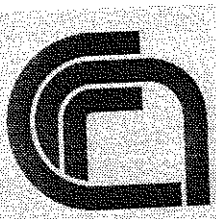

Consiglio Nazionale delle Ricerche

Verification of Properties of Human-Computer Dialogues with an Infinite Number of States

M. Mezzanotte e F. Paternò

CNUCE: C96 - 02

CNUCE

Submission to: *Formal Aspects of The Human Computer Interface BCS FACS Workshop*, Sheffield Hallam University, Sheffield, UK, 10th- 12th September 1996

Verification of Properties of Human-Computer Dialogues with an Infinite Number of States

Menica Mezzanotte & Fabio Paterno'

{menica, fabio}@giove.cnuce.cnr.it

CNUCE - CNR, VIA S.MARIA 36, 56126 PISA, ITALY

Abstract

This paper discusses the problem of verification of properties of user interfaces where the dialogue specification may have an infinite number of states. Recent techniques allow designers to get some results even in this particular case and we discuss when such results are useful for verifying the user interface properties.

Keywords: Application of formal techniques in user interface verification, modelling temporal aspects of interactions, model checking, user interface properties.

1. Introduction

There are many motivations to use formal notations: they allow designers to obtain not ambiguous descriptions of the desired functionalities, force them to clarify aspects of their design and to reason about the specifications performed.

A wide set of formal notations are available: they often differ for their expressive power, and the distinct aspects that they describe better.

It is important for the development of complex user interface to have automatic tools which allow designers to apply some transformations on the specifications and to perform verification of specific requirements and properties, for example usability properties.

The possibility of rigorous reasoning is one of the main advantages of the use of formal notations. This can be carried out either by model checking or by theorem provers. In the former case the specification represents the model against which properties can be checked, the latter allows to move from the theory provided by the specification to an underlying model in which the properties can be checked. Theorem provers are more hard to use.

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 human-computer interaction field is more challenging for verification methods and tools as the specification of human-computer dialogues can be more complex than hardware specifications.

Verification techniques are rapidly improving. They are able to address specifications with millions of states [BCMD92]. Some specifications are able to reach infinite number of states because of the dynamic nature of the related applications. This is the case of some human-computer dialogues. However some new verification techniques are being developed to address these cases and in this paper we discuss their application to user interfaces.

We mainly consider environments for process-based notations such as LOTOS [ISO88] and CCS [M89].

2. Related Works

The first results of the application of tool-supported, formal verification to check properties of basic interaction objects and user interfaces were described in [P93]. Since then other works have been carried out: Palanque and Bastide [PB95] use Petri Nets to specify Interactive Systems and to reason about them. Petri Nets is a powerful notation supporting parallelism and time but the specifications performed with them have a low modularity and it is difficult to modify them or to compose them. Some research work is being developed to investigate whether it is possible to integrate this approach with our TLIM method [PM95a]. Abowd and Wang [AWM95] have used tools similar to those used by us to verify properties expressed in CTL, which has the same power as ACTL but it describes state modifications rather than actions performance, and, in order to simplify the use of a formal notation they use the user interface of Monk's Action Simulator to express the properties which have to be verified.

We have applied our approach which consists in developing a LOTOS specification of the Interactive System considered and then to verify ACTL [DFGR93] properties by model checking to various case study. For example, we considered the user interface of an air traffic controller [PM95b]. In that case, we verified the possibility of undesirable events which, especially for that type of application, can have very serious consequences.

Recent work [BACL95] has been developed to apply the HOL theorem prover in order to verify mechanically specific requirements imposed on the user interface. Duke and Harrison have used modal action logic for reasoning about user interactions [HD96] but in that case there is not yet automatic support for reasoning. Johnson [JH92] used temporal logic as a means to specify and prototype user interfaces rather than to use it in order to reason about a more detailed specification.

3. Expressive power of Specification Notations

We can consider Interactive Systems as a subclass of Reactive Systems. In fact they share the main features of reactive entities: the continuous reaction to events externally generated. The main distinctive feature is that one component is a human being whose input devices are eyes and ears and output devices are hands and mouth. However while in computer science there is a strong tradition in comparing notations depending on their expressive power this is not part of the typical user interface designer background.

This is demonstrated, for example, by the use in many industrial sites of state transition diagrams to represent user interface dialogues. However, this type of notation can be used to describe very simple, not detailed examples. In fact, user interfaces, especially multimodal user interfaces, are characterised by the possibility to perform many actions in parallel and to activate and deactivate interaction techniques. These features are difficult to represent with finite transition diagrams. Viceversa process-based notations, such as CCS, CSP and LOTOS, are suitable to describe these features as they are mathematical models developed to capture in a immediate way the features of concurrent systems. In fact, if we want to describe the possibility of interleaving between two tasks, for example editing a file and printing a file, each one composed by the sequence of three actions, this is immediately expressed in LOTOS:

```
Task1[open_file, modify_file, save_file] ||| Task2[ select_file, select_parameters, print_file]

where
process Task1[open_file, modify_file, save_file]:noexit:=
open_file;modify_file;save_file;Task1[open_file, modify_file, save_file]
endproc

process Task2[select_file, select_parameters , print_file]:noexit:=
select_file; select_parameters;print_file;Task2[select_file, select_parameters,print_file]
endproc
endspec
```

The corresponding labelled transition system is very complex and it is represented in Figure 1. It was obtained by automatically transforming the previous LOTOS expression. The starting state is indicated by a double circle.

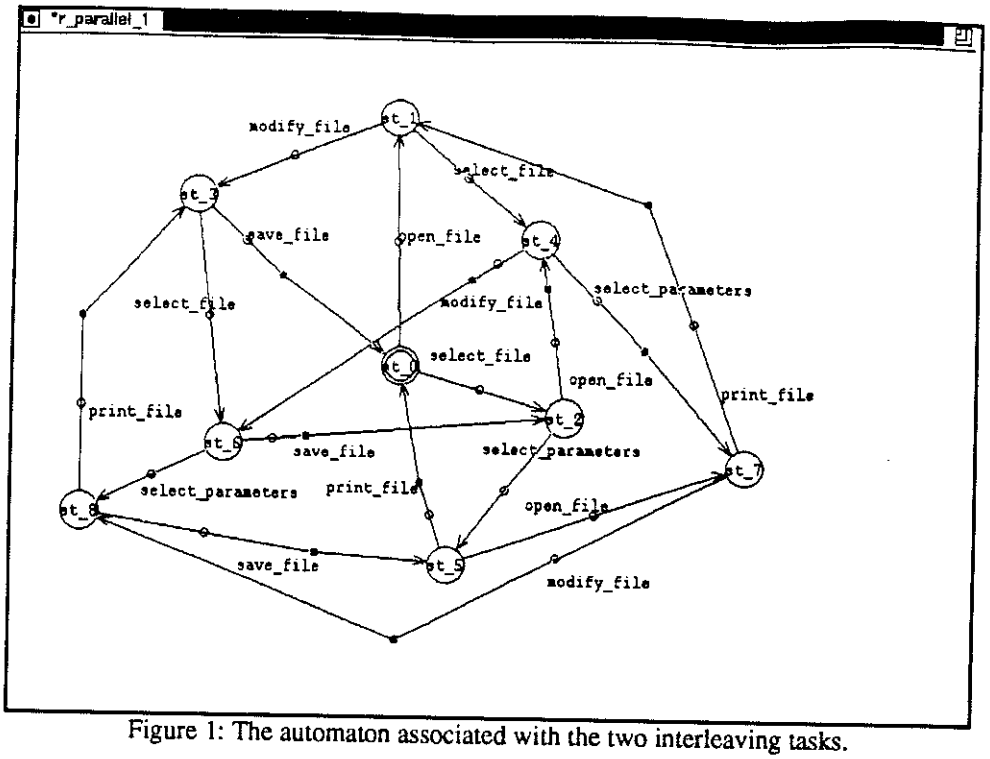

Figure 1: The automaton associated with the two interleaving tasks.

It is important to be aware of the expressive power of a notation both in order to understand whether it is able to describe the desired behaviour and to understand whether it is possible to reason about the specification. In fact in the second case if the specification corresponds to a model with an infinite number of states then verification of properties becomes harder.

In order to avoid this problem it becomes important to be able to understand whether specific subset of a given notation have different expressive power too in order to understand when they provide a specification corresponding to a finite state system and when this not happens.

For this purpose Fantechi and Gnesi [FG92] compared the power of different subsets of LOTOS operators. They show that it is possible to identify a set of LOTOS basic operators that has the expressive power of the regular languages, that is, a set of operators which generates a language whose expressions can be described by a finite state automaton. This set is composed of the basic LOTOS expression obtained with the operators: *choice*, alternative choice between two possible behaviours; ";" which is placed between two sequential actions; *stop*, which indicates that the process cannot be anymore involved in interactions; and *process instantiation*, which provides the definition of a process behaviour.

In [FG92] some other cases of use of LOTOS operators which produce expressions corresponding to finite state automaton are indicated.

4. Examples of Human Computer Dialogues which require different expressive power

We want to consider two simple examples of user interactions one which can be described by a finite state machine and the other which corresponds to a non-finite automaton.

Suppose we want model a simple interaction with a window containing text and the scrollbar associated with it. We have this situation:

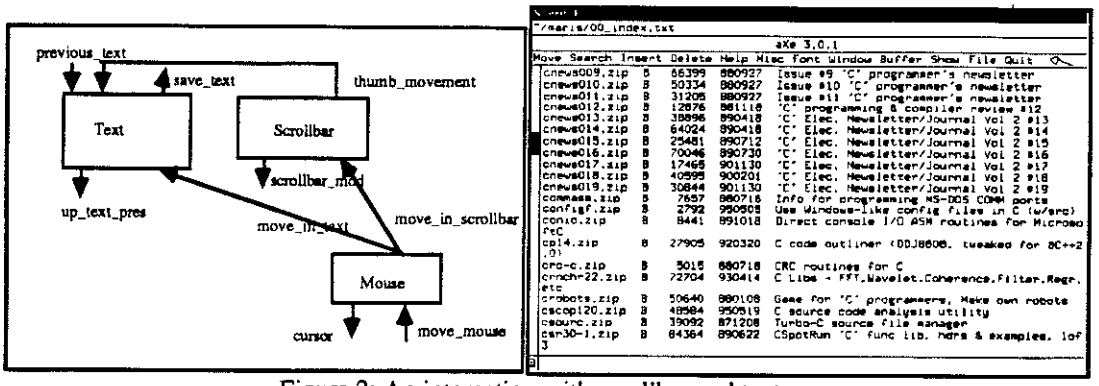

Figure 2: An interaction with scrollbar and text.

We describe the objects illustrated in Figure 2, using the LOTOS language, as follows:

process Object [.....]:noexit:=
(Mouse[move_mouse, ...,cursor]|[move_in_scrollbar]| Scrollbar[move_in_scrollbar, ... scrollbar_mod])
|[move_in_text, thumb_movement]| Text[move_in_text, ... , up_text_pres]

where
process Mouse [move_mouse, move_in_scrollbar, move_in_text, cursor]: noexit:=
move_mouse; (move_in_scrollbar; cursor; Mouse [move_mouse, ..., cursor]
[]move_in_text; cursor; Mouse [move_mouse, move_in_scrollbar, move_in_text, cursor])

endproc

process Scrollbar [move_in_scrollbar, thumb_movement, scrollbar_mod]:noexit:=
move_in_scrollbar; thumb_movement; scrollbar_mod; Scrollbar [move_in_scrollbar, ..., scrollbar_mod]
endproc

process Text[move_in_text, save_text, thumb_movement, previous_text, up_text_pres]:noexit:=
move_in_text; up_text_pres; Text[move_in_text, save_text, thumb_movement, previous_text, up_text_pres]
[] thumb_movement; up_text_pres; Text[move_in_text, ... , up_text_pres]
[] previous_text; up_text_pres; Text[move_in_text, save_text, thumb_movement, previous_text, up_text_pres]
endproc

This LOTOS expression is regular because it uses a set of operators which have been demonstrated to generate regular expressions. Thus the behaviour of this expression can be verified because it can be transformed in a finite state automaton.

The second example considers the case of a text editing application: an icon is active and when it is selected a window available to include the file to edit is activated. The initial icon has two buttons included: one to activate help on-line the other to exit the application.

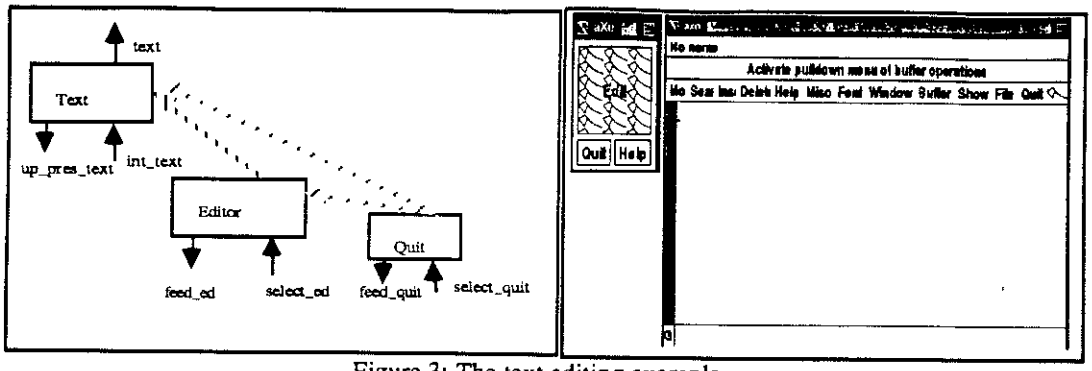

Figure 3: The text editing example.

In Figure 3 both interactor-based description and layout of the example are provided. Here the possible actions are: select the manager to open a text window (select_ed) , to close the axe manager (select_quit), and editing a text (int_text). In the figure on the left dashed arrows indicate enabling or disabling of interactors.

The behaviour of the interactors illustrated in figure 4 is described, using the LOTOS notation, in the following expression. You can note that when the Editor process terminates (exit action) it activates (by the enabling operator) itself in interleaving with a new interactor managing a text window.

```
Object1[ ... ] [>  Quit [select_quit, feed_quit]

where
process Object1 [...] :noexit:=
Editor [... ] >> Object1 [ ... ] ||| Text[int_text, ... up_pres_text]
endproc

process Editor [select_ed, feed_ed] :noexit:=
select_ed; feed_ed; exit
endproc

process Text[int_text, text, up_pres_text]:noexit:=
int_text; text; up_pres_text; Text[int_text, text, up_pres_text]
endproc

process Quit [select_quit, feed_quit] :noexit
select_quit; feed_quit; exit
endproc
```

This LOTOS expression has not the expressive power of a regular expression. This is because the process Object1 is defined in such a way that it uses an enabling operator (>> operator) and it appears on the right of the expression, so that it is called recursively , in interleaving (||| operator) with another process (the text process).
The presence of this situation means that we are not able to build an automaton associated with this behaviour and consequently, we cannot verify its properties.

This type of behaviour is common in many dynamic user interfaces: when we have to describe objects that enable other objects but must remain active.

5. Verification of User Interface Properties

The idea to use model checking techniques to reason about user interface properties was introduced by Paterno' in [P93]. The notation used is Action-based Temporal Logics, a branching-time temporal logic which allows designers to reason about the actions a system can perform. In [PM95a] there is an extended discussion about user interface properties that can be checked with this notation.

The LOTOS specification of the Interactive System software is mainly obtained by associating a LOTOS process with each interactor. Some control process may be added in order to control further the dynamic behaviour of the resulting specification. One of the main advantages of LOTOS specifications is the possibility to reason about their properties by applying automatic tools for model checking. This means that the LOTOS specification is automatically translated into a corresponding labelled transition system which represents the model against which user interfaces properties expressed in Action-based temporal logics are automatically verified.

The sort of properties that we deal with are:

- that a particular input will always result in a particular application input (reachability property);
- that all user actions have a corresponding interface appearance (visibility property);
- that a user action is reflected in an interface appearance immediately before any further user action is permitted (continuous feedback);
- that user actions are available to recover from an error (recoverability).

Its is clear that although these are general properties they can be tailored to the requirements of the particular system being specified.

6 Classification of Properties

As we said before, most of verification environments are based on the hypothesis that a system can be modelled as a Labelled Transition System (LTS). That is, they give no modality to deal with non finite-state LTS. In the Jack environment [BGL94] when a specification has to be verified, first a test to check whether the expression is finite is performed in order to check the possibility to build the model for the verification.

Recently, a new method[DFGI95] has been proposed that allow to check properties expressed in ACTL and on CCS terms. For some kind of properties, they give a

semidecisional procedure, which means that in some cases the verification can be performed in other cases this cannot occur.

Temporal properties are usually subdivided into subclasses: *liveness* properties (something good eventually happen) and *safety* properties (nothing bad can happen). Properties can be divided in two classes: *universal* (for all computations) and *existential* (for one computation).

6.1 Classification of properties expressed in ACTL

If we use the ACTL notation to express these properties we can identify some operators which are more suitable to indicate the related concepts. We say that is a *liveness* property if it is expressed in the following manner: AFf, EFf, AFAGf, EFAGf, AFEGf, EFEGf, AGAFf, EGAFf, AGEFf, EGEFf with *f* positive formula. This mainly means that the F (eventually) operator should be included in the property.
In the case of *safety properties*, they are expressed by using ACTl in the following way: AGf or EGf and f is a finite positive property. This means that the G operator (indicating all the states) should be part of them.

Note that these illustration of properties do not cover all liveness and safety properties that is possible to express using ACTL; infact, all properties containing the negation are not considered. The motivation is that the model checking environment which we will consider is able to give useful answer only with positive properties (properties which do not contain negations).

Finally we can introduce *finite property* if = Xf, and f= truel f or fl f and fl not f l E lA. This properties indicate aspects which should be verified in order to get to the next state (X).

6.2 Properties which can be verified over infinite states automaton

The approach presented in [DFGI95] provides a method to define the validity of a property on a non-finite system. This method is able to give a decisional procedure for some types of properties (finite properties), a semidecisional procedure for other properties (safety and liveness properties) and no answer for the remaining properties.
This is obtained by proving it on the elements of an approximation chain of finite labelled transition system which are bisimulation equivalent with the starting specification.

In this method the liveness properties are more easily preserved. The safety properties are more hard to be verified. In fact, this type of formula cannot be verified on all state of the

specification (as in this case they are infinite). However if the formula is true on an approximation of the system then it can be considered verified over the initial specification. Unfortunately, only a subclass of such properties are provable when finite approximation are considered. In fact, we can not prove universal safety properties.

7 A Discussion over user interface properties which can be verified

We want to analyze what user interface properties can be inserted in the liveness or safety category whose syntax has been described previously and what user interfaces properties can not be classified following that syntax.

7.1 User Interface Liveness Properties

We want to consider the HCI properties that we can insert in the liveness category formulae. The following four properties can be considered examples of liveness properties in the user interface field.

Reachability: this property allows us to verify that a user interaction can generate an effect on a specific part of the user interface. This means that, given a user action, we wonder whether it is possible to reach a given effect.

The next ACTL formula means that for all the possible futures (A operator) and for all the possible states (G operator), if *user actionx* is performed then there exists at least one temporal evolution until the Reachable_effect action has been performed ({Reachable_effect} true operator).

AG([user actionx]EF<Reachable_effect> true)

Possibility of performing a task at any state: this property states that starting from all the states there is one temporal evolution during which, at some time, the action associated with the task performance occurs.

AGEF<task_performance>true

Visibility: this property means that each user action is associated with a modification of the presentation of the user interface to give feedback on the user input. More precisely, in ACTL, we specify that whenever we have a user action we want to be sure that there is at least one temporal evolution where the event associated with the specific user interface modification indicating that the input has been received by the system, will occur. In terms of the abstract model this is expressed as:

AG([user actionx]EF <User interface appearancex>true)

Recoverable Error: an error can be defined as an action which is not needed to perform the current user task. Errors can be divided into minimal, recoverable and unrecoverable. If a minimal error occurs, immediately afterwards the user can perform one action which is useful for task performance. After a recoverable error, several actions are needed to return to the previous state. An error is unrecoverable if it does not allow the user to perform a given task in the current session.
We can use ACTL to define these three types of errors formally.
A recoverable errors can be defined:

AG([recoverable_error]EF<useful_task_action> true & EF<ask_performance >true)

These properties are liveness properties, so we can evaluate these building some approximations of the infinite automaton. If on a approximation of the LTS, the liveness properties are evaluated true, then we can affirm that the properties are true on the infinite automaton too; instead, if on all the approximations the properties are false, we can not affirm nothing on the falsity of these on the automaton; in fact this could simple say that we not yet find the approximation on which the properties are true.

7.2. Safety User Interface properties
Here we want to consider HCI properties that can be inserted in the class of safety formulae.
Here we can have a version of the visibility property which expresses the related concept in a stronger way: if we want that the feedback of a user action occurs immediately after it. This is expressed in ACTL:

AG([user actionx]EX {User interface appearancex}true)

Minimal error: immediately afterwards a erroneously action, the user can perform one action which is useful for the task performance; this is expressed by the ACTL formula:
AG([recoverable_error]EX{useful_task_actiontrue})

The truth of safety properties is more hard to prove than liveness properties on infinite automaton approximation with the method which we have considered.

7 .3. User Interface Properties which cannot be verified
Unfortunately, there are HCI properties on which approximations of the infinite automaton is not capable of give some answers; this is the case of the unrecoverable errors and of all formulae that contain negative formulae.

We indicate with unrecoverable errors the situation in which once the error has been performed, for all the possible temporal evolutions, it is no longer possible to perform the desired task, thus it is not true that once the unrecoverable error is performed then it exists one possible evolution where eventually the task is performed:

AG[unrecoverable_error]~(EFEX{ task_performance}true)

This formula contains a negation, so this is not evaluable, using the method considered. This problem affects the evaluation on some CARE properties too.

The CARE properties [CNSBMY95] have been recognised as a simple but useful framework to evaluate the usability of Multi-Modal Interactive Systems [CDFHP95]. We can express them in ACTL, assuming that m1 and m2 are two actions associated with the use of two different modalities (for example, voice input and graphical input).

Complementarity : the possibility to achieve a desired effect using two modalities in an independent order; it is expressed by:

AG[m1] EF<m2> EF<effect>true & AG[m1] ~(E[true{~m2} U {effect}true])
&
AG[m2] EF<m1> EF<effect>true & AG[m2] ~(E[true{~m1} U {effect}true])

which means that whenever m1 is used then eventually it is possible to use m2 and then to reach the desired effect, but it is not possible to reach it without performing m2 as well.

Assignment : the possibility to reach a desired effect using only one modality is expressed by:

AG [m1] EFEX{effect}true & ~(EF([m2] E[true{~m1} U {effect} true])

which means once m1 has been performed then it is possible to reach the desired effect and there is no possibility to perform m2 and not m1 to reach it.
We can rewrite it as:
For these properties, the approximations do not say anything because they contain formulae with negations.

Another thing that is still impossible to evaluate using approximations of an infinite state automaton is the falsity of a liveness or safety property. This is only possible on finite properties or some safety properties.

7. Conclusions

The importance to be able to reason about user interface properties is increasingly recognised. Model checking is an interesting approach to support this kind of evaluation. When industrial size cases are considered it becomes important to understand the limitations of the notations used in order to drive their application and to recognise when results can be obtained. We have discussed how to use new research approach for verification of infinite state specifications to user interfaces in order to give more precise indications about when model checking can be used in Interactive System design.

References

[AWM95] G.Abowd, H.Wang, A.Monk, "A formal technique for automated dialogue development", Proceedings Designing Interactive Systems'95, August'95, ACM Press.

[BACL95] P.Bumbulis, P.Alencar, D.Cowan, C.Lucena "Combining Formal Techniques and Prototyping in User Interface Construction and Verification", Proceedings of Design, Speciifcation, Verification of Interactive Systems'95, Springer Verlag, pp.17-192.

[BGL94] A.Bouali, S.Gnesi, S.Larosa, "The integration Project for the JACK Environment", Bulletin of the EACTS, N.54, October 1994, pp.207-223.

[BCMD92] J.Burch, E.Clarke, D.Long, K.McMillan, D.Dill, L.Wang, "Symbolic Model Checking: 10^{20} states and beyond", Information and Computation, 98(2), pp.142-170, June 1992.

[CNSBMY95] J.Coutaz, L.Nigay, D.Salber, A.Blandford, J.may, R.Young, "Four easy pieces for assessing the usability of multimodal interaction: the CARE properties," Proceedings INTERACT'95, Norway, June'95.

[DFGR93] DeNicola, R., Fantechi, A., Gnesi, S. and Ristori, G.(1993) An Action Based Framework for Verifying Logical and Behavioural Properties of Concurrent Systems. Computer Networks & ISDN Systems, **25**, 761-778.

[DFGI95] N.De Francesco, A.Fantechi, S.Gnesi, P.Inverardi, "Model Checking of non-finite state processes by Finite Approximations"

[FG92] A.Fantechi, S.Gnesi, "The Expressive Power of LOTOS Behaviour Expressions", IEI Internal Report, B4-43, October'92

[HD96] M.Harrison, D.Duke, "The Specification of User Requirements in Interactive Systems", to appear in Handbook of Human-Computer Interaction

[ISO88] 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.

[JH92] C.Johnson, M.Harrison, "Using Temporal Logic to Support the Specification and Prototyping of Interactive Control Systems". International Journal of Man-Machine Studies, 37, pp.357-385, 1992.

[M89] R. Milner. Communication and Concurrency. Prentice Hall 1989.

[P93] F.Paterno', "Definition of Properties of User Interfaces", Proceedings of SEKE '93 Conference, June'93, S.Francisco, pp.314-318.

[PB95] P.Palanque, R.Bastide, Verification of an Interactive Software by Analysis of its Formal Specification, Proceedings INTERACT'95, Lillehammer, June'95.

[PM95a] F.Paterno', M.Mezzanotte, "A systematic approach to transform task specifications into architectural specifications to verify user interface properties", CNUCE Internal Report, September 95

[PM95b] F.Paterno', M.Mezzanotte, "Formal Analysis of User and System Interactions in the CERD Case Study", Proceedings EHCI'95 Ifip Conference, Chapman&Hall Publisher, Wyoming, August 1995.