Lorenzo Cioni

User Interfaces and Multiple Activities: The User Perspective

(paper accepted at the CALCO '94 "Human Factors" Conference, Flagstaff, Arizona, 14-18 March, 1994)

Abstract.

This paper contains some remarks about the use of window systems and menu based user interfaces as supports for multiple and concurrent activities, activities composed by tasks and actions. The main drawbacks of such systems are examined. Afterwards we discuss the definition of a working set as a tool that allows the user to carry out more than one of its activities by using a computing system. The closing sections of the paper contain some reflections on the improvements to be made to the aforesaid systems so to implement the concept of working set.

1. Introduction: user activities.

People usually act on the basis of concurrent activities. Whenever a computing system comes into play there is the need to map on it such activities. Computers, indeed, work on the basis of programs and there is hardly a one-to-one correspondence between an activity and a built-in program. Consequently, computing systems need to be supplied with some tool that supports the establishing of the correct connections among programs to allow the development of an activity.

Moreover, people interleave their activities ([Cypher 1986]) and often perform them on a step by step basis. Interleaving results either because of long lasting activities or because an activity may consist of a set of concurrent tasks with real time demands.

Therefore, users need a support to decompose an activity on the existing programs through the definition of a set of co-operating programs called working set that can be created, destroyed, saved or stored as needed so to allow either the suspension or the resumption of any activity or task.

This requires the introduction of the concept of context as the actual status of a task, its sub-tasks and its correlated tasks together with any shared data. Suspension needs the saving of the proper context.

In this paper window systems and menu based user interfaces are examined within this perspective. The thesis is that they support multiple processes far better than multiple activities. This implies the need to enrich such environments so to help users in carrying out easier their activities. Besides supports for easy suspension, reminding and resumption the environments, therefore, need to be supplied with tools for the definition and management of contexts as integral elements of a user interface (UI).

2. Activities, tasks and actions.

In this section we state some relations among the elements we are dealing with, i.e. activities, tasks and actions. An activity can be seen as a situation in which a lot of things are happening or being done whereas a task is a piece of work that must be done and an action is something that somebody does on a particular occasion. It should be clear that we have a decreasing complexity together with an increase of the constraints. An action must indeed satisfy the constraints of the physical system on which it is executed more strictly than a task or an activity. In our field of interest the physical system defines a set of available actions, such as a mouse clicking or the selection of an entry in a pop-up or pull-down menu, that a user executes through the filter of the UI in order to perform a task and within a certain activity. A task is therefore composed of several actions within a single sequence (a stream) or within various concurrent streams. On their turn tasks can be carried out either sequentially or concurrently , depending on the presence or the absence of dependencies and/or constraints among them: their course can occur within an activity or in relation with some other group of tasks. It is worth noting, indeed, that a task can be seen as either a sub-task (whenever we have a client÷server relation between two tasks) or a real task (i. e. when we consider it within an activity) or, in the end, as an activity or a sub-activity (for instance, this happens whenever its complexity rises with time or if an activity is formed by few tasks, even only one). Anyway, an action can never be seen as a task.

As to the activities, an activity can be seen either as a structured entity (composed of tasks and sub-activities) or as a single entity that may be part of a more complex activity. In the former case we have to describe the relations among the simpler composing elements whereas in the latter case we consider an activity as a black box: the only data we need are what are its outcomes and how long it is likely to last. This approach, for instance, can be pursued during the planning phase. The presence of relations and time constraints accounts for both step by step and interleaving ([Cypher 1986] and [Cioni 1993]) and can be explained in terms of either transition networks or an event model or an object oriented model ([BroMa 1990]). Our approach is to use an hybrid model base on the concept of semantic network: whenever a user starts or plans an activity/task (see below) he/she defines a semantic network (directed graph) or a semantic node (black box) within a semantic network.

3. *Some remarks on User Interfaces.*

Every computing system, however old-fashioned it may be, is characterised by a UI. Such interfaces, essentially textual ones, for a long time have been nothing more than a barrier between the users and the system ([Cioni 1993]) and only recently, with the introduction of graphic UIs, have evolved towards systems for the handling of the dialogue with the users. In this direction is to be seen the very popular paradigm for the design of user friendly UI (the so called desktop interface), fully and clearly described in [Apple 1987] and embedded in a very diffuse family of computers, and that influenced, for instance, the design of the Windows™ operating system. A deeper examination of such UIs can be found in [Cioni 1993]. In this paper we restrict ourselves to some aspects of such UIs and particularly the use of menus and windows. Menus are often claimed to be a facility for the users (point and click instead of remember and type) but they suffer two major drawbacks: there is no easy (i.e. without any need to be a skilled programmer) way for a user to add a new entry or to modify the semantic of an existing one (unless within a strict and pre-defined range of possibilities) and there is no easy way to allow menu entries to co-operate (apart from the trivial *copy&paste* or *command&undo-command*). This lack of co-operation is true even among distinct application and in presence of more concurrent processes with which a user can interact through distinct windows. Windows, indeed, represent screens within screens and allow a user to have several views of one or several files or processes (or applications) running at the same time: in this case windows reflect only a characteristic of the underlying Operating System more than the fact that a user has started some structured activities that are founded upon co-operating and (even) customised applications. Activity planning and administration, within this situation, is a duty of the user.

4. *User activities and User Interfaces.*

The characteristics of user activities, as discussed in 1, do not seem to find any correspondence with the structure of the nowadays available UIs. The presence of applications, each with its own menus and windows, tend to force the users to think that the only conceivable tasks are those represented by the available commands and that within an application only certain activities can be carried out: in these situations if a menu entry is dimmed or the current application does not fit the scheduled activity an user is unlikely to find by him/herself an alternative.

Moreover, the coexistence of several processes, each with its own windows, may clutter the physical screen: in such a situation can be hard for a user to remind which processes are associated within an activity, which processes are able to exchange data (in case of a Macintosh™ this must be explicitly done by the user through a single clipboard), what is the current state of each process whenever he/she resumes a task or an activity and so on. In such cases the interface controls the interaction with the user, lets him/her execute the proper operations (and undo the last one) and give him/her fixed ways to exchange data among applications. There is no way for a user to go back over a sequence of commands so to understand the current state of his/her activity/task or to execute the same sequence on distinct data. This is true for a purely menu-based UI (such that of Macintosh). It is our opinion that, apart from an exterior friendliness, such UIs are more closed that the old ones and, in conclusion, they give him/her less freedom of action. Though this approach can be correct in case of novices it proves really limiting for more and more expert and sophisticated users. Actually the only answer to these needs has been the introduction of keyboard shortcuts and the design of complex applications (a new and harder barrier for novices).

5. Introducing context and semantic networks.

A *semantic network* (Snet) is composed by semantic nodes linked through arcs that represent relations and dependencies among them. A *semantic node* (Snode), in its turn, can coincide with one of the available programs or can hide an Snet (bottom-up, design) or can be expanded in an Snet (top-down, implementation) or both (middle-out, planning & development).

An Snet formally describes the concept of working set but does not define by itself a one-to one correspondence between a task/activity and one of the available programs: each time the user starts using a program he/she must declare it as belonging to a working set. Moreover he/she must state which are the other co-operating programs, so to establish relations, and which are the shared data, so to establish constraints. Such dynamic data should be modified whenever needed so to allow the addition or the exclusion of any program, relation and constraint.

With the term *context* we, therefore, define a dynamic abstract structure that contains all the data that code an Snet, its structure and the history of its evolution, together with the possible relations among distinct Snets (see below). A context, by definition, can contain and can be part of other contexts. During execution, inner contexts takes precedence, the converse during design or planning. Each activity has a context and a context can be defined even among distinct activities, so to optimise the use of physical resources and to reduce the number of concurrent processes. We have Snets and Snodes to which there correspond contexts recursively nested: the simplest context is associated to a single program and can be described by a journal file (in case of an application) or a status variable (in case of an operating system command).

These abstract structures need to be created, stored, recalled and destroyed whenever needed so to allow a user to concentrate on his/her own activities/tasks instead that on the administration of programs and files. This requires the need to manage persistent objects that need also to be implemented on a computing system: the approach we are thinking to adopt is the object oriented approach.

6. User Interfaces as a support for multiple user's activities.

At this point it should be clear that a UI in order to be both evolutionary and really user friendly should provide users with supports for easy linearizing, suspending and resuming any activity or task (an action can be interrupted but neither suspended nor resumed). Linearizing ([MiNo 1986]) means to organise many parallel tracks of activities into a single stream of actions to be performed and interacts with both interleaving and the step by step style since interleaving implies the presence of currently acted on and suspended activities/tasks and the existence of relations among activities/tasks accounts for the step by step style. From this results the need of a support for suspension (i.e. the ability to put a task in a suspended state and to switch to another one), reminding and resumption (the ability to start again executing tasks/actions within a suspended activity/task at appropriate time, i.e. when all its constraints have been fulfilled). Reminding is important since a user must always be aware of both the presence of suspended activities/tasks and of their status. The definition of a semantic network among applications thus allows a user to execute his/her activities under the control of a sort of supervisor.

After the user has defined the initial structure of the Snet, its links with other Snets the administration of the transactions is no more a duty of the user. In this way he/she can execute applications and commands to carry out tasks and activities: whenever an application is quitted its context is saved, whenever a new application is started its context, if any, is recalled and added at the current activity's context and so on. Moreover this enriched UI handles the opening/closing/saving of files and of shared data (whenever an activity/task is suspended) and prevents accesses to the data of an activity without the activation of the proper context, so to avoid inconsistencies. Snets, therefore, can support suspension and resumption whereas reminding is founded upon both the UI (that ca use icons or the like) and the higher level contexts. They therefore should be added to a UI so to satisfy users' needs.

7. *Conclusions.*

The present paper represents a refinement of some parts of [Cioni 1993] and a further step forward the definition of an evolutionary UI. With this term we mean a UI that is able to evolve together with the user's expertise and sophistication. Moreover such a UI should allow a user to develop easily and satisfactorily his/her activities on a computing system.

As already pointed out in [Cioni 1993], such an evolutionary UI should be based on the use of visual languages and on the embedding of the capabilities of an expert system within the interface. The area of the expert systems, indeed, seems to be worth investigating for, at least, one basic reason: an expert system could simplify both the bootstrapping phase (in case of novice users) and the solution of problems, even in case of skilled users. We note, indeed, that the use of on line help facilities, apart from being boring, assumes that the user either has some basic knowledge of both the computing system and of the mapping of the current problem domain on the computing system or can get hints from users that master such knowledge. The investigation of the potentialities of expert systems as embedded within a UI is the subject of the author's future research in this field.

Bibliography.

[Apple 1987] Apple Inc. (1987), *Human Interface Guidelines: The Apple Desktop Interface,* Addison-Wesley Publishing Company.

[BroMa 1990] Bronzini E. & Mazzotta S. (1990), *Tecniche di programmazione visiva applicate ai Geographical Information Systems*, in Italian, Master Degree Thesis, Università degli Studi di Pisa, Facoltà di Scienze Fisiche, Matematiche e Naturali, Corso di Laurea in Scienze dell'Informazione.

[Cioni 1993] Cioni, L. (1993), "User interfaces: a linguistic approach", paper accepted at the Second National Linguistic Congress, August 23-25 1993, El Colegio, Mexico City, Mexico also published in Quaderni del Laboratorio di Linguistica, vol. 7, Scuola Normale Superiore.

[Cypher 1986] Cypher, A. (1986), "The structure of users activities" in D. A. Norman, ed. , *User Centered System Design, New perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, pp 243-261.

[MiNo 1986] Miyata, Y. & Norman D. A. (1986), "Psychological Issues in Support of Multiple Activities", in D. A. Norman, ed. , *User Centered System Design, New perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, pp 265-284.