A Comparison of Approaches for Specifying MultiModal Interactive Systems

Joelle Coutaz[1], Fabio Paterno[2], Giorgio Faconti[2], Laurence Nigay[1]
L.G.I. - Grenoble (1) - CNUCE - Pisa (2)
Email: Joelle.Coutaz@imag.fr, paterno@vm.cnuce.cnr.it

Abstract

This report discusses possible approaches to the specification of MultiModal Interactive Systems. Two notations which are representative of different but convergent areas are considered and their complementary use in the design and analysis of Interactive Systems is proposed. The remarks we make are raised by a case study consisting in the specification of a system for interacting with a flight data base by voice and graphical devices.
Keywords: Formal Notations, MultiModal Interactive Systems, Interaction Objects.

1. Introduction

This article outlines possible approaches to the specification of MultiModal Interactive Systems deriving from some research developed within the Amodeus II BRA Project. One of the themes considered by the System Modelling Research Package is the design and specification of MultiModal Interactive Systems.

Since different backgrounds are present within the team (designers of MultiModal Systems, formal specifiers, experts in graphics systems) this seemed like a good opportunity to mix them in order to explore new problems. One of these was to specify MultiModal Interactive Systems. The motivations for this goal are to have a precise description of their functionalities. This entails using rather different modalities, either separately or combined. This is useful for both users and designers: for example, the former can receive answers to questions related to how to fulfil given tasks, the latter are constrained to clarify their design in the refinement process towards implementation.

While the application of formal notations is widely used in different areas of software engineering and has already received consistent attention in the application to computer graphics [D91] and in the abstract description of human-computer interaction [HT90], there are no relevant experiences in the application to multimodal interactive systems.

We therefore consider two notations from different areas: UAN [HG92] developed at the Virginia Polytechnic Institute explicitly for describing human-computer interaction, and LOTOS [BB87] which was mainly designed for the specification of communication protocols but which was shown to be useful for specifying any class of systems where concurrency, interaction, reaction to external events, and nondeterminism are relevant aspects. The case study for evaluating the suitability of the considered notations and the specific requirements of this class of systems was Matis [CNS92], a MultiModal Interactive System physically available in the workshop, which specifies requests to a data base for flight reservations.

Another problem we consider is how to obtain the modelling of the corresponding system from the performed specification by using a specific model for interaction objects such as PAC [C87] or Interactors [PF92].

2. UAN

The User Action Notation (UAN) is a task-oriented notation that describes the behaviour of the user and the interface as they perform a task together. Its primary abstraction is the user task. It is a notation that supports task decomposition and refinement of each basic task into the sequence of user actions which can perform it. The resulting user interfaces are represented as a quasi-hierarchical structure of asynchronous tasks, the sequencing within each task is independent of the one in the others. We refer to this notation as described in [HG92].

There are some predefined symbols for common user actions such as moving the cursor in the context of a specific graphical object (*~[X]*) or depressing (*£*) and releasing (*¢*) of the button of the mouse; or for common system feedback such as highlight (*!*), dehighlight (*-!*), display (*Display(X)*) and cancel (*Erase(X)*) an object. Other symbols in the notation are: * which means iterative closure (task is performed zero or more times) and + which indicates that task is performed zero or more times.

Each task is described by a three column diagram with one column for the description of the user actions, another one for the interface feedback, the last one for the state of the Interface. The described tasks and/or actions are performed top to bottom, and left to right. We thus get an immediate representation of the user-generated interactions and the related effects in terms of system feedback and modification of state. Each task can be associated with a precondition which has to be verified before it can be performed.

The notation also considers time-related aspects. Time is described as a one-dimensional quantity, made-up of points, where each point is associated with a value. The points are ordered along the dimension by their values. Different constructs for temporal relations are available:

A B, *Sequence*, tasks A and B are performed left to right, or top to bottom;

A (t > n) B, *Waiting*, task B is performed after a delay of more than n units of time following task A;

(A | B)*, *Repeating Disjunction*, choice of A or B is performed to completion, followed by another choice of A or B, and so on;

A & B, *Order independence*, tasks A or B are order independent (the order of performance is immaterial)

A --> B, *One-way interleavability*, task A can interrupt B and execute, but not vice versa;

A <--> B, *Mutual interleavability*, task A and B are mutually interleavable;

A ÷ B, *Concurrency*, task A and B can be performed concurrently.

As an example we consider the UAN specification of the task of building a request R in Matis:

*((SelR(R) | ClrR(R) | HideR(R))**
--> SpecR(R)+)
Submit(R)

It consists of specifying the request at least once (thus, SpecR $(R)^{+}$). This task can be interrupted 0 or more times by either one of the following atomic tasks: select a request, clear the request, or hide (iconify) the window request ((SelR (R) | ClrR (R) | HideR(R))* →). Once the request has been specified, it must be submitted (Submit(R)).

3. LOTOS

LOTOS is a specification language developed within the International Organization for Standardization (ISO) environment in order to specify network protocols. Its features make it suitable for describing concurrent systems such as Interactive Systems. It combines the process algebra inherited by CSP and CCS with the data algebra provided by ACT ONE. Basic LOTOS is considered to be only the concurrent part of the notation.
The basic idea of LOTOS is that systems can be specified by defining the temporal relation among the interactions that constitute the externally observable behaviour of a system. A system is seen as a set of processes. Each process may interact with its environment via interaction points called gates, i.e. it performs observable actions at the gates and can also perform unobservable (internal, hidden) actions. An observable action consists of offering or accepting zero or more values at a certain gate. An interaction may occur when two or more processes are ready to perform the same observable action. It may involve data exchange and is an instantaneous event (synchronous communication).
Process behaviours are described by algebraic expressions, called behaviour expressions. Complex behaviours are expressed by composing more simple behaviour expressions (subprocesses) via the LOTOS operators such as sequentiality (P>>Q); parallel composition (P|||Q); disabling (P[>Q). In a process definition the specifier has to indicate the process identifier, a formal parameter list, a behaviour expression (and the definition of the data types and the processes that it uses). The main operators in a process behaviour definition are:

a;B, *action prefix*, means that the process can only perform and then behave like B;
B1[]B2, *choice*, this means that the process can act as B1 or as B2;
hide g1, ..., gn in B, *hiding*, is a process which can perform any action of B which does not make use of gates in (g1, ..., gn), any action occurring at one of these gates is hidden and transformed into an **i**-action
[e] -> B, guarding, is realized by applying the boolean expression *e*, if it is verified, B is executed.

When a composition between two processes has to be defined there are three possibilities: *interleaving*, denoted by A|||B which means the processes are independent of each other and any interleaving of their actions is possible, thus the two processes never synchronize; A|[f,g,h]|B that explicitly indicates the gate where the independent processes have to synchronize;
full synchronization, A||B, where the two composed processes are forced to proceed together.

It is also possible to realize: sequential composition (enabling), B1 >> B2, where the behaviour of the process B2 is enabled if and when the behaviour of the other process successfully terminates, and disabling, B1[> B2, where the behaviour of the process B1 can be disabled at any time (except after successful termination) by the realization of an action of the other process.
The process can be communicated in various ways, the most important are:
- value passing, if we have process A realizing a!3; and process B a?x:int; the result is that the value 3 is passed to B in the variable x.
- signal matching, for example process A realizes a!x1; and process B a!x2; the two processes sharing an event gate may attach signal values to events and the corresponding events take place if these values coincide (x1=x2).

4. A UAN Specification of Matis

In [CP93] there is a complete UAN description of Matis. The specification consists of 38 tasks and is 22 pages long. In order to make it more comprehensible we performed a simple graphical representation of the task decomposition. In the resulting tree we have a node for each task, and the children of one node are the tasks which are present in the father definition in both the user action or system feedback columns. This is useful in order to have a compact description of the global logical structure of the specification. The following figure represents the result.

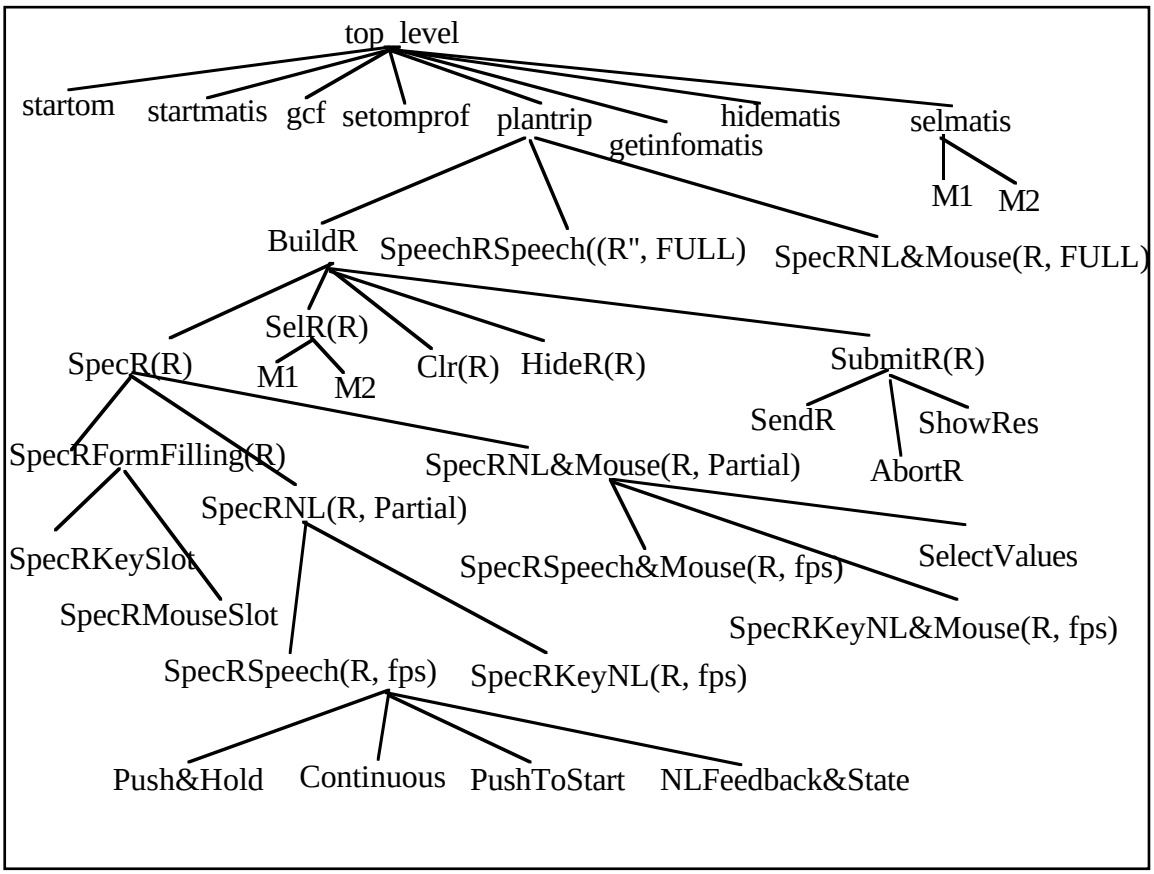

Figure 1: The tree-structure derived from the UAN specification.

In order to give a more precise idea of the UAN specification of Matis here is an example of a task description. We considered the task SpecRSpeech&Mouse which allows the user to specify a request for the data base by using the voice and mouse devices. In the example the third column is not considered because in this case the Interface state is not modified. Task SpecRSpeech&Mouse is similar to SpecRSpeech but sentences contain at least one deictic expression, and deictic expressions must be solved with the mouse.

Task: SpecRSpeech&Mouse (R, fps) is atomic		
User Action	Interface Feedback	
SelAppli = MatisAppli ∧ SelR = R: case SpeechMode = Push&Hold ~[x,y in OMEarW]∨ ProduceDeicticSentence (sentence, fps,) ∧ SelectValues	 OMEarW! *(Listening)* NLFeedback&State(sentence,)	
SpeechMode = Continuous ProduceDeicticSentence (sentence, fps,) ‖ SelectValues	 ($t > t_{silence}$) NLFeedback&State(sentence,)	
SpeechMode = PushToStart ~[x,y in OMEarW]∨∧ ProduceNonDeicticSentence (sentence, fps,) ‖ SelectValues ~[x,y in OMEarW]∨∧	 NLFeedback&State(sentence,)	
endcase		

Figure 2: An Example of UAN Specification of a task supported by Matis.

The speech device can be used in three modes (Push&Hold, Continuous and PushToStart). When SpeechMode is Push&Hold, the mouse is used to inform the speech system that the user is currently speaking. As a result, the selection of slot values on the screen must be performed in sequence once speech input is over (this is an example of the sequential use of input devices). For the other speech modes, values can be selected in true parallelism with speech input.

5. An Evaluation of UAN

A formal description of the user interface, as a UAN specification, opens the way to automation of usability tests. For example, through the UAN exercise for Matis, we have identified three rules that could be embedded in an automatic UAN-based usability test tool:

If preconditions for a task execution cannot be expressed in terms of user interface feedback, then the observability principle is broken. These are conditions which cannot be expressed in terms of user interface features. As a result, the user is not provided with any feedback about the current value of an internal system state variable that is relevant to the task.

If, for identical sequences of user actions and identical system feedback, feedback occurs at different points in the user sequences, then system feedback triggering is not consistent. For example, in one sequence, mouse down triggers a feedback such as a reverse video while for the same sequence in a different task, the same feedback is produced on mouse up only. This may or may not correspond to a sound design decision.

If a feedback object exists as one instance at most, and if it is mapped by the system at different locations on the screen, then the system layout may be inconsistent. For example, in Matis, the RequestTools window is always mapped at the same location. There is only one instance of such a window during a Matis session. On the other hand, request forms are distinct instances of the request form class. They may be mapped by the system at distinct locations to avoid, for example, overlapping.

The limitations of UAN are related to the lack of a clear semantics and to some deficiencies in the power of expression. As for semantics, our main concern is the temporal relationships between the descriptions in the user action, the interface feedback and the interface state columns. The first problem is a lack of a definition for the notion of statement (the spelling of one statement may physically cover several rows). The second problem is that the notation is needed to express scope and relationships such as parallelism or sequentiality between the columns.

For example, in the description of task StartMatis, R1=NewR(NbR) of the interface state column must be executed before Display (Form(R1)) specified in the interface feedback column. On the other hand, in task StartOM, state change of OMICON can occur at the same time as the modification of the interface state. In SetOMPref, the interface feedback is implicitly comprised of two sets of reaction depending on the conditions of the user action column. Nothing in the notation makes this explicit. A similar problem arises in task ExitForm with the | operator. In this example, we have repeated the | operator in the interface state to visually increase the mapping with the behaviour described in the user action column.

We have encountered a number of difficulties related to expressiveness. Some of them are listed below : absence of a kill/break operator, no notion of default task, no attribute to denote the actor of a task, lack of programming facilities such as control statements (e.g., conditional and case statements), procedures, macros, scope, etc.

If and case statements may be replaced by the | operator. Clearly, this solution leads to chunking. We have not analysed yet whether this chunking would have negative consequences or would positively reveal a decomposition that might be psychologically valid or that would be useful for the software architect. From our early experience with

Matis, we have the feeling that the absence of control statements leads to artificial chunking that impedes the readability of the description. As a result, we have introduced case statements and conditional statements. Scoping is not considered in UAN. Specifically, variables that sustain the specification seem to occur throughout the whole description. As a result, the state of the user interface is described as a single global entity. It is useful however to be able to structure a description not only in terms of tasks and actions, but also in terms of programming language constructs. A potential benefit for the software designer would be to derive subcomponents such as interactors or agents, for example from the scope of variables. A set of variables with the same scope may correspond to the local state of some interactor or agent.

6. A LOTOS Description of Matis

We have used LOTOS to specify several systems. In this case it was used to specify Matis at different abstraction levels and by using the interactor model for designing the architecture of this system. We follow the interactor model as described in [PF92]: an abstraction for describing interaction objects which can perform input-output processing in both the application and the user side. The model was previously used to describe Interactive Graphics System. Now we want to explore its suitability for MultiModal Interactive Systems.

One advantage of LOTOS is the availability of a set of automatic tools (LITE [E91]) which support different types of processing: automatic detection of the correctness of the specification with respect to syntax and static semantics of the notation, interactive simulation of the actions which can be performed, automatic verification of properties expressed by using Action-Based Temporal Logic (in [P93a] examples of properties related to single interaction objects and logical integrity of User Interface Systems are provided, [P93b] describes examples of user-oriented properties).

One open problem is how to refine the design of an interactor-based architecture of an Interactive System from a UAN specification. The process of deriving an interactor-based interactive system from a UAN specification can be decomposed into two sub-problems: identifying a set of interactors and indicating how to compose them. In [P93c] two possible approaches are described: one approach, bottom-up, associates basic tasks with interaction objects and then tries to compose them in order to obtain the corresponding interactive system; the other approach is top-down as both interactors and tasks are concepts which can be applied at different levels of abstraction, and it refines interaction objects by reflecting the task refinement performed.

In this modelling process our approach tries to associate basic tasks (tasks which are defined in terms of actions rather than other tasks) with the interactors which describe the corresponding implementation. It also attempts to use the temporal operators (parallelism, interleaving and so on) among tasks, and among the corresponding interactors as well. Another element is to use the task decomposition for refining the corresponding model of the system.

The interactor model can be used at very different abstraction levels. In our case we can start from a one-interactor view of Matis: in this case the input channels from the user are the physical devices available (Mouse, keyboard and microphone) the outputs toward the application are requests to the data base and the inputs from it are the result of the user-requests. Then we can have an initial refinement by the subtasks identified for the build request task (BuildR) which is the main task for MATIS. We can thus see the build request task composed of the specification request task, the clear request task and the

submit request task. We could associate an interactor with each of these tasks. The result of the submit_request interactor is like an input trigger for the specify_request interactor because when it occurs the latter sends a data to the application. On the other hand, the result of the clear_request interactor is like an input to the input part of the specify_request interactor. When the latter receives an input from the clear_request interactor it clears its appearance (the indication of the previously specified values of the current request) as a feedback of the received input.
If we consider the specify_request interactor further refinements are possible (Figure 3): we can obtain an interactor associated with the request (R), one with the request form (GF), one with the recognizer (Rec), one with the request tool (RT), one for each window tool that it can activate (we consider only one, TW, for simplicity), one with the ear button (ear), one with the send request button (SR), one with the clear request button (CR), one with the window for providing requests with the keyboard (Win), and one for the window providing the result of a request (AW). The request interactor (R) provides an output to the request form which visualizes the values provided, and an output to the application which indicates the request selected by the user. The request interactor can receive input data from the graphical form or the clear request button or the recogniser. One possible corresponding interactor-based architecture is in Figure 3.

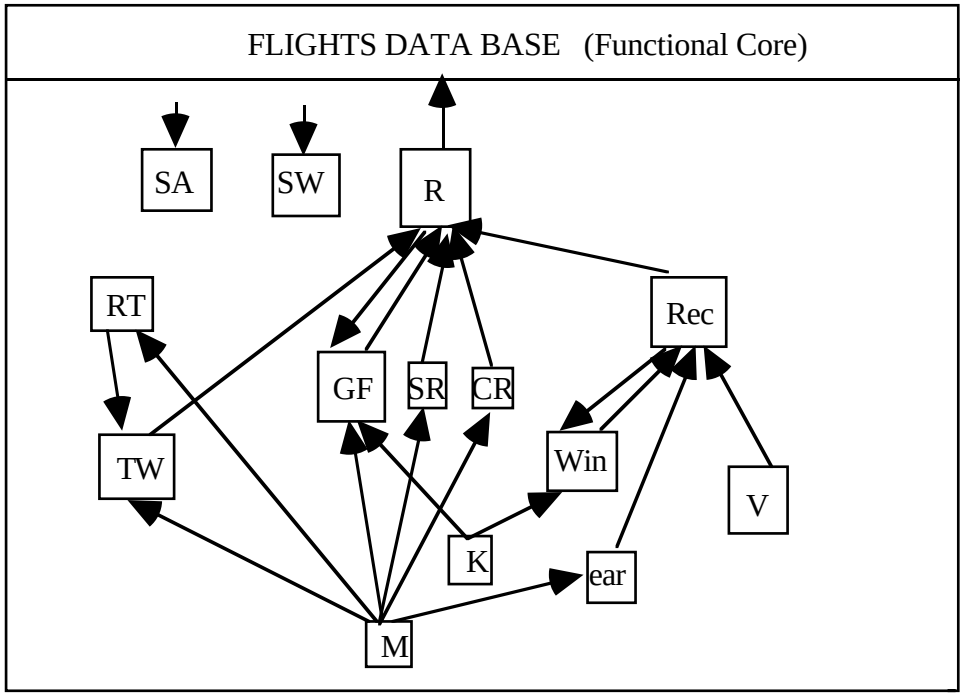

Figure 3: An Interactor Based Description of Matis.

The task decomposition provided by a UAN specification is a useful indication of the logical and temporal connections among interaction objects which perform the related tasks. Some general heuristic rules found in the exercise include:

If a task can indifferently be refined into two interactions (sequences of user actions and system feedback) then it can be performed by three interactors: one for each

interaction and one to gather the data which can be generated in one of the two possible interactions;
If one task is associated with only one action it should not have an entire interactor for itself;
Interactions with a window or with the related icon are described by the same interactor;
If the feedback of several tasks is related to the same graphical entities they should be refined into the same interactor or in communicating interactors.

7. A Comparison of LOTOS and UAN

It is interesting to compare LOTOS and UAN because they come from different areas (the former from software engineering, the latter from human-computer interaction) but they are based on similar concepts. Indeed, both are concurrent notations and use actions: LOTOS describes a system by its externally observable actions, UAN describes human-computer interactions in terms of user actions and system feedback, and these two groups of actions are structured in asynchronous tasks. We can notice that there are functionalities in UAN which LOTOS does not have, and viceversa.
In fact, UAN includes constructs such as one-way interleavability (a task interrupts another one which will be continued at its completion); waiting (a task is performed after a delay of a specific number of units of time); and true parallelism which LOTOS does not support. In fact, the current version of LOTOS does not provide time-dependent constructs and its semantics is an interleaving concurrency, although extensions in these directions are being investigated. On the other hand, LOTOS has a deactivation operator (an operator that takes two processes, when an action of the second one is performed the first is deactivated). It explicitly allows us to indicate on which subset of gates two or more processes can synchronise, it supports message passing among processes, it has operators which indicate when a process finishes its processing and then it may pass the control to another process: exit (which is very similar to the break operator introduced in the UAN specification of Matis) and stop, respectively. Constructs of LOTOS were more formally defined: their operational semantics was provided in terms of Label Transition Systems. In UAN there is a different approach to defining the semantics of the constructs of the notation: time is a one-dimension quantity, made up of points, where each point is associated with a value and it uses intervals of activity associated with each task to describe operators among tasks.
While in UAN operators can be applied indifferently on tasks and actions, in LOTOS processes (it would be more precise to say their behaviour expression) and actions can be associated with different operators: for example disabling and sequential composition need two behaviour expressions while action prefix requires one action and one behaviour expression. Another difference is that LOTOS distinguishes between internal and external actions: the internal actions allow the system to change its state but they are unavailable for synchronization with other processes.
LOTOS processes may have their own state and it is explicitly indicated if they synchronize with other processes, on which gates they can synchronize, and if they perform value passing. In UAN there is only a generic interface state. Tasks cannot synchronize with each other and do not perform value passing: it is only possible to indicate the ordering of their corresponding user actions and system feedback. The interface state and its modifications are described informally, while the state of LOTOS processes is described by the ACT ONE notation for algebraic data types.

Conclusions

In this report we have described the first results of some current research on the specification of MultiModal Interactive Systems. These results seem to indicate a complementary use of the two considered notations: UAN as a notation to evaluate, especially from a usability point of view, existing software and as documentation for users who want to know how to perform specific tasks; LOTOS and related tools as a notation for design, specification and verification of the software related to the development of Interactive Systems.

Acknowledgements

This work has been supported by project ESPRIT BR 7040 Amodeus II.

References

[BB87] T.Bolognesi, H.Brinskma, "Introduction to the ISO Specification Language LOTOS, Computer Networks and ISDN Systems, Vol.14, pp.25-59, 1987.

[HG92] H.R.Hartson, P.D.Gray, "Temporal Aspects of Tasks in the User Action Notation", Human-Computer Interaction, 1992, Vol.7, pp.1-45.

[C87] J.Coutaz. "PAC, an Object Oriented Model for Dialog Design". Proceedings Interact'87, pp.431-436.

[CNS92] J.Coutaz, L.Nigay, D.Salber, "MATIS: A multimodal airline travel information system". Technical Report SM/WP10, ESPRIT BRA 7040 Amodeus-2, Februray 1993.

[CP93] J.Coutaz, G.Faconti, F.Paterno, L.Nigay, D.Salber, MATIS: a UAN Description and Lesson Learned, Technical Report SM/WP14, ESPRIT BRA 7040 Amodeus-2, February 1993.

[D91] D.Duce, Formal Methods in Computer Graphics, Computer Graphics Forum, Vol.10, N.4, pp.359-360.

[D93] D.Duke (editor), "System Modelling Exemplars", Technical Report SM/WP12, ESPRIT BRA 7040 Amodeus-2, March 1993.

[E91] P.van Eijk, The Lotosphere Integrated Environment, proceedings 4th International Conference on Formal Description Techniques, (FORTE91), Sidney, November 1991, North Holland, pp.473-476. 1991.

[HT90] M.Harrison, H.Thimbley, Formal Methods in Human-Computer Interaction, cambridge University Press, 1990.

[PF92] F.Paterno', G.Faconti, On the LOTOS Use to Describe Graphical Interaction, Proceedings HCI Conference 1992, pp.155-173, Cambridge University Press.

[P93a] F.Paterno'. "Definition of User Interface Properties Using Action-Based Temporal Logic". Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering, pp.314-319, S.Francisco, June 1993.

[P93b] F.Paterno. A Formal Approach to the Evaluation of Interactive Systems. Proceedings Workshop on Formal Methods for Interactive Systems, York, July 1993.

[P93c] F.Paterno', "A Methodology to Design Interactive Systems based on Interactors", Technical Report SM/WP6, ESPRIT BRA 7040 Amodeus-2, February 1993.