from ConcurTaskTrees to LOTOS is composed of two relevant steps: to handle the translation of the ConcurTaskTrees tasks into LOTOS processes, and to implement the ConcurTaskTrees temporal operators by means of the operators provided by LOTOS language.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

With regard to the former issue, for each process, its specification implies that all the gates that are used inside the specification have to be declared in its heading. The direct consequence is that in the specification of the root process all the gates detected have to be listed, whereas in the specification of a process corresponding to a leaf task the translation is reduced to insert the associated gate and the *exit* action, in order to indicate when the successful end of the process occurs.

An example of the latter issue is the translation of the iteration operator appearing on a tree root. Then the translation of a ConcurTaskTrees iterative process is reduced to have a recursive call to the same process at the end of the execution of the process itself.

5.3 User Interfaces Properties

As far as it concerns a possible categorisation for properties, classical taxonomies already exist in the area of interactive systems. In a highly cooperative system as that considered in our case study (but the same issues apply to other systems presenting similar features) the distinction between the different user roles involved plays an important part to get a more modular view of the system's and the sub-systems' properties. Depending on the considered role we can elicit properties on the specific user interface customised for the particular user, but at the same time we can bring out other properties that are related to the interconnections and communications between different users.

We will use an extended version of ACTL [DFGR93] to formalise examples of properties in the case study previously introduced. This type of extension can be easily converted in an XTL [MG98] expression that can be verified by the CADP tool.

5.3.1.1 Warning message for time-out expired

With datalink functionality, all the messages have a time-out indicating the time interval within the associated answer have to be received in order to be appropriately considered and evaluated. When it happens that time-out expires, an appropriate notification has to be shown on the message originator's interface, in order to signal either that the message has to be sent again, or that possible answers received after the time-out expiration have to be ignored. More precisely, the property can be expressed in this way:

If there is expiration of an operational time-out without reception of the operational datalink response message, the message originator shall be notified with an appropriate feedback.

The related ACTL-like expression is:

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

$\text{AG}([is_sent(controller, request, pilot)]E[true\{\sim is\ received(controller, answer, pilot)\}U\{timeout_expiration\}A[true\{true\}U\{is_presented(controller, noanswer_feedback)\ true}])$

This means that whenever (AG operator) the *controller* has sent a *request* to a *pilot*, then we have at least (E operator) a temporal evolution during which no associated answer coming from the pilot to the controller has been received and finally, as a result of the expiration of the fixed time-out we reach a state from where for all the possible temporal evolution (A operator) the desired effect of presenting an adequate feedback to the controller's user interface of the missed answer will be reached (*noanswer_feedback* in the above property).

This means that only after the expiration of the time out we are sure that the desired effect (the user interface showing to the controller that a particular order previously sent to the pilot has not followed by any answer in time for being correctly processed) will be reached, thus allowing the controller to decide what is the best action to perform in order to make up for the error.

5.3.1.2 Controllers' mutual awareness

This property means that whenever the ground controller executes an action on his/her user interface, the associated effect has to be shown on the user interface of the tower controller. With this property we want to be sure that the tower controller is aware of all the actions (that we denote with "control_action" wording) performed by the ground controller which can have an impact on the his/her own activity, namely either actions that a controller can perform directly on the system based on his/her own decisions (for example the ground controller can change a previously fixed flight parameter) or actions that involve datalink dialogues with pilots. In other words, we want to pay attention to all the actions that might cause that controllers' activities clash each other, thus we do not consider the actions that the ground performs in order to get information on the system for monitoring it.

More precisely, in ACTL, we specify that whenever (AG operator) the ground controller performs a modification action on his/her user interface then for all the possible temporal evolutions (A operator) the event associated with the user interface modification reception will occur on the tower controller's user interface

$\text{AG}([is_executed(ground, control_action)]A[true\{true\}U\{is_presented(tower, update_effect)\ true}])$

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

With “is_executed” and “is_presented” we want to distinguish when the system generates and undertakes the action from when the effects of the action are presented on the user interface. Of course, the property holds for the tower controller too.

5.3.1.3 Controllers' coordination

A direct consequence of the awareness is that the two controllers are more synchronised on these actions' sequences when (for example) a flight passes from one controller's handling to the other controller. The most intuitive example is during the hand over from the ground controller to the tower controller for departure flights (whenever the pilot reaches the holding position, the ground controller performs a last contact and then the control is passed to the tower controller) and vice versa for arrival flights. However, it is worth noting that in these cases, the previously cited awareness mechanisms present in the system (the last contact message performed by the ground controller is displayed on the tower controller's user interface, so the tower expects a pilot's message in the near future) comes just before the explicit pilot's first contact message requiring to the tower controller to be considered ready to take-off and then properly scheduled.

But there is another case of proper controllers' *co-ordination*: for example, this is the case when a departure flight has to cross an active runway in order to reach a different assigned runway. The ground controller gives to the flight a path on the taxiways until the flight reaches the runway that s/he has to cross, thus on the one hand the pilot is aware of when s/he arrives at that point he has to wait for a message from a tower controller (who takes on responsibility for runways), and, more importantly, the tower controller knows that, when the pilot has reached the crossing s/he has to provide clearance to go through the runway as soon as it is possible, without any explicit request from the pilot.

$\mathbf{AG}([\text{is_sent}(\text{ground}, \text{path}, \text{pilot})] \mathbf{E}[\text{true}\{\text{is_received}(\text{ground}, \text{path}, \text{pilot})\} \mathbf{U}\{\text{is_stopped}(\text{pilot}, \text{runways_crossing})\} \mathbf{A}[\text{true}\{\text{true}\} \mathbf{U}\{\text{is_sent}(\text{tower}, \text{ok_crossing}, \text{pilot})\} \text{true}]])$

This means that once the ground controller has sent the path to a pilot in order to reach the assigned runway, we have a temporal evolution during which the previous message has been received by the pilot and finally we reach a state by performing the pilot's action of stopping at the crossing of the taxiway with the runway, from where for all the possible temporal evolution the desired effect (the tower controller sending the authorisation to cross the runway) will be reached.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

5.3.1.4 Controlled sharing

The tower and ground controllers share flight information of all the planes currently under their control (for example they can always obtain flight data by means of datalink menus) however, in order to serialise the control actions performed by each controller (for example sending datalink messages to pilots), it is important to guarantee that, while the flight is under the control of the tower controller, the ground controller can not send (voluntarily or unintentionally) control orders to pilot until the tower controller performs a last contact and then the flight passes under the control of the ground controller.

$\mathbf{AG}[\text{is_sent}(\text{pilot}, \text{first_contact}, \text{tower})] \mathbf{A}[\text{true}\{\text{not}(\text{is_sent}(\text{ground}, \text{control_order}, \text{pilot}) \mathbf{U} \{\text{is_sent}(\text{tower}, \text{last_contact}, \text{pilot})\} \text{true})]$

This property means that if an arrival pilot sends a first contact message to the tower controller then it will not be possible to have that the ground controller sends a control_order to the pilot, until (**U** operator) the tower controller has been sent to the pilot a last contact message.

Title: Task models and task-based design	Id Number: D 2.1
-------------------------------------------------	-------------------------

6. Conclusions

In this report we have described the use of task models that has been developed in the MEFISTO project.

The original contribution lies in the engineering approach to task models that stems from the use of a flexible and expressive notation, the support of automatic tools and the development of criteria to support the design of user interfaces using information contained in the task model.

In the third year we plan to focus on some specific design aspects and to use the task models to support the usability evaluation of the prototypes developed.

7. References

- [ASDR98] B.d'Ausbourg, C.Seguín, G.Durrieu, P.Rochè, Helping the Automated Validation Process of User Interfaces Systems, Proceedings ICSE'98 pp.219-228
- [AWM95] G.Abowd, H.Wang, A.Monk, "A formal technique for automated dialogue development", Proceedings DIS'95, ACM Press, pp.219-226.
- [BBDGMPY96], S.Shum, A.Blandford, D.Duke, J.Good, J.May, F.Paterno', R.Young, "Multidisciplinary Modelling for User-Centred System Design: An Air-traffic Control Case Study, Proceedings HCI'96, London, Springer Verlag, pp.200-218.
- [BP93] Burns, D.J. and Pitblado, R.M. (1993) A Modified HAZOP Methodology For Safety Critical System Assessment. *Directions in Safety Critical Systems — Proceedings of the Safety-Critical Systems Symposium*, Bristol, 1993, Springer-Verlag.
- [C95] Carroll, J. (Ed.), *Scenario-Based Design*, John Wiley and Sons & C., 1995.
- [DFGR93] De Nicola, R., Fantechi, A., Gnesi, S. and G. Ristori (1993). An action-based framework for verifying logical and behavioural properties of concurrent systems, Computer Network and ISDN systems, 25, 1993, 761-778.
- [D91] A.Dix. "Formal Methods for Interactive Systems". Computers and People Series. Academic Press 1991.
- [G97] H. Garavel, M. Jorgensen, R. Mateescu, Ch. Pecheur, M.Sighireanu, B.Vivien, CADP'97 - Status, Applications and Perspectives
- [H96] A.Hall, "Using Formal Methods to Develop an ATC Information System", IEEE Software, pp.66-76, March 1996.
- [H88] Hopkin, V.D. (1988) Air Traffic Control. In E. L. Wiener and D. C. Nagel, Eds. *Human Factors in Aviation*. Academic Press, 1988. Pages 639-663.
- [L97] Leathley, B.A., (1997) HAZOP Approach to Allocation of Function in Safety Critical Systems, In *ALLFN'97, Proceedings of the 1st International Conference on Allocation of Functions.*, Galway, Ireland, IEA Press.
- [ISO88] ISO (1988). Information Processing Systems - Open Systems Interconnection – LOTOS - A Formal Description Based on Temporal Ordering of Observational Behaviour. ISO/IS 8807. ISO Central Secretariat.
- [JK96] John, B., Kieras, D., "The GOMS Family of Analysis Techniques: Comparison and Contrast". *ACM Transactions on Computer-Human Interaction*, Vol.3, N.4, pp.320-351, 1996.
- [MOD96] HAZOP Studies of Systems Containing Programmable Electronics. UK Ministry of Defence, Interim Def Stan 00-58 Issue 1.
- [MG98] R. Mateescu and H. Garavel, XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. Proceedings of the International Workshop on Software Tools for Technology. Transfer STTT'98 (Aalborg, Denmark), July 1998.
- [N93] Nielsen, J., *Usability Engineering*, Academic Press, 1993.
- [P97] F.Paternò, Formal Reasoning about Dialogue Properties with Automatic Support , Interacting with Computers, August 1997.
- [P97] F.Paternò, Formal Reasoning about Dialogue Properties with Automatic Support , Interacting with Computers, August 1997.
- [P99] Paternò F., *Model-Based Design and Evaluation of Interactive Applications*, Springer Verlag, 1999
- [PM99] Paternò, F., Mancini, C. , Developing task models from informal scenarios. Proceedings of CHI '99 - May 15-20, 1999, Pittsburgh, USA.
- [PST98] Paternò, F., Santoro, C., Tahmassebi, S. (1998) Formal Models for Cooperative Tasks: Concepts and an Application for En-Route Air Traffic Control. In Proceedings DSV-IS '98, Springer Verlag, U.K.
- [PSF99] Paternò, F., Santoro, C., Fields, B., (1999) Analysing User Deviations in Interactive Safety-Critical Applications, Proceedings DSV-IS'99.
- [WP3-3] F.Paternò, C.Santoro, Task-based design of aerodrome case study, MEFISTO Working Paper.