

ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models

F. Paterno', C.Mancini, S.Meniconi

CNUCE-C.N.R.
Via S.Maria 36
56126 Pisa
Italy
{fabio, cristian, silvia}@cnuce.cnr.it

ABSTRACT In this paper we discuss a notation to describe task models, which can specify a wide range of temporal relationships among tasks. It is a compact and graphical notation, immediate both to use and understand. Its logical structure and the related automatic tool make it suitable for designing even large sized applications.

KEYWORDS Task models, formal notations, model-based user interface design, tools for design.

1. INTRODUCTION

The use of task models in user interface design and development has recently been recognised as an important contribution to obtain more user-oriented interactive applications. A new generation of model-based (Foley et al., 1991) tools such as Trident (Bodart et al., 1992), TLIM (Paterno' et al., 1996), Mastermind (Szekely et al., 1995), Adept (Johnson et al., 1994), ICO (Palanque et al., 1995), and AIDE (Sears, 1995), all use, in different ways, task-models to support user interface design.

It is becoming increasingly common for the various specialists involved in this process (developers, designers, psychologists, application domain experts) to discuss the tasks that the system should support. To this end it is very important to have notations to develop task specifications so that:

- they are easy to understand and use, thus improving communication among people discussing the design;
- they are able to structure large sized specifications which are developed in industrial applications;
- their semantics is precisely defined to avoid ambiguities in the communication.

When choosing the notation we found that some important features have to be supported:

- hierarchical logical structures which were introduced by GOMS (Card et al., 1983) have proved to be a useful way to represent task models because they allow designers to reason about the design at different abstraction levels and they support the refinement design process better;
- modern user interfaces are characterized by highly interactive behaviours in multimedia environments, it is thus important to be able to express a wide variety of temporal relationships;
- a task model for an industrial application may be complex, it is thus important to be able to express relevant relationships precisely and to have information on more detailed aspects in an interactive way.

When we started our work on task models we considered UAN (Hartson and Gray, 1995) as a good starting point. UAN supports hierarchical specifications and has operators to express temporal relationships among tasks. However we found it had two main limitations: it has a textual specification which makes it difficult to read and interpret (for example to find cross-references among tasks); and it gives limited support for deriving a software architecture, as its main purpose is to specify only the

externally perceivable behaviour of the user interface by associating tables indicating above all user actions and system responses.

Thus our first contribution was (Paterno' and Mezzanotte, 1995) to provide a graphical representation of the hierarchical structure and to use LOTOS (ISO, 1988) operators to express the temporal relationships among tasks at the same abstraction level since they have a more formally defined semantics. Furthermore LOTOS is an international standard notation already used in various industrial and research centers to specify applications which have, as an important feature, temporal ordering among possible actions.

We found this approach useful but, especially when we started to apply it to industrial applications, we found some points which required further improvements:

- ambiguity in the priority among operators used at the same level of the task tree;
- the need to specify which component of the Interactive System considered will perform the task;
- additional information about the tasks that could be used in the user interface development;
- the necessity to define better the relationships between a father task and its subtasks;
- new operators for a compact specification of common situations.

In this paper we describe how we overcame these problems. We first introduce the main concepts which drive how we build task models. Next we show pieces of ConcurTaskTrees specifications for expressing examples of interactive applications. Then we show how the tool supporting these specifications works, and finally some concluding remarks are given.

2. TASK MODEL

A task defines how the user can reach a goal in a specific application domain. The goal is a desired modification of the state of a system or a query to it.

A task in our approach is described by the following attributes:

Name : used to identify the task

Type: there are four possible types: abstract, user, application, interaction

Subtask of: name of father task

Objects: vector of objects, each element defines: name of object, type of object (internal, perceivable), list of input object actions, list of output object actions

Iterative: a Boolean indicating whether the task is iterative

First action: The set of possible initial actions is indicated

Last action: The set of possible final actions is indicated.

At each level the task specification is built in two steps: first the objects are identified and then the actions which allow the communication among them are defined.

2.1 Objects

Objects are entities which are manipulated to perform tasks by the associated actions.

The objects are user-perceivable and internal objects, which are both manipulated to perform the tasks by using the associated actions.

Perceivable objects are items which users can interact with using their senses, for example menus, icons, windows, voice, sounds, and so on. They can belong to application or interaction tasks.

Internal objects are entities which belong to the application and which need to be mapped onto perceivable objects to be presented to the user. Examples of internal objects are: the state of a request for a data base, the data base itself.

Each object can belong to one or more tasks. In the task decomposition process when we consider a new task level in the task-tree, objects, which were defined in the previous task level, may receive two types of manipulation:

- decomposition (one object at a task-level is decomposed into two or more objects at the next task level);
- refinement (the set of actions associated with one object is increased when we consider the next task level).

2.2 Actions

Actions are associated with objects. Actions can be cognitive, logical, or physical. For tasks which are not iterative we have to indicate what the last action is. We need to specify the possible initial actions

because they are used when we are evaluating expressions which indicate a possible choice ($[]$ operator, which means that at the beginning, both tasks are available and once one of them is started the other is no longer available) between tasks or disabling among tasks ($[>$ operator, which means that once the first action of a task occurs then the other task is deactivated).

2.3 Specification of the component performing the task

We can identify four types of tasks depending on the allocation of their performance:

- *user tasks* are performed entirely by the user, they require cognitive or physical activities without interacting with the system. Possible examples are when the user reads a list of flights satisfying some constraints and selects one of them for his/her journey or, in a video conference application, the possibility to analyze the content of some information received and evaluate whether or not it is sufficiently clear; if not, the user asks for clarification. Thus user tasks are associated with some processing performed by the user on information received from the environment.
- *application tasks* are completely executed by the system. They receive information from the system and they can supply information to the user. They are activated by the application. For example, compiling a program and sending messages when some errors are detected, or receiving network messages and displaying them.
- *interaction tasks* are performed by user interactions with the system. These interactions are activated by the user. Examples are editing a diagram or formulating a query to a data base.
- *abstract tasks* are tasks which require complex actions, and their performance does not completely fall into one of the three previous cases.

In the task specification the types of tasks are presented differently either by different icons or different geometric shapes as in Figure 1:

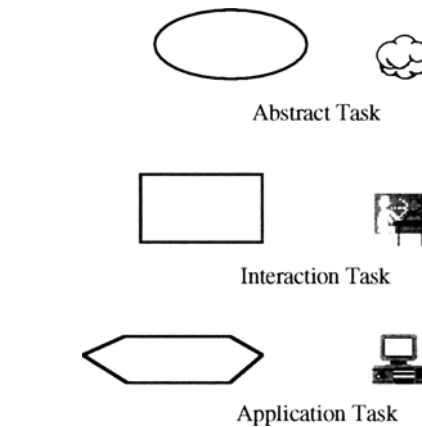
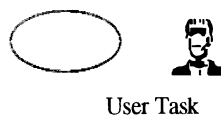


Figure 1: The two possible presentations of Task types.

We make a further distinction. Tasks can be:

- *processing tasks*, given an input they perform some processing and then provide a related result;
- *control task*, their main purpose is to generate an event control: they inform other tasks that a given condition has been reached.

3. CONCURTASKTREES

The task model is built in three phases:

- a hierarchical logical decomposition of the tasks represented by a tree-like structure;
- an identification of the temporal relationships among tasks at the same level;
- an identification of the objects associated with each task and of the actions which allow them to communicate with each other. The identification process is performed layer-by-layer.

The temporal relationships among tasks are expressed by using an extension that we have defined of the operators of the LOTOS notation which is a concurrent notation. ConcurTaskTrees thus allows designers to describe concurrent tasks, unlike the GOMS proposal which uses hierarchical task decomposition but is only able to analyse sequential tasks (though some more recent proposals have tried to overcome this limitation).

Unlike UAN, two tasks can synchronise ($[]$ operator) in the approach proposed. This happens when they

have to exchange information: the output information of one task is the input information for the other task. Tree-like structures with concurrent operators to indicate temporal relationships among tasks at the same level allow designers to specify more complex behaviours than those associated with finite state automaton.

The operators that we use to describe the temporal relationships are:

T1 ||| T2, *interleaving*: the actions of the two tasks can be performed in any order;

T1 |[]| T2, *synchronization*: the two tasks have to synchronize on some actions in order to exchange information;

T1 >> T2, *enabling*: when the first task is terminated then the second task is activated;

T1 []>> T2, *enabling with information passing*, in this case we want to highlight that when T1 task terminates it provides some value for task T2 besides activating it;

T1 [> T2, *deactivation*, when one action from the second task occurs the first task is deactivated;

T1*, *iteration*, the task is iterative;

T1(n) *finite iteration*, how many times the task will be performed is specified;

[T1], *optional task*, its performance is not mandatory;

T, *recursion*, the possibility to include in the task specification the task itself.

3.1 Solution to the ambiguity problem

If we simply build task models using these operators the first problem is the possible ambiguity of some expressions. For example in Figure 2, we can interpret the specification in two ways: either (T1 [] T2) ||| T3 or T1 [] (T2 ||| T3).

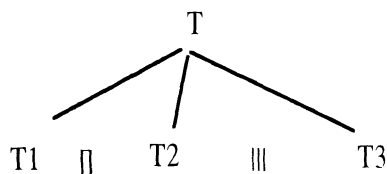


Figure 2: An example of possible ambiguity.

To solve the ambiguity there are two possibilities: we can use the priority order among operators defined in the standard LOTOS (choice > parallel composition > disabling > enabling), or we can introduce a task which disambiguates the expression, as in Figure 3. Parallel composition can be either completely

interleaving among tasks, or interleaving with synchronization on some actions.

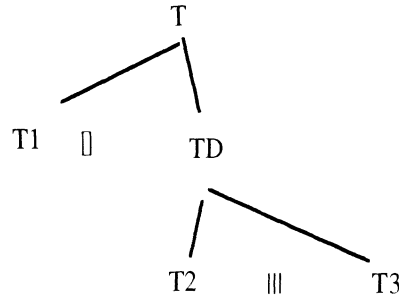
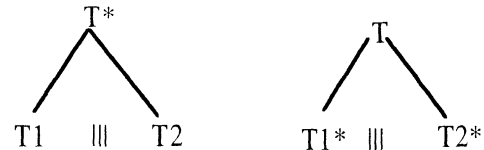


Figure 3: A solution to solve the ambiguity.

3.2 Relationships between father task and its subtasks

If we use these operators in a hierarchical context we can achieve a lot of flexibility in terms of being able to specify different behaviours. For example in the next diagram there are two possibilities: the same tasks are involved. T1 and T2 are performed in interleaving in both cases but if the iteration is at the father level then we mean *order independence performance*: tasks can be executed in any order but before executing them again both tasks have to be terminated. In the second case, with iteration introduced at the subtask level, we have a *continuous interleaving among tasks* which means that once one task is terminated it can be executed again without waiting for the termination of the other task.

An example of order independence is when specifying a request for a flights database then users have to specify both departure and arrival towns but the order is not meaningful. Once the request is transmitted then a new one can be composed in the same way. In the same application we have continuous interleaving between the task specifying a request and the one clearing it: they can both be executed many times without any limitation on their order.



Order independence performance

Continuous interleaving

Another example which gives a clear indication of the possibilities of this type of representation is shown in Figure 4. In both cases we have a selection and then a choice of two possibilities which provide different ways to modify the element selected. However, there is one important difference. In one case the choice of task to perform is made by the application which detects how many elements have been selected, and depending on the result, allows the user either to choose between editing and deleting (if there is only one selected element) or just deleting (if there is more than one selection element). In the other case the

choice of task (changing an object size or an object colour) is made directly by the user. We express these different behaviours by specifying the type of task which is the father of the tasks that can be chosen: if the father task is an application task, this means that the choice is made by the application (though the performance of the selected task can still be made by the user or his/her interactions) otherwise, if the father task is an interaction task, the choice is made by a user interaction.

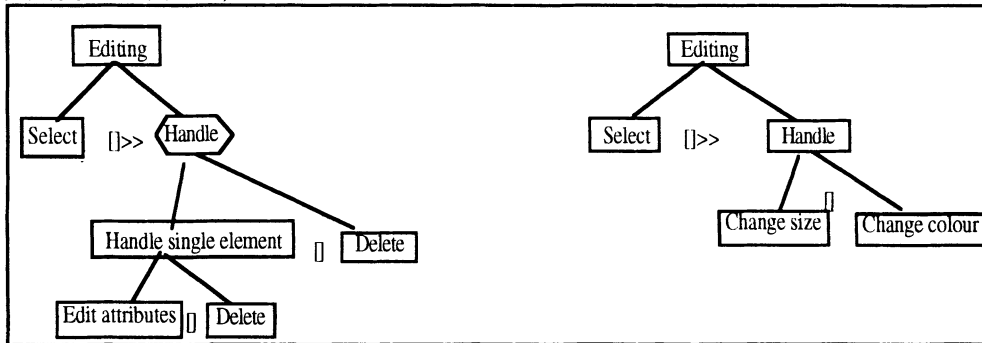


Figure 4: Different ways to make the choice.

3.3 Recursion

We assume that we want to describe the example described in Figure 5. Whenever an edit button is selected then a new order is activated. At some time the set of orders can be deactivated by the close button and the activated orders remain active.

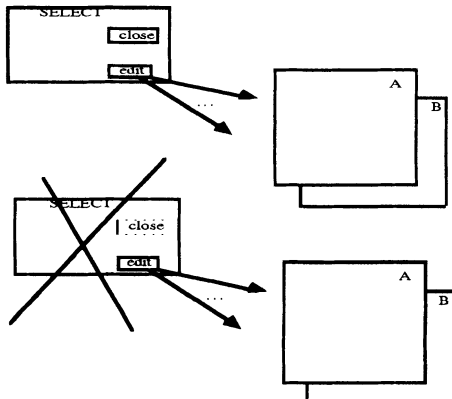


Figure 5: An example of user interaction with a set of orders.

In ConcurTaskTrees this behaviour can be described in the following way where we have a recursive definition of the Application task.

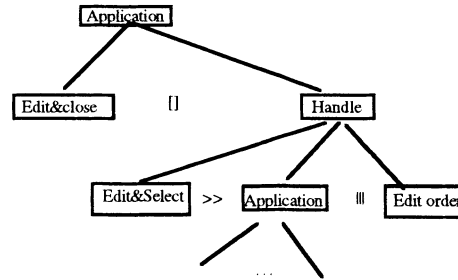


Figure 6: An example of recursive task specification.

The difference between iterative and recursive specifications in ConcurTaskTrees is that in the first case, when we use the * operator, the performance of the related task is repeated in the same modalities until another task deactivates it. The second case, when in the definition of a task we find the task itself again,

allows us to describe the cases when, depending on some condition (in our example the selection which activates a new order form), we want to perform a task again without knowing in advance how many times we will perform it and, for each new performance, we will give the possibility to perform a new, additional task (in our example editing a new form). In the task tree it is sufficient to specify the possible recursion at only one level to indicate that it may happen more than once.

In our example the user can stop the recursion by selecting the Edit&close task. Once this task has been selected then we stop the possible recursion (because it is in the other option of the choice) but we do not stop the Edit order tasks created, which can be terminated separately. We had to introduce the handle task to manage correctly the priorities among operators.

3.4 Examples of specifications in ConcurTaskTrees

One common situation is the need for the user to cancel a modification which has just been specified. This can be described by the tasks represented in Figure 7. We have a generic application during which the user can edit various possible modifications. Once a modification has been specified the user can either perform it or cancel it. In both cases, as editing is an iterative task, then the user will be able to specify a new modification until the close task is performed.

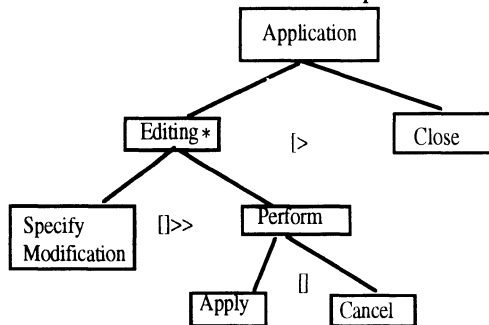


Figure 7: Specification of cancelling.

Another interesting situation is *exception handling*: a request is transmitted to the functional core which sends a message to the user indicating that something exceptional has happened. If we consider the structure of the previous example we have to add an Exception task, as a subtask of the Perform task. This Exception task has a first phase, performed by the application, which generates a warning message and a second part,

an interactive subtask, which allows the user to manage the exception in some way.

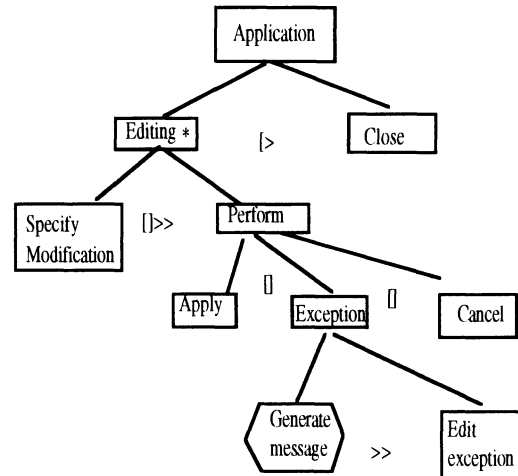


Figure 8: Specification of exception handling.

4. THE TOOL SUPPORTING TASK MODELS DESIGN

We have developed an automatic environment (Paterno' et al., 1997) which supports designers in developing task and architectural models (Coutaz, 1993), and in identifying their relationships. It is implemented in Java. At the beginning there are three choices:

- editing a task model;
- transforming a task model into an architectural model in semi-automatically;
- analysing the relationships between a task and an architectural model.

Whatever choice the designer has made it is always possible to select again the other possibilities. The task editor is a direct manipulation task which allows the designer to build the task tree and to use the LOTOS operators to indicate their temporal relationships. When a task is selected then by selecting one of the icons associated with the possible four task types, new subtasks are automatically created and located in the editing area.

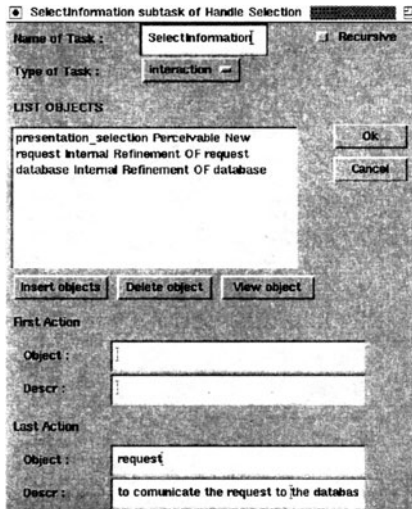


Figure 9: An example of the specification of objects.

It is also possible to select a task and to provide additional information about it (Figure 9): the objects that it manipulates and what the possible first and last actions are. For each of these objects, it is possible to activate dialogue boxes which allow the designer to specify the actions which they require (see Figure 10).

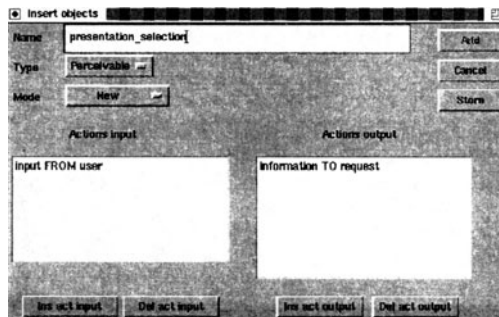


Figure 10: An example of specification of actions.

Actions are classified depending on whether they are input or output actions. Actions are used to allow objects defined at the same level of the task tree to communicate. At any task level it is possible to check the semantic consistency which means, for example, given an input action of an object, at the same task level, there must be an object with the corresponding

output action. A task specification can be saved in order to be reused and edited again later on.

Figure 11 shows a simple example of a task model for interacting with an electronic museum. At the beginning we identify a session (Handle selection task) which can be disabled by a specific task (CloseSession task). The session first allows the user to select a type of work of art (sculptures, paintings, and so on). Once a type has been selected, the application presents information related to the selected type of work. Further selection criteria can be recursively specified and cancelled until the user has made a specific request to indicate all the information satisfying the desired requirements. Finally, the user can select one of them and the application will present it.

5. CONCLUSIONS

We have presented a notation supported by a related tool to build task models. It uses graphical constructs and operators derived from a formal specification. The resulting notation gives a compact specification of a wide variety of dynamic task behaviours. The tool which supports ConcurTaskTrees is available to the public at <http://verdolo.cnuce.cnr.it/task.tgz>. We have applied our approach to various applications (videoconference, museum systems, business applications) which confirmed its ability to address many possible situations.

To facilitate its use in industrial applications we are designing a set of templates which allow models for common patterns in user interactions to be reused.

We have developed a method (Paterno' et al., 1997) able to derive architectural models and implementations which are consistent with the temporal and semantical requirements indicated in task models obtained following the approach described in this paper.

ACKNOWLEDGEMENTS

We wish to thank Ilse Breedvelt and Camjel Severins for useful discussions on the topics of the paper, and Progetto Finalizzato Beni Culturali for partially supporting the work presented.

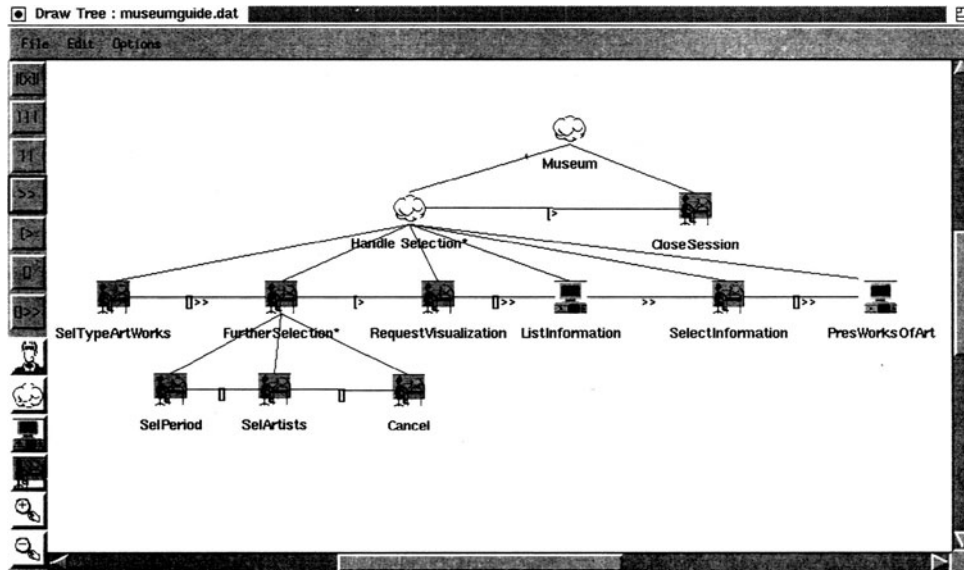


Figure 11: An example of task model obtained with the support of the automatic tool.

6. REFERENCES

- F.Bodart, A.Hennerbert, J.Leheureux, J.Vanderdonckt, (1995) "A Model-based approach to Presentation: A Continuum from Task Analysis to Prototype", in *Interactive Systems: Design, Specification, and Verification*, pp.77-94, Springer Verlag.
- S.Card, T.Moran, A.Newell, (1983) "The Psychology of Human-Computer Interaction", Lawrence Erlbaum Ass. Publ., Hillsdale, N.J., 1983
- J.Coutaz, (1993) "Software Architecture. Modelling for User Interfaces", in the *Enciclopedia of Software Engineering*, pp.38-50, 1993, Wiley.
- J.Foley, W.Kim, S.Kovacevic, K.Murray, (1991) "UIDE - An Intelligent User Interface design Environment" In Sullivan and Tylor (eds.), *Intelligent User Interfaces*, ACM Press, 1991, pp.339-384.
- R.Hartson, P.Gray, (1992) "Temporal Aspects of Tasks in the User Action Notation" *Human Computer Interaction*, Vol.7, pp.1-45, 1992.
- ISO (1988) *Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on temporal Ordering of Observational Behaviour*. ISO/IS 8807, ISO Central Secretariat.
- P.Palanque, R.Bastide, V.Senges, (1995) "Task model - system model: towards an unifying formalism", *Proceedings HCI95 International*, Japan, September 1995.
- F.Paterno', M.Mezzanotte, (1995) "Formal verification of undesired behaviours in the CERD case study", *Proceedings EHCI'95*, pp.213-226, Chapman & Hall, 1995.
- F.Paterno', C.Mancini, S.Meniconi, (1997) "Understanding Task and Software Architecture Relationships", *CNUCE Internal Report*, February '97
- P.Szekely, P.Sukaviriya, P.Castells, J.Muthukumarasamy, E.Salcher, (1995) "Declarative Interface Models for User Interface Construction Tools: the Mastermind Project", *Proceedings EHCI'95*, pp.120-150.
- Sears, A.. (1995) "AIDE: A step toward metric-based user interface development tools". *Proceedings of UIST'95*. ACM Press, pp.101-110.
- S.Wilson, P.Johnson, C.Kelly, J.Cunningham, P.Markopoulos, (1993) "Beyond Hacking: a Model-based Approach to User Interface Design, *Proceedings HCI'93*.