Reconciling Safety and Usability Concerns through Formal Specification-based Development Process

David Navarre, Philippe Palanque, & Rémi Bastide

Université Paul Sabatier & Université Toulouse 1, France
{palanque, navarre, bastide}@irit.fr
http://lihs.univ-tlse1.fr/{palanque, bastide, navarre}

Abstract

The design of safety critical systems calls for advanced software engineering models, methods and tools in order to meet the safety requirements that will avoid putting human life at stake. When the safety critical system encompasses a substantial interactive component, the same level of confidence is required towards the human-computer interface. Conventional empirical or semi-formal techniques, although very fruitful, do not provide sufficient insight on the reliability of the human-system cooperation, and offer no easy way to, for example, quantitatively compare two design options. The aim of this paper is to present a method with supporting tools and techniques for engineering the design and development of usable user interfaces for safety-critical applications. The specific application area that we consider is air traffic management but most of the results will be valid for any application areas with similar requirements.

Introduction

Usability and safety requirements in the design and specification of Interactive Systems are usually investigated in separate and limited ways. There is a lack of structured methods or tools that can drive the work of designers and developers, especially in safety critical application domains that require sophisticated interaction techniques such as Air Traffic Management.

Air Traffic Management (ATM) is an important application area, in which many problems are still to be solved. For example a number of air traffic control incidents routinely occur because of the undesired effects of interactions between human actors, or because of the lack of efficiency of current systems, which ends up by wasting time both for air traffic controllers and for pilots and above all endangering passengers. The main reason for this seems to be the misunderstandings resulting from the fact that interaction between controllers and pilots is done exclusively through voice communications. Airlines have estimated that an improvement in air-traffic control could lead to savings up to 20% of fuel costs and of course to reduce significantly delays both at departure and landing phases.
This class of application represents a challenge for people involved in specification, design and development of user interfaces.

We believe that the use of formal description techniques for both task and system can be part of a valid answer to these problems provided this activity is tool-supported for edition, simulation and verification of the models. Another part of the answer is to deal with the contradictions inherent to current practice in the fields of software engineering and human-computer interaction (HCI). Indeed, software engineering is reliability and functionality-oriented promoting global and structured development of software systems (Hall, 96) through, for instance, formal methods and models while HCI is usability oriented promoting user/usage-centered iterative development through mock-ups and prototypes.

This paper addresses the problem of specification and verification of all the components of a safety critical interactive application and presents how formal methods can be applied on a full-scale case study. Special attention is paid to the development process of the interactive application and to tool support for formal development and prototyping.

The first section of the paper presents a development process dedicated to the design of safety critical interactive applications using formal description techniques. Section 3 is devoted to the formal specification techniques used in this development process namely Interactive Cooperative Objects (ICO). A case study in introduced in section 4, while section 5 presents PetShop environment dedicated to the iterative development of interactive software.

Development Process of Interactive and Safety Critical Software Systems

The development process of software systems has been the focus of a significant amount of work in the field of software engineering. Early models (including waterfall and V models (Mc Dermid & Ripkin 1984)) focus mainly on the identification and the clear separation of the various phases of the development of software systems. However the way

they represent the process (i.e. in a linear and mainly one-way structure) is very limited and not able to deal with prototyping issues. The spiral development model was introduced by Boehm in 1988 (see Figure 1) to deal explicitly with prototyping and iteration. Prototyping is a key issue in the development of interactive systems and thus this model has been widely adopted.

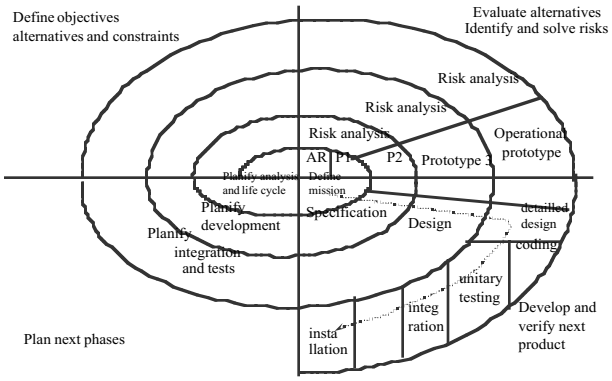

Figure 1. The Spiral model (from (Boehm, 88))

However, this model does not encompass the various models that have to be built during the development of an interactive system. For instance, it does not refer to task models that are now recognized as critical for the design of usable interactive systems. Besides, usability evaluation is not explicitly represented in the model thus leaving user involvement unsupported and at the discretion of software engineers. Research has been conducted in this field and the star model (Hix & Hartson, 92) explicitly introduces explicitly task analysis and usability evaluation as phases of the development process. However, most of the phases must generally be conducted manually i.e. without tool support for both representing and analyzing the models built during those phases. This is not critical when dealing with "classical" (i.e. non safety critical) interactive systems, but each manual operation may be source of a fatal error when safety critical systems are concerned.

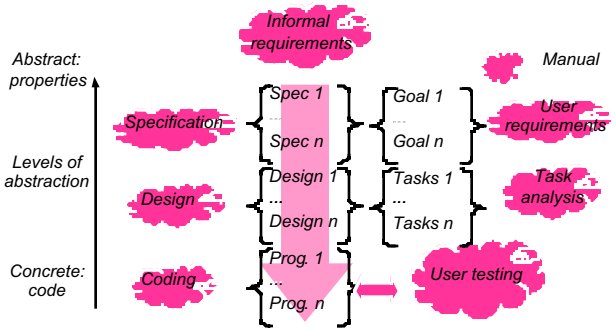

Figure 2. Various phases of the design process of interactive systems (left), user related parts (right)

Figure 2 presents the various phases of the development process of a software system with an emphasis on the output of those phases. The phases that require human creativity and intervention are represented as clouds on the figure. An important aspect is that each phase produces several models corresponding to the top-down process of taking into account information in a stepwise refinement manner. Even though the process is highly iterative (as for the spiral model) we have decided to represent here only the flow of information between the various phases.

When dealing with interactive systems it is now widely agreed upon that user information (that leads to usability) has to be taken into account and that task analysis and task modeling may support this activity. Right hand side of Figure 2 shows that user goals have to be analyzed and that their level of abstraction is the same as the one of specification phase, while task analysis level corresponds to design phase.

The advantage of using formal notations to support the design of the models is the potential for mathematical verification they provide. A formal model (whether it describes a task or a system) may be automatically checked for several behavioral properties such as deadlock freedom or termination or other more user related properties such as honesty or predictability (Gram & Cockton 96). If, at a given stage, the model fails to comply with such properties, it has to be reworked and corrected until the problem is solved. This process is illustrated on the left hand side of Figure 3.

If formal methods are used during the design process, the coding phase can be at least partly automated for instance by means of code generation. This automation of the coding phase can also be done by the interpretation at run-time of the models built in the earlier phases (this is close to the model-based approaches to the design of user interfaces (Wilson et al., 93), (Szekely et al., 93) and (Paterno 1999). We have previously investigated the pros and cons of these two approaches in (Bastide & Palanque, 96).
It is important to notice that such a process may, at the very best, ensure the system under design is "defect free". This process alone can by no means ensure that the system will be usable at all, and much less that it will be "user friendly". In order to cope with all the issues a d thus deal both with safety and usability requirements that are crucial for safety critical interactive applications, we propose an iterative development process based on formal notations.

The solution we propose to this end is twofold:

- We propose a development process supporting the use of formal notations and iterative user-centered prototyping. This process (see Figure 3) is based both on prototyping and formal description technique in order to deal with reliability and usability concerns. This process is tool-supported as it is shown on the Air Traffic Management case study in next sections.
- We propose a formal notation for the modeling of interactive systems. This brings the advantages of formal approaches, the most important of which are conciseness, consistency and lack of ambiguity. This also makes task models amenable to mathematical verification i.e. the possibility for proving properties over system models. As this paper deals in priority with the

development process the formal notation proposed (the ICO formalism) is only superficially introduced in the next section.

The left-hand side of Figure 3 presents the iterative prototyping phase. This is what we call the super-high fidelity prototyping development phase. Indeed, this is not a low-fidelity prototyping activity as the prototype produced represents the exact look ad feel of the final system. It is more than a high-fidelity prototype as the system is not built using a "classical" Rapid Application Development (RAD) environment (such as MicroSoft Visual Basic for instance). Indeed, the system is built using the ICO formalism that is analyzed (through formal analysis) and executed. These prototyping activities allow for taking into account user's constraints and needs through a stepwise refinement process and ensuring a certain quality of the system through the formal verification of the specifications.
It is important to notice that the building of the system is

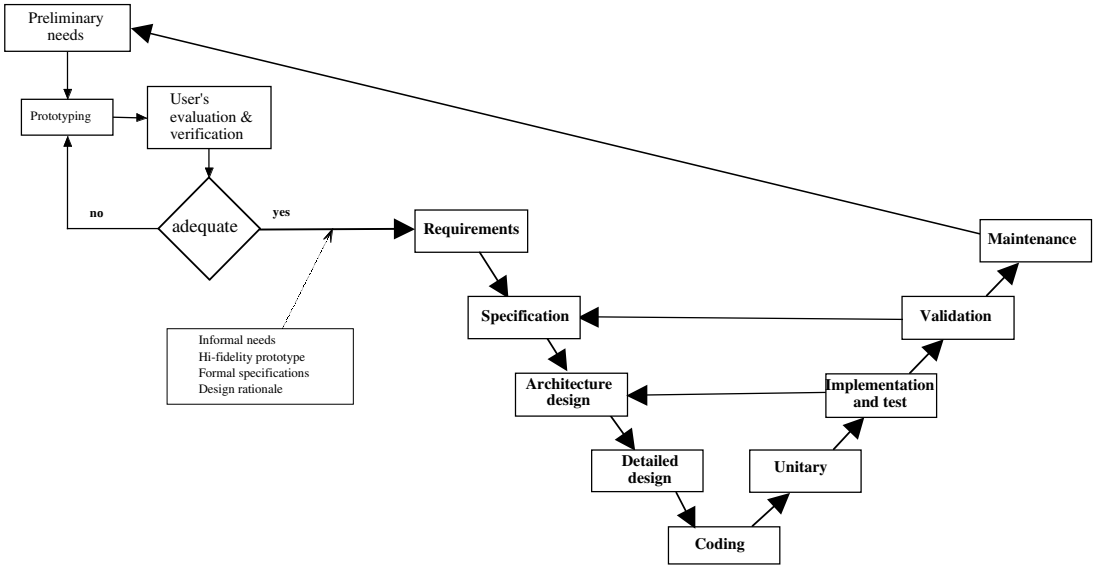

Figure 3. Integrating formal prototyping with software development

not finished at the end of this prototyping phase. Indeed, development of safety critical applications require a more structured and global development process as the one promoted by the waterfall model. The right-hand side of Figure 3 shows the basics of the waterfall development process. This Figure represents also how the prototyping iterative process and the waterfall one are related. The approach we promote provides several valuable inputs for this "classical" development process:

- a set of validated requirements elicited and tested by the users during the prototyping phase and thus reducing time spent in the requirement phase;
- a set of formal specification of the interactive part of the application. These specifications will be used as inputs in the specification phase and will thus contribute to reduce development time;
- a complete user interface (both its presentation and its behavior) that will have to be re-implemented in the development phases. Indeed, the user interface produced by the execution (within PetShop) of the ICO specification cannot be used for the final system as PetShop is oriented towards interpretation and thus cannot reach the level of performance required for safety critical applications.

The ICO Formal Description Technique

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems (Bastide & Palanque 1999). It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets (Bastide & Palanque 1990) to describe their dynamic or behavioral aspects.
ICOs are dedicated to the modeling and the implementation of event-driven interfaces, using several communicating objects to model the system. ICO are used to provide a formal description of the dynamic behavior of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user).In the ICO formalism, an object is an entity featuring four components: a high-level Petri net (describing the behavior of the object), a presentation part, and two functions (the activation function and the rendering func-

tion) that make the link between the cooperative object and the presentation part.

Presentation part: the Presentation of an object states its external appearance. This Presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user _ system interaction) and/or a way to display information from this interactive system (system _ user interaction).

Activation function: the user _ system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the ICO's user services. The relation between user services and widgets is fully stated by the activation function that associates to each couple (widget, user action) the user service to be triggered.

Rendering function: the system _ user interaction (outputs) aims at presenting to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented (Navarre et al. 2000). The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

An Excerpt of a Case Study

This paper is based on a real-size case study on Air Traffic Management (ATM).

The airspace is divided in sectors, each of them being controlled by two air traffic controllers managing different tasks and working in a cooperative way[1]. The air traffic controller uses a workstation for handling the traffic over a given sector, and communicating with the pilots of the planes currently flying through the sector.

The information about the planes is displayed on a radar screen. This radar screen is slightly more complex than classical ones, as several information about the plane are displayed aside each plane icon (several past positions, aircraft identification number, speed vector, …). The controller has a kind of memory jogger organized in a set of paper ribbons called strips. Each strip is initially emitted by a computerized system that fills up the information corresponding to the initial flight plan of the aircraft. When controllers ask pilots to modify flight parameters of the flight (such as speed, heading, ...) they write down the information on the strip. VHF radio equipment for communicating with pilots is managed by a phone. It is important to notice that there is no way to target a specific plane for communication, since all aircrafts use the same frequency.

An alternative system called Data-Link is currently under study in several countries. This system addresses the bandwidth problem by providing an additional communication channel between pilots and controllers. This system implies a modification of both controller's workstation and pilot's board. In this paper we mainly focus on the air traffic control workstation. This is reasonable as these two parts of the same system are currently dealt with in a relatively independent manner by different companies. Besides the prototype we have been working on encompasses a challenging user interface part featuring both graphical presentation and direct manipulation.

One of the challenges of building a system encompassing a graphical user interface for this kind of safety critical systems is that the level of reliability of the air traffic control applications as to be guaranteed at the same level as before. This level is very high as, for instance, in France no air plane accident has ever been attributed to an error made by air traffic management. An example of such data-link application is presented in Figure 4 showing a snapshot of Druides system. Only part of the application is shown here[2].

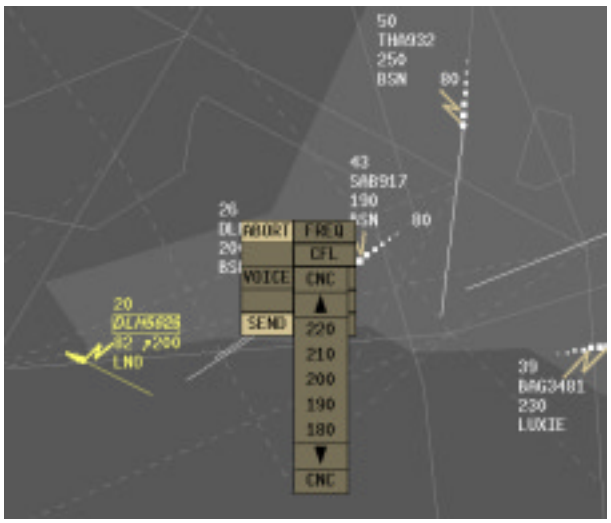

Figure 4. Part of the user interface of the Druides: an ATM prototype featuring Data Link communication

The menu (currently opened on Figure 4) is used by the controller for building data-link commands (called clearances). This menu is a pop-up menu that appears when the user clicks on the label of an aircraft. The left-hand side of the menu offers three commands, SEND and ABORT (for sending or canceling the current data-link clearance) and VOICE for asking the pilot to call the controller using the VHF communication channel. On the right-hand side of the menu the controller can select one out of five commands:

- FREQ (for asking the pilot to switch form one radio frequency to another one),
- CFL (for changing the Cleared Flight Level of the aircraft),

[1] For sake of simplicity we will consider in the remainder of this paper that all the work is done by a single controller.

[2] The other part of the application is dedicated to the user interface for handling strips in an electronic way (called electronic stripping). This is not taken into account here as it is a separated issue with respect to data-link communication facilities.

- SPEED (for changing speed),
- HEAD (for changing the heading of the plane)
- BEACON (for changing the route of the plane i.e. the next beacon it has to fly over).

On Figure 4 the controller has already selected the second command (CFL). In response the system has opened a pull down menu offering the most suitable possible values for changing the CFL. By pressing CNC (cancel) this pop-up menu is closed, by selecting a value, the parameter is set (the data-link command in built), and by clicking on the arrows, the values presented are scrolled.
The formal description of this case study is not the purpose of the paper and thus will not be presented here. The complete description of the case study is available on MEFISTO web site http://giove.cnuce.cnr.it/mefisto.html.

Petshop Support for Prototyping

In this section we present the PetShop environment and the design process it supports.
At run time, the user can both look at the specification and the actual application. They are in two different windows overlapping in Figure 5. The window PlaneManager (on the right hand side) corresponds to the execution of the window with the high-level Petri net (on the left hand side).

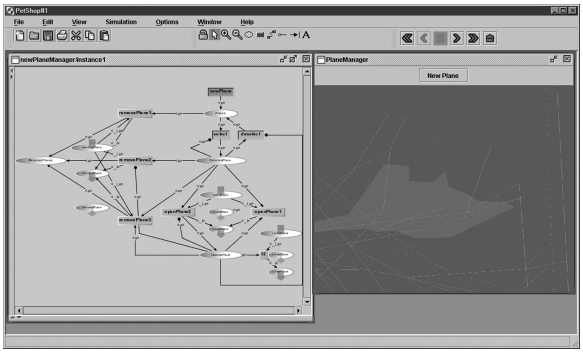

Figure 5: The execution of ICO specification in PetShop

Each time the user acts on the PlaneManager the event is passed onto the ICO interpreter. If the corresponding transition is enabled then the interpreter fires it, performs its action (if any), changes the marking of the input and output places and perform the rendering associated (if any). Within PetShop, prototyping from specification is performed in an interactive way. Anytime during the design process it is possible to introduce modifications either in order to make the specification more precise or to change it. The advantage of model-based prototyping is that it allows designers to immediately evaluate the impact of a modification and thus to make communication between users and designers easier.
Figure 6 presents the overall architecture of PetShop. The upper part of the Figure presents activities and models at design time while the lower part deals with runtime. In fact those two activities are deeply intertwined and the main difference is related to the user. At design time the designer edit the ICO specifications and can immediately execute it to see whether or not it behaves as expected. When the specification is "good enough" it can be presented to the user for validation/modification. At that time, the designer can directly amend the specifications according to user's comment and test immediately the modification. This environment supports in a synergistic way usability (through iterative prototyping and user testing) and reliability (through formal description techniques and verification).

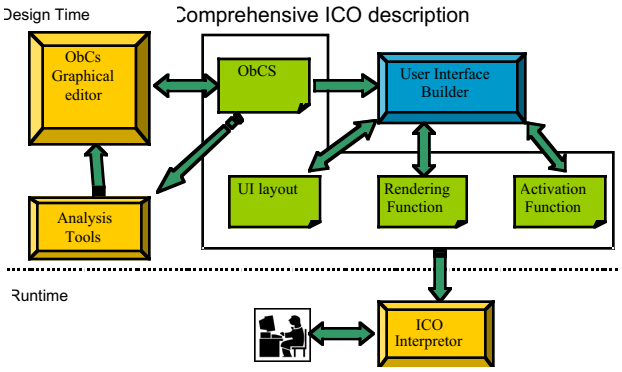

Figure 6: Architecture of the PetShop Environment

Related Work

Since the early work from D. Parnas (Parnas, 69), a lot of work has been devoted to the use and the development of formal methods for interactive systems. However, no formal approach has been fully successful yet, and some even wonder whether a unique formalism will ever permit a full description of an interactive system (Palanque et al., 96), leading to a variety of partial approaches. An interactive system can be considered according to its intrinsic nature or to its design process, these approaches are partial in the sense they only address:

- part of the design process,
- some of the components of an interactive system.
- For instance, some focus on the early stages of the design process such as requirements elicitation and analysis or early specification (McCarthy et al. 95), (Johnson & Jones, 97), (Johnson, 96).

Going back to the components of interactive systems as defined in the Arch architectural model (Bass et al., 91) a significant amount of work has been devoted specifically to the Dialogue component. This kind of work does not deal with Logical Interaction and Physical interaction components, as they make the reasonable assumption that these components have been taken care of by some error-free industry developer. In this category we find all the work done on the formal design of WIMP interfaces relying on the set of predefined interaction objects (for instance (Beck et al., 95).

Another trend in the field of formal methods for interactive systems is the investigation and the elicitation of generic properties (Dix, 91). This kind of work is very important as it characterizes interactive systems and then allows for verification of these properties over models. In the case of the framework we present here, this issue corresponds to the generic requirements for interactive systems as stated in section 5.1.
Lastly, significant work in this area has been devoted to defining interactive systems by means of elementary components called interactors. Two main models have been particularly studied York interactors (Duke & Harrison, 93) and CNUCE interactors (Paternò & Faconti, 92), and a comparison of this work can be found in (Duke et al., 94).
Formal notations featuring graphical representation have been extensively used for the design of various parts of interactive systems. The dialogue part of classical interactive systems have been modeled using various existing formalisms such as state diagrams as in (Parnas, 69) (Jacob, 86), data flow diagrams as in (Schreiber, 95), statecharts as in (Carr, 94), Petri nets as in (Beck et al., 95), Petri nets with objects as in (Bastide & Palanque, 90). Some authors have enhanced exiting graphical notations in order to cope more efficiently with specific aspects of interactive systems as in (Schlungbaum & Elwert, 96) or in (Torres et al., 96).

Conclusion

Prototyping is now recognized as a cornerstone of the successful construction of interactive systems as it allows making users at the centre of the development process. However, prototyping tends to produce low quality software as no specification or global design is undertaken. We have shown in this paper how formal specification techniques can contribute to the development process of interactive systems through prototyping activities.
While the ICO formal specification technique has reached a maturity level allowing coping with real size dynamic interactive applications, the Petshop environment is still under development. A real size application has been completely specified in the field of the European project MEFISTO (http://giove.cnuce.cnr.it/mefisto.html). More information about PetShop can be found on PetShop's web site: http://lis.univ-tlse1.fr/petshop/
However, the work done on this Air Traffic Control application has also shown the amount of work that is still required before the environment can be used by other people than the ones that took part in its development.
In order to make it attractive to developers we are integrating additional features such as:

- tool-supported verification of properties,
- analysis of conformance with other representations such as tasks models
- performance analysis in order to support design decisions and informal users' validation.

Another stream of research we are investigating is the generation of test cases from the formal specification in order to help developers checking whether an implementation is conformant with respect to the specification. This will allow development teams to take the specifications, still use their favorite programming environment and later check whether their implementation is conformant with it.

Acknowledgments

The work has been partly funded by the Esprit project MEFISTO n° 24963.

References

Bass, L., R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard, and M. R. Szezur. "The Arch Model: Seeheim Revisited." User Interface Developpers' Workshop, Apr. 26. Version 1.0, 1991.

Bastide, Rémi, and Philippe Palanque. "Petri Net Objects for the Design, Validation and Prototyping of User-Driver, Interfaces." 3rd IFIP Conference on Human-Computer Interaction, Interact'90, Cambridge, UK, Aug. 1990, 625-31. North-Holland, 1990.

Bastide, Rémi, and Philippe. Palanque. "Implementation Techniques for Petri Net Based Specifications of Human Computer Dialogues." in CADUI'96, 285-302. Presses Universitaires de Namur, 1996.

Bastide, Rémi, and Philippe Palanque. "A Visual and Formal Glue Between Application and Interaction." Journal of Visual Language and Computing 10, no. 3, 1999.

Beck, A., C. Janssen, A. Weisbecker, and J. Ziegler. "Integrating Object-Oriented Analysis and Graphical User Interface Design." editors Joëlle Coutaz, and Richard Taylor. Lecture Notes in Computer Science (1995).

Boehm, B. W. "A Spiral Model of Software Developement and Enhancement." IEEE Computer 21, no. 5, 61-72, 1988.

Dix, Alan. Formal Methods for Interactive System Academic Press 1991.

Duke, David J., Giorgio Faconti, Michael Harrison, and Fabio Paternò. "Unifying Views of Interactors." AVI'94.1994.

Duke, David J., and Michael Harrison. "Abstract Interaction Objects." EUROGRAPHICS'93, 25-36.1993.

Gram, Christian, and Gilbert Cockton, editors. Design Principles for Interactive Software Chapman & Hall 1996.

Hall, Antony. "Using Formal Methods to Develop an ATC Information System." IEEE Software , no. 3, 66-76, 1996.

Hix, Deborah, and Rex Hartson. Developping User Interfaces Wiley 1992.

Jacob, R J K. "A Specificationlanguage for Direct Manipulation User Interfaces." ACM Transactions on Graphics 5, no. 4, 283-317, 1986.

Johnson, Chris W. "Literate Specifications." Software Engineering Journal , 225-37, 1996.

Johnson, Chris W., and S. Jones. "Human-Computer Interaction and Requirements Engineering." SIGCHI Bulletin, Editorial for Special Issue on HCI and Requirements 29, no. 1, 31-32, 1997.

McCarthy, J, Peter Wright, and Michael Harrison. "A Requirements Space for Group-Work Systems." in Human-Computer Interaction, Interact'95, LilleHammer, Norway, June 1995, 283-88. Knut Nordby, Per H. Helmersen, David J. Gilmore, and Svein A. Arnesen, editors. Chapman & Hall, 1995.

Mc Dermid, John et Ripkin K., Life cycle support in the ADA environment. Cambridge University Press; 1984.

Palanque, Philippe, Fabio Paternò, Rémi Bastide, and Menica Mezzanotte. "Towards an Integrated Proposal for Interactive Systems, Based on LOTOS and Object Petri Nets." Design, Specification and Verification of Interactive Systems'96, 162-87. Springer-Verlag, 1996.

Parnas, D. L. "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System." 24th ACM Conference, 379-85.1969.

Paternò, Fabio, and Giorgio Faconti. "On the LOTOS Use to Describe Graphical Interaction." in BCS HCI'92 conference, Cambridge University Press, pp. 155-74.

Andrew Monk, Dan Diaper, and Michael Harrison, Editors. BCS Conference, 1992.

Paternò, Fabio, Model Based Design and Evaluation of Interactive Application. Springer Verlag; 1999.

Schlungbaum, Egbert, and Thomas Elwert. "Dialogue Graphs: a Formal and Visual Specification Technique for Dialogue Modelling." in BCS-FACS Workshop on Formal Aspects of the Human Computer Interface. Chris Roast, and Jawed Siddiqi, Editors.Sheffield Hallam University, Springer-Verlag, 1996.

Schreiber, Siegfried. "The BOSS System: Coupling Visual Programming With Model Based Interface Design." Interactive Systems: Design, Specification and Verification. Fabio Paternò, Editor, 161-78. Springer verlag, 1995.

Szekely, P., O. Luo, and R. Netches. "Beyond Interface Builders: Model-Based Interface Tools." INTERCHI'93, Amsterdam, The Netherlands, 383-90. ACM Press, 1993.

Torres, J. C., M. Gea, F. L. Gutierrez, M. Cabrera, and M. Rodriguez. "GRALPLA: An Algebraic Specification Language for Interactive Graphics Systems." Design, Specification and Verification of Interactive Systems, 1996, 272-91.eds. F. Bodart, and J. Vanderdonckt, Springer Verlag, 1996.

Wilson, S., P. Johnson, C. Kelly, J. Cunningham, and P. Markopoulos. "Beyond Hacking: a Model Based Approach to User Interface Design. " HCI'93, Loughborough, U.K., 217-31. Cambridge University Press, 1993.