Formal validation of HCI user tasks

Yamine Ait-Ameur, Mickael Baron, Patrick Girard

Formal validation of HCI user tasks

Yamine AIT AMEUR, Mickael BARON and Patrick GIRARD
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA-University of Poitiers
BP 40109, 86961 Futuroscope Cedex, France
Email: {yamine, baron, girard}@ensma.fr

***Abstract*— Our work focuses on the use of formal techniques in order to increase the quality of HCI software and of all the processes resulting from the development, verification, design and validation activities. This paper shows how the B formal technique can be used for user tasks modelling and validation. A trace based semantics is used to describe either the HCI or the user tasks. Each task is modelled by a sequence of fired events. Each event is defined in the abstract specification and design of the HCI system.**

***Index Terms*— Human Computer Interaction, formal methods, B method, proof by refinement, interaction properties verification and validation, task modelling and validation, software architecture.**

I. INTRODUCTION

This paper starts from the two following observations:

- the use of formal methods for software development, verification, validation and maintenance is being widely accepted in the software engineering area and particularly in human computer interaction (HCI) software,
- the development of the HCI part of a given software has considerably increased in the last twenty years. In several cases, this part can reach fifty per cent of a software development.

Our work focuses on the use of formal techniques in order to increase the quality of HCI software and of all the processes resulting from the development, verification, design and validation activities. In past workshops and conferences, we presented our approach through papers dealing with formal specifications of HCI software [1] formal verification of HCI software [2], test based validation of existing applications [3] and refinement from formal specifications to ADA code [4]. This paper addresses another topic not tackled yet by our approach: design and formal validation of formal specifications with respect to informal requirements. This work completes the whole development process of a HCI software [5][2][3] [4]. Indeed, our approach uses the B formal technique for representing, verifying and refining specifications (see figure 1).

This paper adresses the problem of validation for HCI developments. In general, the development of HCI is concerned by two important interleaving phases.

1) A *design phase* which allows to produce the code implementing the suited HCI and its link with the functional core (heart of the application). In this phase, architectural notations, verification, validation, specification, refinement and programming techniques are used by HCI developers.
2) *A task validation phase* which consists in validating user needs. This phase is not well mastered by HCI designers since most of these validations are issued from non computer scientists like psychologists and ergonomists. A set of tasks is described at the requirement analysis phase and shall be supported by the final HCI product.

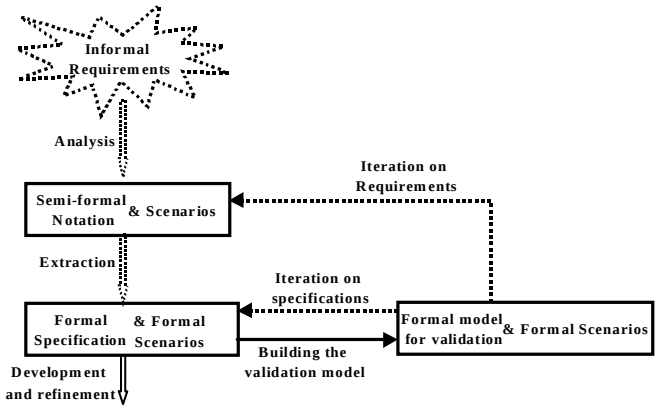

Fig. 1. A generic approach for requirement engineering.

Our concern is related to the second point. We claim that it is possible to validate user tasks at the specification and design phase. Moreover, we claim that formal specification and verification techniques allow to perform user tasks validations.

This paper is structured as follows. Next section presents a brief overview of the semi-formal notations and of the formal techniques used in HCI engineering. Section 3 outlines our approach for formal HCI design and section 4 presents our approach for user tasks validation. Then, an example showing how this approach works is detailed in section 5. Finally we conclude and give an overview of our future work.

Notice that this paper is shortened in order to be contained in 7 pages. More information can be obtained from the authors and/or in its long version.

II. Development of HCI software

Like for other areas of software engineering, HCI area has its own notations, techniques and methods. This section, outlines briefly these elements and fixes the context of our work.

A. Notation for design and validation of HCI.

Several semi-formal notations and models have been defined in the HCI area. These models are issued from several different communities already mentioned like ergonomy and psychology. These notations have followed the same evolutions as for software engineering. They are divided into two main categories: design notations and task description notations.

1) Design and architectural notations: They allow to express the static structure of the software implementing the system. All the architecture and design models in HCI separate the design of the HCI from the design of the functional core of the application for whom the interface is defined. The Seeheim model [6] is an architecture model with three modules: presentation, dialog control and the interface (usually an API) with the application modules. The PAC [7] model (Presentation, Abstraction and Control) is a decomposition of the Seeheim model since it contains several small Seeheim models that interact with each other. The Arch [8] model integrates the whole application from the toolkit to the functional core of the application. It is the most complete model.

Other design models have been suggested, we do not review them in this paper. Finally, we have used the ARCH model for our developments.

2) User tasks notations: They allow to describe the user needs in terms of usability of an HCI. They are usually issued from non computer scientists (ergonomy and psychology for example). The majority of these notations are user centered. They support the definition of user task by a sequence of actions to be performed by the user. Approaches like MAD [9], HTA [10], UAN [11] support notations which decompose a task in a tree where the basic actions are defined in the leaves of the obtained tree. Other notations like CTT [12] define a formal language allowing to describe a set of tasks. However, these notations do not have formal semantics and therefore they cannot be checked on the current HCI design.

3) Properties in HCI: Properties of HCI are those commonly described for interactive systems in general like safety and fairness more those properties related to usability of the HCI like reachability and insistance. In the HCI development area, two categories of properties have been distinguished by several authors [13] [14].

- *Validation properties* characterize the behavior of the HCI suited by the user (completeness, flexibility, task achievement, and so on).
- *Robustness properties* are related to the reliability of the HCI in particular and of the system in general (observability, insistance and honesty).

B. Need of formalisation

The different models and notations we outlined above do not give an enough formal representation to allow the validation and verification of the HCI design with respect to given properties and user tasks. Verification and validation are reported to the testing phase when all the software development is completed. Representing both design and tasks will permit such verification and validation.

III. Formal development of HCI

Several formal techniques have been applied in the development of HCI software. Among the several used formal description techniques in software engineering, model oriented approaches play a major role in the HCI area. These methods are based on the description of the model by a set of variables. These variables are modified by the operations and events of the model. Generally, these techniques are divided into two categories: automatic proving and proof systems. Both of these two techniques have been applied to interactive systems [15].

The first category allows automatic proofs (model checking). It is based on the evaluation of logical properties on the state transition system which is obtained from the evolving variables. Among these techniques, we can find temporal logics, Petri nets and so on. In the area of interactive systems, these techniques are assumed to have been used first in formal verification of interactive systems [15]. For example, [1] verifies user interfaces with SMV (Symbolic Model Verfier)

using CTL (Computational Tree Logic), while [16] uses LOTOS to write interactors specifications, and analyses translated finite state machines using ACTL (Action-based Temporal Logic). [17] develops a new temporal logic based formalism, named XTL (eXtended Temporal Logic), to address singularities in interactive specification. Model checking is also used by [18] who model user and system by the way of object-oriented Petri nets ICO [18]. More recently, [19] used the Lustre language for the automatic validation of user interface systems.

The second category is based on proof systems where the model is described by variables, operations, events, temporal properties and invariants. The operations must preserve these invariants and a set of other properties (preconditions and/or post conditions). To ensure the correctness of these specifications a set of proof obligations are generated and shall be proved. According to its implementation, the proof system can achieve some of the proofs automatically. Among these techniques, we can find Z, based on set theory [20], VDM [21], based on preconditions and postconditions calculus [21]; [22] [23], and B, based on the weakest precondition calculus [24] [25]. In the HCI field VDM and Z have been used for defining atomic structures like interactors [26] [27], and Z and Object-Z are now used more extensively [28], HOL (a Higher Order Logic Theorem Prover) has been used in the verification of User Interface specifications [29][30].

Despite this large use of methods, the usability of formal methods in HCI is always under questions. Ben Shneiderman [31] ensures that currently, formal models ”are beneficial for only small components”, and identifies ”scalable formal methods and automatic checking of user-interface features” as a major point for researcher's agenda. Topics such as adequate tool support, cost/benefits ratio, and scalability for formal methods in interactive design are particularly relevant.

IV. Formal design of HCI using the B method

Our approach contributes towards the use of formal techniques for describing, designing and proving interactive systems. We showed that formal methods are scalable to complex HCI applications.

We use the B formal method to show that it is possible to handle a complete interactive software formal development. This approach allows specifying, verifying and refining the formal B specifications and design. Particularly, programs written in Ada and Tcl-Tk have been generated within this technique, demonstrating that our approach may be used in real size systems [5].

A. The B approach

We use the B method [24] and its event based definition [32]. This method allows to describe various kinds of systems like distributed, parallel, multi-modal, reactive, interactive systems and so on.

A B model is composed of a set of ”atomic” events which are described by particular generalized substitutions (assignment, ANY, BEGIN and SELECT). Events modify the state of the specified system. This state is defined by a set of typed variables which may evolve when events are fired.

For the purpose of this paper, we will only use the $SELECT$ substitution $Evt = SELECT\ P\ THEN\ S\ END$; The event Evt is fired and S is executed when P is true. Each event Evt is fired if and only the guard P associated to this event is true.

Moreover, a set of properties like invariants, liveness, safety and reachability properties may be associated to each B model. These properties allow to formally ensure the correctness and the validation of the behavior of the described system. They are proved during the development thanks to the embedded proof system associated to B and to the $Atelier\ B$ tool [33].

Finally, B models can be refined into other B models which can be enriched by new events and new properties. The refinement process leads a developer to the final HCI design after finite refinement steps.

B. Design and architecture with B

In our case, we use the set of events to define a transition system that allows to represent the dialog controller of the HCI we want to specify. The events of the dialog controller define an automaton which describes the interactive system. Each event is guarded by a strong predicate defining a guard which shall be true in order to fire the corresponding event. Moreover, it ensures that the associated operation (in the $THEN$ part) can be executed. The underlying semantics is a trace based semantics where the events are interleaved. Thus, there is no parallel firing of events (asynchronous modelling), and when the ”time is stretched” enough, the events can be considered to be fired in a sequential way.

In the case, of an interactive application described by several systems, our approach uses the refinement technique to introduce the events of the composed automata. Each system is then described progressively by refinements in an incremental way.

The different events defined during the specification and the design phases are gathered into different B models. These models obey themselves to architectural notations issued from the HCI engineering practice. In our case, as applied below, the ARCH architectural model will be used to describe the architecture of our case study.

C. Task validation with B

A Task is described by an initial state and a final state. It is decomposed in a *sequence* of several subtasks which are themselves decomposed in other *sequences* of subtasks and so on (see figure 2). This process is applied until the tasks cannot be decomposed i.e. the atomic events belonging to the dialog controller are reached in the leaves of the decomposition tree. From a specification point of view, this decomposition is represented by a set of refinement steps. A task is refined into a sequence of basic events which lead from the initial state to the final state. The only allowed control operator is the *sequence* *";"*.

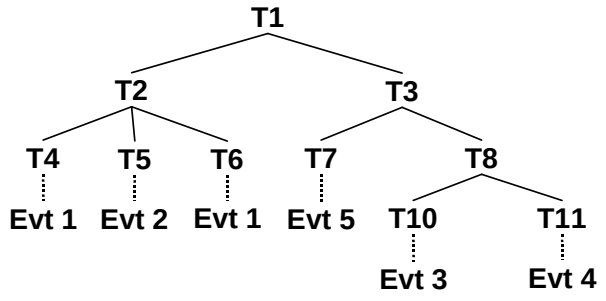

Fig. 2. Task decomposition into subtasks.

The higher level task is represented in B through a B model or an abstract machine. We use B to perform the decomposition of a given task into a set of ordered (by sequence) subtasks. The refinement technique consists in introducing the *sequence* and the subtasks. The proof obligations related to refinement, embedded in the B technique, ensure that this decomposition is correct. Thus our claim is that task validation may be performed by refinement.

On figure 2, a decomposition of task T_1 into an ordered set of subtasks T_2 and T_3 is shown. Moreover, the whole decomposition is given by:

$$
\begin{array}{l}
T_1 = T_2; T_3 \\
T_2 = T_4; T_5; T_6 \\
T_3 = T_7; T_8 \\
T_8 = T_{10}; T_{11} \\
T_1 = T_4; T_5; T_6; T_7; T_8 \\
T_1 = Evt_1; Evt_2; Evt_1; Evt_5; Evt_3; Evt_4
\end{array}
$$

Evt_i are "atomic" events from the dialog controller and sequence is the only allowed task composition operator.

The refinement preserves all the properties of the initial task T_1. When the basic events are reached by the refinement, the validation process is completed.

Two validation aspects are addressed at this point:

- first, the final sequence of events shows that there exists a sequence of basic elements implementing the upper abstract task. *This is a task validation aspect.*
- Second, if one of the basic events is not present and/or some proof obligation related to the basic events cannot be proved, in the B models of the design, then, we can assert that some of basic events are missing and/or wrongly specified and therefore, the design shall be updated and/or completed. *This is a design and architectural decomposition validation aspect.*

This approach allowed us to validate a full task model since each task can be, at the end, decomposed into an ordered sequence of basic events.

V. Application of our approach

This section applies our approach, previously described, on a simple but pedagogical case study involving all the relevant parts encountered in a WIMP (Windows, Icons, Mouse and Pointers) like HCI design.

A. The case study

The franc/euro exchange application is an application that makes conversions from French Francs to Euros and vice-versa. The user enters the value he/she wants to convert, he/she chooses the the target currency of conversion and makes the conversion, and last, user reads the result.

Figure 3 shows a screenshot of this application we have developed with Java/Swing.

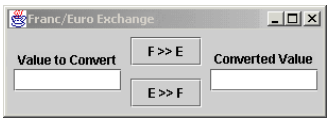

Fig. 3. Interface of franc/euro exchange application.

B. Design and architecture with B

The WIMP applications can be described by the ARCH model. This model gives a graphical representation of abstract machines and B models relations, conforming to interactive needs. Several B models and machines are built to correspond with **Domain**, **Domain Adapter**, **Dialog Controller**, **Presentation** and **Toolkit** (ARCH model elements), (see figure 4). Thus, the ARCH model explicitly includes graphical toolkits, which are required for most HCI developments. Using this framework, object oriented criteria were used to build B models, and we exploited the EXTENDS clause to ensure the consistency between these machines.

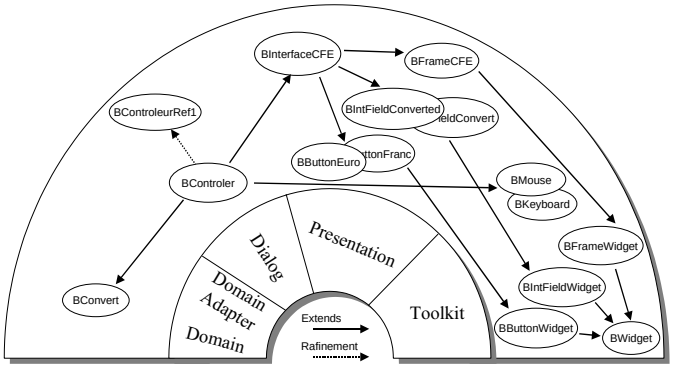

Fig. 4. Architectural ARCH-like decomposition of franc/euro exchange application.

Our design consists of sixteen B models (not completely represented on the figure 4). These machines contain the specification corresponding to:

- **Toolkit:** This module gathers a set of B models (*BButtonWidget*, *BFrameWidget*, *BIntFieldWidget*, *BScreenWidget* and *BWidget*) which describe the basic components. *BKeyboard* and *BMouse* denote the machine related to the interaction part. All these B models have been specified applying reverse-engineering rules from JAVA/Swing toolkit.
- **Presentation:** *BInterfaceCFE* specifies the interactive interface by gathering the *BButtonEuro*, *BButtonFranc*, *BIntFieldConvert*, *BIntFieldConverted*, *BFrameCFE* and *BSceen* machines.
- **Domain and Domain Adapter:** *BConvert* is a machine related to the domain and domain adapter. It manages the helpful operations which compute exchange values.
- **Dialog Controller:** This module is an event based model defined applying the event based definition of B previously addressed. The the dialog controller describes a transition system which defines the franc/euro exchange system. *BControler* and its refinement *BControlerRef1* gather the whole models into a common one in order to describe the whole application of the exchange case study. For the purpose of this paper, we limited the dialog controller to the main events like $InputValueToConvert$ (which input a real number), $ValidateInputValue$ (which sends the input to the converter), $clickEF$ (which chooses the target currency from Euros to French Francs), $DisplayConvertedValue$ (which displays the converted value) etc. Several other events are defined in the dialog controller, they are reported here. All of these events are described in the $BController$ model. Only the *clickEF* event is described.

MODEL $BController$
$\cdots$
EVENTS
$clickEF =$
SELECT
$b\_mouse\_y \in b\_frame\_y \cdots \wedge$
$b\_buttonEF\_state = unpressed \wedge$
$b\_mouse\_state = clicked \wedge$
$\cdots$
THEN
$\cdots$ $Event\ Body$
END
$\cdots$

The guard of the *clickEF* event checks that the mouse position is located on the button of conversion from French Francs to Euros, the button state is unpressed, and mouse state is clicked. So, if this guard is respected, event body is executed.

C. Task validation with B

Two tasks are validated from this case study: task of conversion from French Francs to Euros (*francsToEurosTask*) and conversely task of conversion from Euros to French Francs (*eurosToFrancsTask*). One B abstract machine (it is the higher level task) describes two operations. These operations calls one operation (opT_1 and opT_2), which describes the task postcondition.

MACHINE $TaskModel$
EXTENDS $BController$
$\cdots$
OPERATIONS
$eurosToFrancsTask =$
PRE
$\cdots$ $Task\ Precondition$
THEN
opT_1
END;
$francsToEurosTask =$
PRE
$\cdots$ $Task\ Precondition$
THEN
opT_2
END
END.

The refinement of these two operations leads to a sequence of basic events (from the dialog controller). The refinement steps introduce the *sequential* operator.

REFINEMENT $TaskModelRef1$
EXTENDS $BController$
REFINES $TaskModel$
$\cdots$
OPERATIONS
$eurosToFrancsTask =$
PRE
$\cdots$ $Task\ Precondition$
THEN
$InputValueToConvert$;
$ValidateInputValue$;
$clickEF$;
$DisplayConvertedValue$
END;
$francsToEurosTask =$
PRE
$\cdots$ $Task\ Precondition$
THEN
$\cdots$
END
END.

For example, the task of conversion from francs to euros is described by: *input value to convert* event, *validate input value* event (by pressing the enter key) and *francs to euros conversion* event (the display of converted value being done in the conversion event).

Proof of properties on the tasks *eurosToFrancsTasks* and *francsToEurosTasks* were expressed and checked thanks to *invariants* and *assertions* clauses of the B method. Among these properties, we checked that the conversion is correct (according to the conversion rate) and that the conversion is done only one since the mouse is clicked only once. The insistance property was checked as well. The appearing colors (not addressed in this paper) show the successfulness of the conversion task.

VI. Conclusion

The approach we have presented is based on the use of formal techniques for the development, verification and validation of HCI software. We have kept the traditional notations used by the designers and we have proposed a formalisation of such notations. One of the major advantages of our approach is that no change in the design is introduced. There is no fundamental change in the development process, but our contribution provides formal techniques which give a formal assistance to the HCI developers.

The formal technique we use is the proof based technique B. We have been capable to represent and verify the whole development of a HCI and its user tasks. Verification and validation are performed using proof techniques and therefore they avoid the combinatorial explosion problem devoted to most of the model checking techniques. The traces representing a task are obtained after a set of refinement steps. Task validation consists in building explicitly the user task. Therefore, this approach needs a user expertise and knowledge of the dialog controller elements. This approach is qualified as *explicit*.

In order to avoid the explicit description of the tasks, we are currently working on an implicit approach which consists in describing a task at a more abstract level using a task description language. This approach represents an upper layer of the one presented in this paper. We use a language, namely CTT, to represent tasks. The task model expressed by B models refinements (like the one presented in this paper) is generated form a CTT description. This approach allows to handle concurrency, disabling, interruption and iterative tasks which were not supported by the approach defined in this paper.

References

[1] G. Abowd, H.-M. Wang, and A. Monk, "A Formal Technique for Automated Dialogue Development," in *Proceedings of DIS'95*, G. Olsan and S. Schuon, Eds., 1995, pp. 219–226.

[2] Y. Ait-Ameur, P. Girard, and F. Jambon, "Using the B Formal Approach for Incremental Specification Design of Interactive Systems," in *IFIP TC2/WG2.7 Engineering for Human-Computer Interaction*, S. Chatty and P. Dewan, Eds. Kluwer Academic Publishers, 1998, pp. 91–110.

[3] Y. Ait-Ameur, B. Bréholée, L. Guittet, F. Jambon, and P. Girard, "Formal Verification and Validation of Interactive Systems Specifications," LISI/ENSMA, Tech. Rep., March 2000.

[4] Y. Ait-Ameur and P. Girard, "Specification, Design, Refinement and Implementation of Interactive Systems: the B Method," LISI/ENSMA, Tech. Rep., March 2001.

[5] Y. At-Ameur, P. Girard, and F. Jambon, "A uniform approach for the specification and design of interactive systems: the b method," in *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, P. Markopoulos and P. Johnson, Eds., vol. Proceedings, Abingdon, UK, 1998, pp. 333–352. [Online]. Available: http://www.lisi.ensma.fr/ftp/pub/documents/papers/1998/1998-dsvis-yaa_pg_fj.pdf

[6] G. E. Pfaff, Ed., *User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim*, ser. Eurographic Seminars. Berlin: Springer-Verlag, 1985.

[7] J. Coutaz, "PAC an Implementation Model for Dialogue Design," in *Proceedings of INTERACT*. North Holland, 1987, pp. 431–437.

[8] L. Bass, R. Pellegrino, S. Reed, S. Sheppard, and M. Szczur, "The Arch Model : Seeheim Revisited," in *CHI 91 User Interface Developper's Workshop*, 1991.

[9] D. L. Scapin and C. Pierret-Golbreich, "Towards a Method for Task Description : MAD," in *Work with display units*. Elsevier Science Publishers, North-Holland, 1990.

[10] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*. Prentice Hall, 1993.

[11] D. Rix and H. Hartson, *Developping User Interfaces: Ensuring Usability Through Product & Process*, ser. Wiley professional computing. John Wiley & Sons, inc. NY, USA, 1993.

[12] F. Patern, *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2001.

[13] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*. Prentice Hall, 1993.

[14] P. Roche, "Modélisation et Validation d'Interfaces Homme-Machine," Ph.D. dissertation, ENSAE, March 1998.

[15] J. Campos and M. Harrison, "Formally Verifying Interactive Systems: A Review." in *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'97)*. Springer Verlag, 1997, pp. 109–124.

[16] F. Paterno and G. Faconti, "On the LOTOS Use to Describe Graphical Interaction," in *Proceedings of HCI, People and Computer*. Cambridge University Press, 1992, pp. 155–173.

[17] P. Brun, "XTL: a Temporal Logic for the Formal Development of Interactive Systems," in *Formal Methods for Human-Computer Interaction*, P. Palanque and F. Paterno, Eds. Springer-Verlag, 1997, pp. 121–139.

[18] P. Palanque, R. Bastide, and V. Sengs, "Validating Interactive System Design Through the Verification of Formal Task and System Models," in *IFIP TC2/WG2.7 Engineering for Human-Computer Interaction*, 1995, pp. 189–212.

[19] B. D'Ausbourg, "Using Model Checking for the Automatic Validation of User Interface Systems," in *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*. Springer Verlag, 1998, pp. 242–260.

[20] J. M. Spivey, *The Z notation: A Reference Manual*. Prentice–Hall Int., 1988.

[21] D. Bjorner, "VDM a Formal Method at Work," in *Proc. of VDM Europe Symposium'87*, S.-V. LNCS, Ed., 1987.

[22] C. Hoare, "An Axiomatic Basis for Computer Programming," *CACM*, vol. 12, no. 10, pp. 576–583, 1969.

[23] C. Hoare, I. Hayes, H. Jifeng, C. Morgan, A. Sanders, I. Sorensen, J. Spivey, and B. Sufrin, "Laws of Programming," *CACM*, vol. 30, no. 8, 1987.

[24] J. Abrial, *The B Book. Assigning Programs to Meanings*. Cambridge University Press, 1996.

[25] E. Dijkstra, in *A Discipline of Programming*. Prentice-Hall Englewood Cliffs, 1976.

[26] D. Duke and M. D. Harrison, "Abstract Interaction Objects," in *Proceedings of Eurographics conference and computer graphics forum*, vol. 12, no. 3, 1993, pp. 25–36.

[27] D. Duke and M. Harrison, "Towards a Theory of Interactors," Amodeus Esprit Basic Research Project 7040, System Modelling/WP6, Tech. Rep., 1993.

[28] A. Hussey and D. Carrington, "Specifying a Web Browser Interface Using Object-Z," in *Formal Methods for Human-Computer Interaction*. Springer Verlag, 1997, pp. 157–174.

[29] P. Bumbulis, P. Alencar, D. Cowan, and C. Lucena, "Combining Formal Techniques and Prototyping in User Interface Construction and Verification," in *2nd Workshop on Design, Specification and Verification of Interactive Systems DSVIS*, P. Palanque and R. Bastide, Eds. Springer Verlag, 1995, pp. 174–192.

[30] ——, "Validating Properties of Component-Based Graphical User Interfaces," in *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96)*. Springer Verlag, 1996, pp. 347–365.

[31] B. Shneiderman, *Designing the User Interface*. Addison Wesley, 1998.

[32] J.-R. Abrial, "Extending b without changing it (for developing distributed systems)," in *First B Conference, Putting Into Pratice Methods and Tools for Information System Design*, H. Habrias, Ed., Nantes, France, 1996, p. 21.

[33] ClearSy, "Atelier b - version 3.5," 1997. [Online]. Available: http://www.atelierb.societe.com/