Chapter 1
State of the Art on Formal Methods for Interactive Systems

Raquel Oliveira, Philippe Palanque, Benjamin Weyers, Judy Bowen and Alan Dix

Abstract This chapter provides an overview of several formal approaches for the design, specification, and verification of interactive systems. For each approach presented, we describe how they support both modelling and verification activities. We also exemplify their use on a simple example in order to provide the reader with a better understanding of their basic concepts. It is important to note that this chapter is not self-contained and that the interested reader should get more details looking at the references provided. The chapter is organized to provide a historical perspective of the main contributions in the area of formal methods in the field of human–computer interaction. The approaches are presented in a semi-structured way identifying their contributions alongside a set of criteria. The chapter is concluded by a summary section organizing the various approaches in two summary tables reusing the criteria previously derived.

R. Oliveira
IRIT— MACAO Group, University of Toulouse 3—Paul Sabatier, Toulouse, France

P. Palanque
IRIT— Interactive Critical Systems Group, University of Toulouse 3—Paul Sabatier, Toulouse, France
e-mail: palanque@irit.fr

B. Weyers (✉)
Visual Computing Institute—Virtual Reality & Immersive Visualization, RWTH Aachen University, Aachen, Germany
e-mail: weyers@vr.rwth-aachen.de

J. Bowen
Department of Computer Science, The University of Waikato, Hamilton, New Zealand
e-mail: jbowen@waikato.ac.nz

A. Dix
School of Computer Science, University of Birmingham, Birmingham, UK
e-mail: alanjohndix@gmail.com

A. Dix
Talis Ltd., Birmingham, UK

B. Weyers et al. (eds.), *The Handbook of Formal Methods in Human-Computer Interaction*, Human-Computer Interaction Series,
DOI 10.1007/978-3-319-51838-1_1

1.1 Introduction

Building reliable interactive systems has been identified as an important and difficult task from the late 1960s on (Parnas 1969), and methods and techniques developed in computer science have been applied, adapted, or extended to fit the need of interactive systems since then. Those needs have been thoroughly studied over the years, and the complexity of interactive systems has followed or even pre-empted the non-interactive part of computing systems. Such evolution is mainly due to the technological progression of input and output devices and their related interaction techniques.

Another important aspect is related to the intrinsic nature of the interactive systems as clearly identified in Peter Wegner's paper (Wegner 1997) as the input chain is not defined prior to the execution and the output chain is processed (by the users) before the "machine" (in the meaning of Turing machine) halts.

Two books (Harrison and Thimbleby 1990; Palanque and Paternó 1997) have been published to gather contributions related to the adaptation and extension of computer science modelling and verification techniques in the field of interactive systems. Contributions in these books were covering not only the interaction side, the computation side (usually called functional core), but also the human side by presenting modelling techniques applied, for instance, to the description of the user's mental models.

Over the years, the community in Engineering Interactive Computing Systems has been investigating various ways of using Formal Methods for Interactive Systems but has also broadened that scope proposing architectures, processes, or methods addressing the needs of new application domains involving new interaction techniques. Simultaneously, the Formal Methods for Interactive Systems community has been focusing on the use of formal methods in the area of interactive computing systems.

This chapter summarises selected contributions from those two communities over the past years. For each approach presented, we describe how they both support modelling as well as verification activities. We also exemplify their use on a simple example in order to provide the reader with a better understanding of their basic concepts. It is important to note that this chapter is not self-contained and that the interested reader should get more details looking at the references provided. This chapter is organized to provide a historical perspective of the main contributions in the area of formal methods in the field of human–computer interaction. Lastly, the approaches are presented in a semi-structured way identifying their contributions alongside a set of criteria. This chapter is concluded by a summary section organizing the various approaches in two summary tables reusing the criteria previously used.

1.2 Modelling and Formal Modelling

In systems engineering, modelling activity consists of producing a theoretical view of the system under study. This modelling activity takes place using one or several notations. The notation(s) allows engineers to capture some part of the system while ignoring other ones. The resulting artefact is called a model and corresponds to a simplified view of the real system.

In the field of software engineering, modelling is a well-established practice that was very successfully adopted in the area of databases (Chen 1976). More recently, it has been widely advertised by the UML standard (Booch 2005). It is interesting to see that UML originally proposed nine different notations and thus to produce as many different models to capture the essence of software systems. SysML (OMG 2010), the recent extension to UML, proposes two additional notations to capture elements that were overlooked by UML as, for instance, a requirements' notation. Modelling is advocated to be a central part of all the activities that lead up to the production of good software (Booch 2005). It is interesting to note that recent software engineering approaches such as agile processes (Schwaber 2004) and extreme programming (Beck 1999) moved away from modelling considering that on-time delivery of software is a much more important quality than correct functioning, as bugs can always be fixed in the next delivered version.

However, building models in the analysis, specification, design, and implementation of software bring a lot of advantages (Booch 2005; Turchin and Skii 2006):

- to abstract away from low-level details;
- to focus on some aspects while avoiding others (less relevant ones);
- to describe and communicate about the system under design with the various stakeholders;
- to better understand the system under development and the choices that are made; and
- to support the identification of relationships between various components of the system.

Beyond these advantages, modelling (when supported by notations offering structuring mechanisms) helps designers to break complex applications into smaller manageable parts (Navarre et al. 2005). The extent to which a model helps in the development of human understanding is the basis for deciding how good the model is (Hallinger et al. 2000).

When the notation used for building models has rigorous theoretical foundations, these models can be analysed in order to check soundness or detect flaws. Such activity, which goes beyond modelling, is called verification and validation and is detailed in the next section.

1.3 Verification and Validation

The notation used for describing models can be at various levels of formality that can be classified as informal, semi-formal, and formal (Garavel and Graf 2013):

- Informal models are expressed using natural language or loose diagrams, charts, tables, etc. They are genuinely ambiguous, which means that different readers may have different understanding of their meaning. Those models can be parsed and analysed (e.g. spell checkers for natural text in text editors), but their ambiguity will remain and it is thus impossible to guarantee that they do not contain contradictory statements.
- Semi-formal models are expressed in a notation that has a precise syntax but has no formal (i.e. mathematically defined) semantics. Examples of semi-formal notations are UML class diagrams, data flow diagrams, entity relationship graphical notation, UML state diagrams, etc.
- Formal models are written using a notation that has a precisely defined syntax and a formal semantics. Examples of formal specification languages are algebraic data types, synchronous languages, process calculi, automata, Petri nets, etc.

Thus, formal models are built using formal notations and are unambiguous system descriptions. Such formal models can then be analysed to assess the presence or absence of properties, analyse the performance issues (if the formal notation can capture such elements), possibly simulate the models to allow designer checking their behaviour, and generate descriptions in extension (such as state-space or test cases) if the formal notation represents such elements in intention (e.g. set of states represented in intention in Petri nets while represented in extension in an automata).

Formal verification involves techniques that are strongly rooted in mathematics. Defects in models can be detected by formal verification. In such cases, either the model has to be amended (to remove the defect) or the system under analysis has to be modified, for instance, by adding barriers (Basnyat et al. 2007). Such a modified system can then be modelled and analysed again to demonstrate that the modifications have not introduced other (unexpected) problems. This cycle (presented in Fig. 1.1) is repeated until the analysis results match the expectations. Examples of formal verification techniques are model checking, equivalence checking, and theorem proving.

Theorem proving is a deductive approach for the verification of systems (Boyer and Moore 1983). Proofs are performed in the traditional mathematical style, using some formal deductive system. Both the system under verification and the properties that have to be verified are modelled usually using different types of formal notations. Properties are usually expressed using declarative formal notations (e.g. temporal logics Clarke et al. 1986) while system behaviours are usually represented using procedural formal notations such as automata. Checking that the properties are true on a formal model of the systems is done as a theorem demonstration using the deductive proof calculus (see, for instance, verification of temporal logic

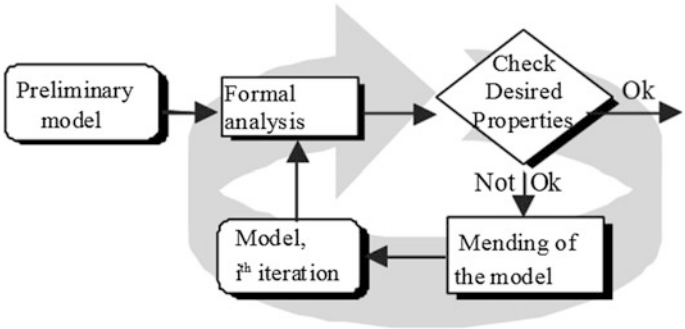


Fig. 1.1 Iterative cycle of models' construction and analysis from Palanque et al. (2009)

formulas over Petri nets Sifakis 1979). Proofs progress by transforming a set of premises into a desired conclusion, using axioms and deduction rules and possibly integrating previously demonstrated theorems. Such a proof production process is usually not fully automated: analyst guidance is required, for instance, regarding the proof strategy to be followed. Good user interfaces for theorem provers can significantly reduce the burden of the users as argued in Merriam and Harrison (1996). Some formal methods have been adapted to address the specificities of interactive systems and if they are specifically supporting theorem proving results, they have been tuned to address interactive systems properties such as the adaptation of B presented in Aït-Ameur et al. (2003a).

Model checking (Fig. 1.2) allows verification of whether a model satisfies a set of specified properties. A property is a general statement expressing an expected behaviour of the system. In model checking, a formal model of the system under analysis must be created, which is afterwards represented as a finite-state machine (FSM). This FSM is then subject to exhaustive analysis of its entire state space to determine whether the properties hold or not. The analysis can be fully automated and the validity of a property is always decidable (Cofer 2010). Even though it is easier for a human being to express properties in natural language, it can result in imprecise, unclear, and ambiguous properties. Expected properties should, thus, be also formalized by means of, for instance, a temporal logic. The analysis is mainly supported by the generation of counterexamples when a property is not satisfied. A counterexample can be a sequence of state changes that, when followed, leads to a state in which the property is false.

Since the introduction of model checking in the early 1980s, it has advanced significantly. The development of algorithmic techniques (e.g. partial-order reduction and compositional verification) and data structures (e.g. binary decision diagrams) allows for automatic and exhaustive analysis of finite-state models with several thousands of state variables (Aït-Ameur et al. 2010). For this reason, model checking has been used in the past years to verify interactive systems in

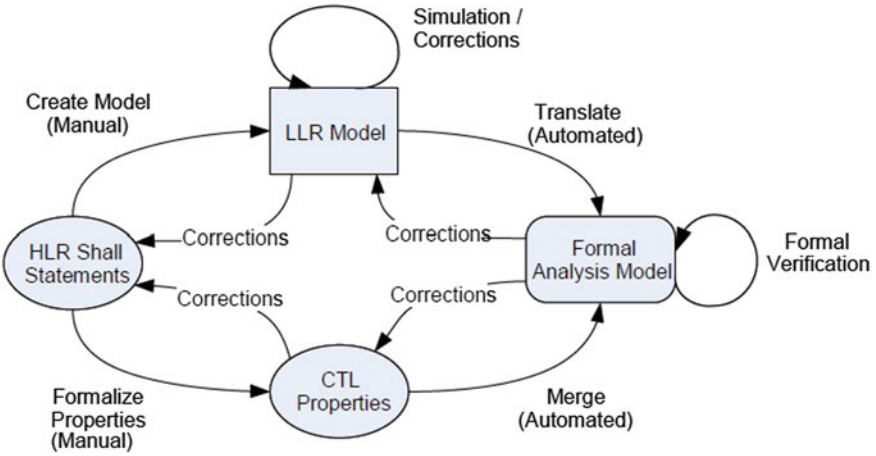


Fig. 1.2 Principle of model checking as defined in DO-178C aeronautics standard—*HLR* stands for high-level requirements, *LLR* stands for low-level requirements (and correspond to procedural systems descriptions)

safety-critical systems of several domains, such as avionics (Degani and Heymann 2002), radiation therapy (Turner 1993), and health care (Thimbleby 2010). In the field of interactive systems, several model-checking approaches have been proposed. With respect to mainstream software engineering, such approaches have been focussing on interactive systems-specific properties (such as predictability Dix 1991 or human errors identification Curzon and Blandford 2004).

Rather than verifying the satisfiability of properties, equivalence checking (Figs. 1.2 and 1.3) provides the ability to formally prove whether two representations of the system exhibit exactly the same behaviour or not. In order to verify whether two systems are equivalent or not, a model of each system should also be created, and then both models are compared in the light of a given equivalence relation. Several equivalence relations are available in the literature (e.g. strong bisimulation Park 1981 and branching bisimulation van Glabbeek and Weijland 1996). Which relation to choose depends on the level of details of the model and the verification goals. As for model checking and theorem proving, results of the analysis are exploited to identify where the models have to be amended in order to ensure their behavioural equivalence. In the field of interactive systems, this can be done for checking that two versions of interactive software exhibit the same behaviour or to check that the descriptions of user tasks are equivalent to the behaviour of the system (Palanque et al. 1995).

These three different approaches to formal verification have been applied to interactive systems in various works. In Sect. 1.6, we present those approaches by describing how formal models are described and how verification is addressed.

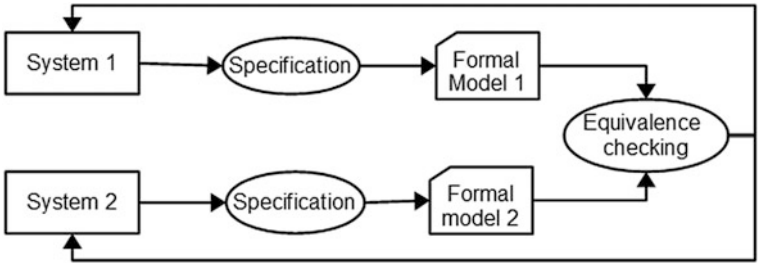


Fig. 1.3 Equivalence checking

1.4 Criteria to Describe and Analyse the State of the Art

Each approach is presented with respect to the following structure: after a brief introduction of the approach, it is unfolded step by step, identifying which language/formalism is used to model the interactive system. Then, the properties addressed by the approach are listed, together with the language/formalism used to describe them, the verification technique employed and whether the approach is tool supported or not. It is important to note that the approaches might be more powerful than presented here. Indeed, we only gather information that the authors were demonstrating in their publications. Thus, instead of presenting what an approach can do, we present what the authors have been doing with it.

After the description of each approach, an analysis is performed according to the following criteria:

- **Modelling coverage**: the verification of the system relies on the system model. For this reason, the model coverage should be large enough for the verification to be useful. It is analysed whether the studied approach covers aspects of the functional core and the user interfaces or not. The functional core of a system implements the domain-dependent concepts and functions, and the user interfaces implement the look and feel of the interactive system (Bass et al. 1991). We call a "User Interface" (UI) the information that is presented to the user with which the users can interact. In addition, it is also analysed if aspects of the users are included in the model, in order to take into account user behaviours. The more sophisticated the interaction techniques used to interact with these user interfaces, the more expressive power is required for the formal description technique. For instance, in multi-modal interactions, fusion of events is usually based on a temporal window in which the events have been received. If the events are too far away (in quantitative time), then they will not be fused. In order to describe such behaviour, the formal methods must allow engineers to describe quantitative temporal information (such as timed automata or temporal Petri nets).
- **Kinds of properties**: one kind of analysis that can be performed over a system model is property verification. In the context of safety-critical interactive

systems, we believe that the focus should be directed both towards dependability (to ensure that the functioning of the system is correct), to usability (to ensure that the system is usable; effective, efficient, and satisfactory), and to prevent users from making errors. For each author, the kinds of properties that have been demonstrated as verifiable using their approach are analysed.

- **Application to safety-critical systems**: whether each approach is applied to safety-critical domains or not. We provide here examples of the domains addressed, e.g. health care, air traffic management, avionics, or nuclear power.
- **Scalability**: while a lot of work has been performed on simple examples, we will identify approaches that have been applied to industrial applications or at least have demonstrated means (e.g. structuring mechanisms) for dealing with real-life systems.

1.5 Modelling and Verification

Interactive systems models can deal with the various aspects of interactive systems. Low-fidelity prototypes usually deal with their presentation part (i.e. how they look and how information is presented to users) while behavioural models usually address interaction or dialogue descriptions. While a model at specification level would describe **what** the system is supposed to do, models at design levels would describe **how** the system is supposed to behave. In the area of interactive systems, formal models have been proposed at different levels. Properties are closer to the specification level as they express constraints on system presentation or behaviour. A presentation property would require, for instance, that all the user interface buttons have the same size. A behavioural property, for instance, could require that all buttons are always available. Verification activity aims at assessing whether or not a property holds on a given system as discussed above.

Several authors propose different categories of properties. For instance, three kinds of properties are identified in Campos and Harrison (1997): **visibility** properties, which concern the users' perception, i.e. what is shown on the user interface and how it is shown; **reachability** properties, which concern the user interfaces, and deal with what can be done at the user interface and how it can be done (in the users' perspective); and **reliability** properties, which concern the underlying system, i.e. the behaviour of the interactive system.

1.6 Succinct Presentation of the Approaches

This section will briefly describe the approaches reviewed for this chapter. The current FoMHCI community and many of the strands of work in this review largely owe their origin to a number of projects funded by the Alvey Programme in the UK

in the 1980s, and particularly the "York Approach" (see Dix et al. below). However, it is possible to trace the roots deeper, in particular Reisner's (1981) use of BNF to describe the "action language" (what we would now call dialogue) of an interactive graphics programme, and Sufrin's (1982) use of the Z specification language to specify a simple display editor (see d'Ausbourg 1998 for an early review).

Both Reisner's and Sufrin's work used existing formal notations. This use of existing notations or creation of specialized notations or methods for interactive systems has always been one of the main strands of FoMHCI research. Many of the approaches below and in this book use existing notations (e.g. LOTOS, Petri nets); however, these often either need to extend or create new notations in order to be able to effectively specify behaviours and properties of interactive systems.

While this has emerged as the dominant strand of work in the area and the main focus of this review, there are a number of other strands that have influenced the field, elements of which can be seen in various chapters of this book (see also Dix 2012).

- **Abstract models**: this used a variety of notations, but with the aim of describing classes of systems to define generic properties and prove generic properties (see Dix et al. below). The main legacy of this approach is the formulation of properties including variations of predictability and observability that are adopted by many system modelling approaches, which can be seen in many of the approaches below.
- **Architectural models**: early user interface implementers reflected on their experience. The MVC (Model–View–Controller) paradigm grew out of the Smalltalk programming environment (Kieras and Polson 1985), and a workshop of those developing User Interface Management Systems (UIMS) led to the Seeheim Model (Pfaff and Hagen 1985). The former has been particularly influential in subsequent practical UI development, and the latter in framing a language for interaction architecture, especially the formulation of the presentation–dialogue-functionality distinction. Within the formal modelling community, this work was especially strongly associated with the work of Coutaz, Nigay, and others at Grenoble including the development of the PAC model (Coutaz 1987), which itself fed into the ARCH/Slinky metamodel (Bass et al. 1991). The main legacy of this work has been in its inputs into modelling of multi-modal systems and plasticity. Oddly, many current systems that describe themselves as MVC are actually unintentionally following PAC model (Dey 2011).
- **User and task modelling**: the cognitive modelling and task analysis communities have often used models that have a formal nature, although come from different roots and have had different concerns to those adopting a more computer science formal modelling approach. However, there have been many overlaps including CCT (Cognitive Complexity Theory), which used a dual system and cognitive model (Kieras and Polson 1985), and TAG (Task Action Grammar), which expressed system descriptions in ways that made

inconsistencies obvious (Payne and Green 1986). Of course, the CTT task modelling approach (see Paterno et al. below) has been very influential in the FoMHCI community, and, while having its roots in the LOTOS specification notation, it is very similar to pure task analysis notations such as HTA (Shepherd 1989).

1.6.1 Abowd et al. (USA 1991–1995)

Early approaches to applying formal notations to the study of human–machine interaction and the modelling of interactive systems paved the way for other researchers to explore different alternatives to assess the quality of such systems. In Abowd (1991), a framework for the formal description of users, systems, and user interfaces is proposed.

1.6.1.1 Modelling

In Abowd (1991), the interactive system is modelled as a collection of agents. The language to describe the agents borrows notations from several formal languages, such as Z, VDM, CSP, and CSS. Such agent language contains identifiers to describe internal (*types, attributes, invariants, initially*, and *operations*) and external specifications of agents, as well as communication between agents (input/output). Therefore, data, states, and events can be modelled in this language. When describing the *operations* agents can execute, it is possible to define *pre-* and *post-conditions* for each operation, which may be used to define a given ordering of actions, allowing *qualitative time* to be represented. The external specification of the language allows description of synchronous parallel composition, which can express *concurrent behaviour.* Finally, multi-touch interactions can be implicitly modelled by agents: each finger could be represented by a given agent.

Alternatively, another approach is proposed in Abowd et al. (1995), Wang and Abowd (1994), in which interactive systems are described by means of a tabular interface using Action Simulator, a tool for describing PPS (propositional production system) specifications. In PPS, the dialogue model is specified as a number of production rules using pre- and post-conditions. Action Simulator permits such PPS specification to be represented in a tabular format, in which the columns are the system states, and the production rules are expressed at the crossings of lines and columns. It is possible to represent *multi-modality* using this approach by identifying the states related to each modality, and how they relate to each other using the production rules. However, it does not allow *concurrent behaviour* to be expressed: production rules are executed sequentially. The approach covers the modelling of the functional core and the UIs, and to some extent the modelling of the users, by describing the user actions that "fire" the system state changes.

1.6.1.2 Verification

The verification of specifications written using the agent language described in Abowd (1991) can be tool supported, for instance, by using ProZ for Z specifications. However, such verification is not described in Abowd (1991).

A translation from the tabular specification of the interactive system proposed in (Abowd et al. 1995; Wang and Abowd 1994) into SMV input language is described in Wang and Abowd (1994). The CTL temporal language is used to formalize the properties, allowing the following usability properties to be verified:

- **Reversibility (Abowd et al**. 1995): Can the effect of a given action be reversed in a single action?
- **Deadlock freedom (Abowd et al.** 1995): From an initial state, is it true that the dialogue will never get into a state in which no actions can be taken?
- **Undo within N steps (Wang and Abowd** 1994): From any state of a given state set, if the next step leads the system out of the state set, can a user go back to the given state set within N steps?

In addition, the following functional properties can be verified:

- **Rule set connectedness (Abowd et al.** 1995): From an initial state, can an action be enabled?
- **State avoidability (Wang and Abowd** 1994): Can a user go from one state to another without entering some undesired state?
- **Accessibility (Wang and Abowd** 1994): From any reachable state, can the user find some way to reach some critical state set (such as the help system)?
- **Event constraint (Wang and Abowd** 1994): Does the dialogue model ensure/prohibit a particular user action for a given state set?
- **Feature assurance (Wang and Abowd** 1994): Does the dialogue model guarantee a desired feature in a given state set?
- **Weak task completeness (Abowd et al.** 1995): Can a user find some way to accomplish a goal from initialization?
- **Strong task completeness (Abowd et al.** 1995): Does the dialogue model ensure that a user can always accomplish a goal?
- **State inevitability (Abowd et al.** 1995): From any state in the dialogue, will the model always allow the user to get to some critical state?
- **Strong task connectedness (Abowd et al.** 1995): From any state, can the user find some way to get to a goal state via a particular action?

The automatic translation of the tabular format of the system states into the SMV input language is an advantage of the approach, since it allows model checking of properties to be performed. The tabular format of the system states and the actions that trigger state changes provide a reasonable compact representation in a comprehensible form. However, it looks like the approach does not scale well to larger specifications, unless an alternative way to store a large sparse matrix is provided. Besides, no application to safety-critical systems is reported.

1.6.2 Dix et al. (United Kingdom 1985–1995)

1.6.2.1 Modelling

The PIE model (Dix et al. 1987) considers interactive systems as a "black-box" entity that receives a sequence of inputs (keystrokes, clicks, etc.) and produces a sequence of perceivable effects (displays, LEDs, printed documents, etc.). The main idea is to describe the user interfaces in terms of the possible inputs and their effects (Dix 1991). Such practice is called surface philosophy (Dix 1988) and aims at omitting parts of the system that are not apparent to the user (the internal details of systems, such as hardware characteristics, languages used, or specification notations). The domain of input sequences is called P (standing for programs), the domain of effects is called E, and both are related by an interpretation function I that determines the effects of every possible command sequence (Fig. 1.4). In this sense, the interpretation function I can be seen as a means to represent *events* of the modelled system, *data* cannot be represented, and internal *states* of the system are inferred by what is called *observable effects* (Dix 1991).

The effects E can be divided into permanent results (e.g. printout) and ephemeral displays (the actual UI image). Such specialization of the effects constitutes another version of the PIE model, called the Red-PIE model (Dix 1991).

The PIE model is a single-user single-machine model and does not describe interleaving and the timing of the input/output events (Dix 1991). However, extensions of the basic PIE model dealt with *multi-user* behaviour (including the first formulations of collaborative undo Abowd and Dix 1992); the first formal work on *real-time* interactive behaviours (Dix 1991); continuous interaction (such as mouse dragging) through status-event analysis (Dix 1991); and non-deterministic external behaviours (e.g. due to concurrency or race conditions) (Dix 1991). *Multi-modality* can be expressed by describing the input/output and interpretation function for each modality, the status–event analysis extensions would allow multi-touch applications.

The PIE model is focused on the external behaviour of the system as perceived by the user; it does not model the users themselves, nor more than minimal internal details. Because of this, the external effects of internal behaviour such as *concurrency behaviour* or *dynamic instantiation* can be modelled, but not their internal mechanisms.

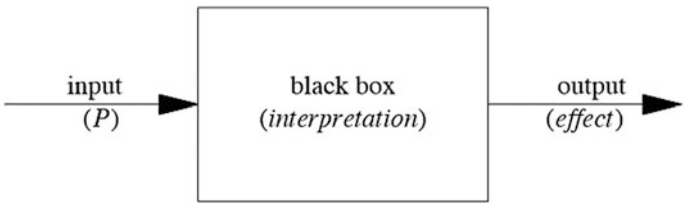


Fig. 1.4 The PIE model (Dix 1991)

1.6.2.2 Verification

The PIE model provides a generic way of modelling interactive systems and permits the following usability properties to be formalized:

- **Predictability (Dix** 1995**)**: The UI shall be predictable, i.e., from the current effect, it should be possible to predict the effect of future commands.
- **Simple reachability (Dix** 1991**)**: All system effects can be obtained by applying some sequences of commands;
- **Strong reachability (Dix** 1988**)**: One can get anywhere from anywhere;
- **Undoability (Dix et al.** 1987**)**: For every command sequence, there is a function "undo" which reverses the effect of any command sequence;
- **Result commutativity (Dix et al.** 1987**)**: Irrespective of the order in which different UIs are used, the result is the same.

The PIE and Red-PIE models are one of the first approaches that used formal notations for the modelling of interactive systems and desired properties. As abstract models, their role in respect of verification is therefore more in formulating the user interaction properties that subsequent system modelling and specification approaches (such as Abowd et al., above and Paterno et al., below) seek to verify for specific systems.

Some proofs and reasoning about PIEs are quite extensive, notably Mancini's category theoretical proof of the universality of stack-and-toggle-based undo (Mancini 1997). However, the mathematical notations are very abstract, and no tool support is provided; instead, proofs follow a more traditional mathematical form.

1.6.3 Paternò et al. (Italy 1990–2003)

1.6.3.1 Modelling

Interactive systems can be formally described as a composition of interactors (Hardin et al. 2009). Interactors are more concrete than the agent model described in section (Abowd 1991 above), in that they introduce more structure to the specification by describing an interactive system as a composition of independent entities (Markopoulos 1997).

The interactors of CNUCE (Paternó and Faconti 1992) provide a communication means between the user and the system. Data manipulated by the interactors can be sent and received through events in both directions: towards the system and towards the user (Paternó 1994), which are both abstracted in the model by a description of the possible system and user actions.

The CNUCE interactors are specified using LOTOS (ISO 1989), which has concurrent constructs. However, since LOTOS is a language with action-based semantics, the system states cannot be represented. Besides, only *qualitative time* can be modelled, *dynamic instantiation* cannot be modelled, neither *multi-modality*.

Multi-touch interactions can be modelled by defining one interactor for each finger, and by integrating these interactors to other interactors of the system. Despite the fact that the approach covers mainly the modelling of user interfaces, a mathematical framework is provided to illustrate how to model the user and the functional core too (Paternó 1994).

1.6.3.2 Verification

Figure 1.5 illustrates how a formal model using CNUCE interactors can be used afterwards for verification (Paternó 1997).

This approach has been used to verify the following usability properties:

- **Visibility (Paternó and Mezzanotte** 1994**)**: Each user action is associated with a modification of the presentation of the user interface to give feedback on the user input;
- **Continuous feedback (Paternó and Mezzanotte** 1994**)**: This property is stronger than visibility; besides requiring a feedback associated with all possible user actions, this has to occur before any new user action is performed;
- **Reversibility (Paternó and Mezzanotte** 1994**)**: This property is a generalization of the undo concept. It means that users can perform parts of the actions needed to fulfil a task and then perform them again, if necessary, before the task is completed in order to modify its result;
- **Existence of messages explaining user errors (Paternó and Mezzanotte** 1994**)**: Whenever there is a specific error event, a help window will appear.

In addition, the following functional property can be verified:

- **Reachability (Paternó and Mezzanotte** 1994**)**: This property verifies that a user interaction can generate an effect on a specific part of the user interface.

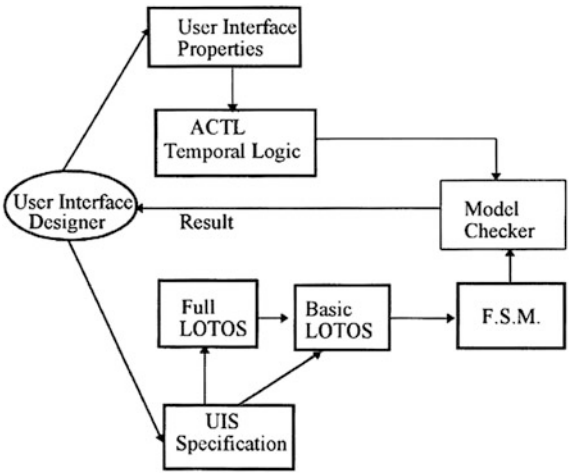


Fig. 1.5 The TLIM (tasks, LOTOS, interactors modelling) approach (Paternó 1997)

The approach has been applied to several case studies of safety-critical systems in the avionics domain (Navarre et al. 2001; Paternó and Mezzanotte 1994, 1996; Paternó 1997; Paternó and Santoro 2001, 2003). These examples show that the approach scales well to real-life applications. Large formal specifications are obtained, which describe the behaviour of the system, permitting meaningful properties to be verified.

1.6.4 Markopoulos et al. (United Kingdom 1995–1998)

1.6.4.1 Modelling

ADC (Abstraction–Display–Controller) (Markopoulos 1995) is an interactor model that also uses LOTOS to specify the interactive system (specifically, the UIs). In addition, the expression of properties is facilitated by templates.

The ADC interactor handles two types of data: display data, which come (and are sent to) either directly from the UI or indirectly through other interactors, and abstraction data, which are sustained by the interactor to provide input to the application or to other interactors (Markopoulos et al. 1998). A UI can be modelled as a composition of ADC interactors. Once formalized in LOTOS, the ADC interactors can be used to perform formal verification of usability properties using model checking.

The ADC approach concerns mostly the formal representation of the interactor model. Regarding the coverage of the model, the focus is to provide an architectural model for user interface software. The functional core and the user modelling are not covered. ADC emphasizes the architectural elements of the interactor: its gates, their role, their grouping to sides, the separate treatment of dialogue and data modelling and the composition of interactors to form complex interface specifications (Markopoulos et al. 1998). When connected to each other, ADC interactors exchange data through gates. Connection types (*aout, dout*), (*dout, dout*), and (*aout, aout*) concern pairs of interactors which synchronize over common output gates. These can be useful for modelling multi-modal output where different output modalities synchronize, e.g. sound and video output (Markopoulos et al. 1998). Touch interactions can also be modelled by the combination of such interactors.

1.6.4.2 Verification

The properties to be verified over the formal model are specified in the ACTL temporal logic. For example, the following properties can be verified:

- **Determinism (Markopoulos** 1997): A user action, in a given context, has only one possible outcome;

- **Restartability (Markopoulos** 1995): A command sequence is restartable if it is possible to extend it so that it returns to the initial state;
- **Undoability (Markopoulos** 1995): Any command followed by undo should leave the system in the same state as before the command (single step undo);
- **Eventual feedback (Markopoulos et al.** 1998): A user-input action shall eventually generate a feedback.

In addition, the following functional properties can be verified:

- **Completeness (Markopoulos** 1997): The specification has defined all intended and plausible interactions of the user with the interface;
- **Reachability (Markopoulos** 1997): It qualifies the possibility and ease of reaching a target state, or a set of states, from an initial state, or a set of states.

In this approach, the CADP (Garavel et al. 2013) toolbox is used to verify properties by model checking (Markopoulos et al. 1996). Specific tools to support the formal specification of ADC interactors are not provided (Markopoulos et al. 1998).

No case study applying the approach to the verification of critical systems is reported. In fact, the approach is applied to several example systems (Markopoulos 1995) and to a case study on a graphical interface of Simple Player for playing movies (Markopoulos et al. 1996), which makes it difficult to measure whether it can scale up to realistic applications or not.

1.6.5 Duke and Harrison et al. (United Kingdom 1993–1995)

1.6.5.1 Modelling

Another interactor model is proposed by the University of York (Duke and Harrison 1995) to represent interactive systems. Compared to the CNUCE interactor model (below), the main enhancement brought by the interactors of York is an explicit representation of the state of the interactor.

The York interactor (Fig. 1.6) has an internal state and a rendering function (i.e. rho in Fig. 1.6) that provides the environment with a perceivable representation (P) of the interactor internal state. The interactor communicates with the environment by means of events. Two kinds of events are modelled:

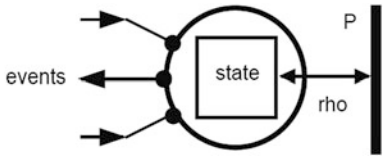


Fig. 1.6 The York interactor (Harrison and Duke 1995)

- **Stimuli events**: They come from either the user or the environment and modify the internal state of the interactor. Such state changes are then reflected to the external presentation through the rendering function.
- **Response events**: These events are generated by the interactor and sent to the user or to the environment.

York interactors are described using the Z notation (Spivey 1989). This notation facilitates the modelling of the state and operations of a system, by specifying it as a partially ordered set of events in first-order logic (Duke and Harrison 1993). Multi-modality can be represented (Duke and Harrison 1995) as each action has an associated modality from the given set *[modality]*. Besides, the authors describe two approaches to define a notion of interaction: the *one-level model* and the *two-level model*, which bound several interactors (Duke and Harrison 1993). This allows multi-touch interactions to be represented by this approach. In addition, the York interactor model provides an abstract framework for structuring the description of interactive systems in terms of layers. It encapsulates two specific system layers: the state and the display (Harrison and Duke 1995), thus covering both the functional core and the UIs in the modelling. However, *concurrent behaviour* cannot be expressed, neither does it support *dynamic instantiation* (even though instantiation is proposed to compose interactors (Duke and Harrison 1993), it seems that the interactors are not dynamically instantiated).

1.6.5.2 Verification

This approach permits usability properties expressed in first-order logic formulas to be verified. Unlike the previous approaches, the York interactor model uses theorem proving as formal verification technique. Examples of properties that can be verified are (Duke and Harrison 1995):

- **Honesty**: The effects of a command are intermediately made visible to the user;
- **Weak reachability**: It is possible to reach any state through some interaction;
- **Strong reachability**: Each state can be reached after any interaction p; and
- **Restartability**: Any interaction p is a prefix of another q such that q can achieve any of the states that p initially achieves.

The approach is applied to a case study in a safety-critical system, an aircraft's fuel system (Fields et al. 1995) in which the pilot's behaviour is modelled, thus showing that the approach also covers the modelling of users. No further case studies applying the approach were found in the literature, which makes it difficult to tell whether the approach scales up to larger interactive systems or not.

1.6.6 Campos et al. (Portugal 1997–2015)

1.6.6.1 Modelling

The York interactor model is the basis of the work proposed in Bumbulis et al. (1995b). Here, Campos chooses MAL (Modal Action Logic) language to implement the York interactor model, since MAL's structure facilitates the modelling of the interactor behaviour. The use of MAL allows *data* to be represented, since *attributes* can be expressed in the language. In Campos and Harrison (2001), the authors propose the MAL interactor language to describe interactors that are based on MAL, propositional logic is augmented with the notion of action, and deontic operators allows ordering of actions to be expressed.

The approach covers the three aspects we are considering: in Campos and Harrison (2011), the approach is used to model the functional core and user interfaces of an infusion pump; and assumptions about user behaviours are covered in Campos and Harrison (2007), by strengthening the preconditions on the actions the user might execute.

A tool called i2smv is proposed in Campos and Harrison (2001) to translate MAL specifications into the input language of the SMV model checker. However, *concurrent behaviour* cannot be modelled. Although the stuttering in the SMV modules allows interactors to evolve independently, a SMV module will engage in an event while another module does nothing (Campos and Harrison 2001).

1.6.6.2 Verification

The approach is applied to several case studies. An application of both model checking and theorem proving to a common case study is described. Further, deeper investigations are performed (and tools developed) into the usage of model checking (only), in order to verify interactive systems.

To support the whole process, a toolbox called IVY is developed (Campos and Harrison 2009). In this framework, the properties are specified using the CTL (computational tree logic) temporal logic, allowing the verification of usability and functional properties (Campos and Harrison 2008). Particularly, the following usability properties can be expressed:

- **Feedback (Campos and Harrison** 2008**)**: A given action provides a response;
- **Behavioural consistency (Campos and Harrison** 2008**)**: A given action causes consistent effect;
- **Reversibility (Campos and Harrison** 2008**)**: The effect of an action can be eventually reversed/undone;
- **Completeness (Campos and Harrison** 2009**)**: One can reach all possible states with one action.

The approach is applied to several case studies (Campos and Harrison 2001; Harrison et al. 2013), specifically in safety-critical systems (e.g. healthcare systems Campos 1999; Campos and Harrison 2009, 2011; Harrison et al. 2015 and avionics systems Campos and Harrison 2007; Doherty et al. 1998; Sousa et al. 2014), showing that the approach scales well to real-life applications.

1.6.7 d'Ausbourg et al. (France 1996–2002)

1.6.7.1 Modelling

Another approach based on the York interactor model is proposed in d'Ausbourg (1998) and d'Ausbourg et al. (1998). These authors push further the modelling of an interactive system by events and states initially proposed by the York approach.

Their interactor model is called CERT. It also contains an internal state, and the interface between an interactor and its environment consists of a set of input and output events. Both internal state and events are described as Boolean flows. Such representation of interactors by flows allows their specification using the LUSTRE data flow language. A system described in LUSTRE is represented as a network of nodes acting in parallel, which allows *concurrent behaviour* to be represented. Each node transforms input flows into output flows at each clock tick.

The approach can handle *data* in the system modelling. However, a drawback is that it does not handle sophisticated data types. The representation of the internal system state and events by Boolean flows considerably limits the modelling capabilities of the approach. In LUSTRE, a flow variable is a function of time, denoting the sequence of values that it takes at each instant (d'Ausbourg et al. 1998). Specifically, two LUSTRE operators allow *qualitative time* to be represented: the "previous" operator *pre* and the "followed-by" operator $\rightarrow$. Besides, *quantitative time* can also be represented: the expression *occur-from-to(a, b, c)* is a temporal operator whose output is true when "*a*" occurs at least once in the time interval [b…c] (d'Ausbourg et al. 1998).

1.6.7.2 Verification

The LUSTRE formal model is then verified by model checking. Verification is achieved by augmenting the system model with LUSTRE nodes describing the intended properties, and using the Lesar tool to traverse the state space generated from this new system. The properties can be either specific or generic properties.

Specific properties deal with how presentations, states, and events are dynamically linked into the UIs, and they are automatically generated from the UIL file (they correspond to functional properties). Generic properties might be checked on any user interface system, and they are manually specified (they correspond to usability properties). The verification process allows the generation of test cases,

using the behaviour traces that lead to particular configurations of the UI where the properties are satisfied.

In particular, the following usability properties are verified (d'Ausbourg et al. 1998):

- **Reactivity**: The UI emits a feedback on each user action;
- **Conformity**: The presentation of an interactor is modified when its internal state changes;
- **Deadlock freedom**: The impossibility for a user to get into a state where no actions can be taken;
- **Unavoidable interactor**: The user must interact with the interactor at least once in any interactive session of the UIs.

As well as the following functional property:

- **Rule set connectedness**: An interactor is reachable from any initial state.

The approach was applied to the avionics field (d'Ausbourg 2002). In this case study, the interactions of the pilot with the system and the behaviour of the functional core are modelled. Unfortunately, no evidence is given that the approach scales well to real-life applications.

1.6.8 Bumbulis et al. (Canada 1995–1996)

1.6.8.1 Modelling

Similar to the interactor models (Campos and Harrison 2001; d'Ausbourg et al. 1998; Duke and Harrison 1995), user interfaces can be described by a set of interconnected primitive components (Brat et al. 2013; Bumbulis et al. 1995a). The notion of component is similar to that of interactor, but a component is more closely related to the widgets of the UI. Such component-based approach allows both rapid prototyping and formal verification of user interfaces from a single UI specification.

In Bumbulis et al.'s approach, user interfaces are described as a hierarchy of interconnected component instances using the Interconnection Language (IL). Investigations have been conducted into the automatic generation of IL specifications by re-engineering the UIs (Bumbulis et al. 1995a). However, such automatic generation is not described in the paper. From such component-based IL specification of the UI, a Tcl/Tk code is mechanically generated, in order to provide a UI prototype for experimentation, as well as a HOL (higher-order logic) specification for formal reasoning using theorem proving (Bumbulis et al. 1995a).

The approach covers only the modelling and verification of user interfaces. The user and the functional core are not modelled. Besides, the Interconnection Language does not provide means to represent *multi-modality*, *multi-touch* interactions, *concurrent behaviour*, or *time.*

1.6.8.2 Verification

Properties are specified as predicates in Hoare logic, a formal system with a set of logical rules for reasoning about the correctness of computer programs. Proofs are constructed manually, even though investigations to mechanize the process have been conducted (Bumbulis et al. 1995a). No usability properties are verified in this approach. Instead, the approach permits functional properties to be verified, which are directly related to the expected behaviour of the modelled UI.

No application to safety-critical systems was found in the literature. Besides, it is not clear how to model more complex UIs in this approach, since UI components are not always bound to each other. In addition, it is not clear how multiple UIs could be modelled, neither the navigation modelling between such UIs. All these aspects indicate that the approach does not scale well for larger applications.

1.6.9 Oliveira et al. (France 2012–2015)

1.6.9.1 Modelling

In Oliveira et al. (2015a), a generic approach to verifying interactive systems is proposed, but instead of using interactors, interactive systems are described as a composition of *modules*. Each system component is described as such a *module*, which communicates and exchanges information through channels. This approach allows *plastic* UIs to be analysed. *Plasticity* is the capacity of a UI to withstand variations in its *context of use* (environment, user, platform) while preserving usability (Thevenin and Coutaz 1999). In this approach, interactive systems are modelled according to the principles of the ARCH architecture (Bass et al. 1991), and using LNT (Champelovier 2010), a formal specification language derived from the ELOTOS standard (ISO 2001). LNT improves LOTOS (ISO 1989) and can be translated to LOTOS automatically. LOTOS and LNT are equivalent with respect to expressiveness, but have a different syntax. In Paternó (1997), the authors point out how difficult it is to model a system using LOTOS, when quite simple UI behaviours can easily generate complex LOTOS expressions. The use of LNT alleviates this difficulty.

The approach enhances standard LTS to model interactive systems. An LTS represents a system by a graph composed of states and transitions between states. Transitions between states are triggered by actions, which are represented in LTS transitions as labels. Intuitively, an LTS represents all possible evolutions of a system modelled by a formal model. The approach enhances LTS by proposing the ISLTS (Interactive System LTS) (Oliveira 2015), in which two new sets are added: a set C of UI components and a set L of action names. In addition, the set A of actions of standard LTS is enhanced to carry a list of UI components, representing the UI appearance after the action is performed.

The approach covers aspects of the users, the user interfaces, and the functional core of the system. *Data* and *events* of the system can be modelled by the ISLTS, but not the *state* of the system. The ordering of transitions of the LTS can represent *qualitative time* between two consecutive model elements, and LNT contains operators that allow *concurrent behaviour* to be modelled. Models described with this approach were of WIMP-type, and no evidence was given about the ability of the approach to deal with more complex interaction techniques such as multi-touch or multi-modal UIs (especially, dealing with quantitative time required for fusion engines).

1.6.9.2 Verification

The approach is twofold, allowing: *usability* and *functional* properties to be verified over the system model (Oliveira et al. 2014). Using *model checking,* usability properties verify whether the system follows ergonomic properties to ensure a good usability. Functional properties verify whether the system follows the requirements that specify its expected behaviour. These properties are formalized using the MCL property language (Mateescu and Thivolle 2008). MCL is an enhancement of the modal mu-calculus, a fixed point-based logic that subsumes many other temporal logics, aiming at improving the expressiveness and conciseness of formulas.

Besides, different versions of UIs can be compared (Oliveira et al. 2015b). Using *equivalence checking*, the approach verifies to which extent UIs present the same interaction capabilities and appearance, showing whether two UI models are equivalent or not. When they are not equivalent, the UI divergences are listed, providing the possibility of leaving them out of the analysis (Oliveira et al. 2015c). In this case, the two UIs are equivalent less such divergences. Furthermore, the approach shows that one UI can contain at least all interaction capabilities of another (UI inclusion). Three abstraction techniques support the comparison: *omission*, *generalization*, and *elimination*. This part of the approach can be used to reason of *multi-modal* user interfaces, by verifying the level of equivalence between them.

The approach is supported by CADP (Garavel et al. 2013), a toolbox for verifying asynchronous concurrent systems: systems whose components may operate at different speeds, without a global clock to synchronize them. Asynchronous systems suit the modelling of human–computer interactions well: the modules that describe the users, the functional core, and the user interfaces can evolve in time at different speeds, which reflects well the unordered sequence of events that take place in human–machine interactions. Both parts of the approach can be used either independently or in an integrated way, and it has been validated in three industrial case studies in the nuclear power plant domain, which indicates the potential of the approach with respect to scalability (Oliveira 2015).

1.6.10 Knight et al. (USA 1992–2010)

1.6.10.1 Modelling

Another example of the application of formal methods to safety-critical systems, specifically, to the nuclear power plant domain, can be found in Knight and Brilliant (1997). The authors propose the modelling of user interfaces in three levels: *lexical*, *syntactic*, and *semantic* levels. Different formalisms are used to describe each level. For instance, the lexical level is defined using Borland's OWL (*Object Windows Library)*, allowing *data* and *events* to be represented. The syntactic level in the approach is documented with a set of context-free grammars with one grammar for each of the concurrent, asynchronous dialogues that might be taking place. Such syntactic level imposes the required temporal ordering on user actions and system responses (Knight and Brilliant 1997). Finally, Z is used to define the semantic level. The notion of user interfaces as a dialogue between the operator and the computer system consisting of three components (lexical, syntactic, and semantic levels) is proposed by Foley and Wallace (1974).

Each of these three levels is specified separately. Since different notations are used, the communication between these levels is defined by a set of tokens (Knight and Brilliant 1997). The concept of a multi-party grammar is appropriate for representing grammars in which tokens are generated by more than one source (Knight and Brilliant 1997). Such representation could allow multi-modality to be covered by the approach. However, the authors have elected to use a conventional context-free grammar representation together with a naming convention to distinguish sources of tokens (Knight and Brilliant 1997).

Following this view of user interface structure, the authors develop a formal specification of a research reactor used in the University of Virginia Reactor (UVAR) for training nuclear engineering students, radiation damage studies, and other studies (Loer and Harrison 2000). In order to illustrate the specification layers, the authors focus on the safety control rod system, one of the reactor subsystems. They give in the paper the three specifications for this subsystem.

The approach is also applied to other safety-critical systems, such as the Magnetic Stereotaxis System (MSS), a healthcare application for performing human neurosurgery (Elder and Knight 1995; Knight and Kienzle 1992). UIs, users, and the functional core of systems are covered by this approach. The UI syntactic level in their approach defines valid sequences of user inputs on the UIs, which is to some extent the modelling of the users, and the cypher system case study described in Yin et al. (2008) verifies the correctness of the functional core. Finally, the approach covers the representation of *dynamic reconfiguration* (Knight and Brilliant 1997).

1.6.10.2 Verification

However, the formal specification is not used to perform formal verification. According to the authors, the main goal is to develop a formal specification approach for user interfaces of safety-critical systems. Concerning verifiability, the authors claim that the verification of a UI specification using this approach is simplified by the use of an executable specification for the lexical level and by the use of a notation from which an implementation can be synthesized for the syntactic level. For the semantic level, they argue that all the tools and techniques developed for Z can be applied (Knight and Brilliant 1997).

Later, a toolbox called Zeus is proposed to support the Z notation (Knight et al. 1999). The tool permits the creation and analysis of Z documents, including syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving. The tool is evaluated in a development of a relatively large specification of an international maritime software standard, showing that Zeus meets the expected requirements (Knight et al. 1999).

Following such a separation of concerns in three levels, the authors propose another approach called Echo (Strunk et al. 2005), and this time applied to a case study in the avionics domain. In order to decrease complexity with traditional correctness proofs, the Echo approach is based on the refactoring of the formal specification (Yin et al. 2009a, b), reducing the verification burden by distributing it over separate tools and techniques. The system model to be verified (written in PVS) is mechanically refactored. It is refined into an implementable specification in Spark Ada by removing any un-implementable semantics. After refactoring, the model is documented with low-level annotations, and a specification in PVS is extracted mechanically (Yin et al. 2008). Proofs that the semantics of the refactored model is equivalent to that of the original system model, that the code conforms to the annotations, and that the extracted specification implies the original system model constitute the verification argument (Yin et al. 2009a).

An extension of the approach is proposed in Yin and Knight (2010), aiming at facilitating formal verification of large software systems by a technique called proof by parts, which improve the scalability of the approach for larger case studies.

The authors did not clearly define the kinds of properties they can verify over interactive systems with their approach. The case studies to which the approach is applied mainly focused on the benefits of modelling UIs in three layers using formal notation.

1.6.11 Miller et al. (USA 1995–2013)

1.6.11.1 Modelling

Also in the safety-critical domain, but in avionics, deep investigation has been conducted at Rockwell Collins of the usage of formal methods for industrial

realistic case studies. Preliminary usage of formal methods aimed at creating consistent and verifiable system specifications (Hamilton et al. 1995), paving the way to the usage of formal methods at Rockwell Collins. Another preliminary use of formal methods was the usage of a synchronous language called RSML (Requirements State Machine Language) to specify requirements of a flight guidance system. RSML is a *state-based* specification language developed by Leveson's group at the University of California at Irvine as a language for specifying the behaviour of process control systems (Miller et al. 2006). Algorithms to translate specifications from this language to the input languages of the NuSMV model checker and the PVS theorem prover have been proposed (Miller et al. 2006), enabling one to perform verification of safety properties and functional requirements expressed in the CTL temporal logic (i.e. functional properties). Afterwards, deeper investigations are conducted to further facilitate the usage of formal methods.

According to Miller (2009), relatively few case studies of model checking to industrial problems outside the field of engineering equipment are reported. One of the reasons is the gap between the descriptive notations most widely used by software developers and the notations required by formal methods (Lutz 2000). To alleviate the difficulties, as part of NASA's Aviation Safety Program (AvSP), Rockwell Collins and the research group on critical systems of the University of Minnesota (USA) develop the Rockwell Collins Gryphon Translator Framework (Hardin et al. 2009), providing a bridge between some commercial modelling languages and various model checkers and theorem provers (Miller et al. 2010). The translation framework supports Simulink, Stateflow, and SCADE models, and it generates specifications for the NuSMV, Prover, and SAL model checkers, the ACL2 and PVS theorem provers, and generates C and Ada code (Miller et al. 2010) (BAT and Kind are also included as target model checkers in Cofer et al. 2012). Alternatively, Z specifications are also covered by the approach as an input language, since Simulink and Stateflow models can be derived from Z specifications (Hardin et al. 2009).

Algorithms to deal with the time dependencies were implemented in the translator, allowing multiple input events arriving at the same time to be handled (Miller et al. 2006). Concerning the modelling coverage, the approach covers only the functional core of the avionics interactive systems that were analysed (Combéfis 2013; Miller 2009; Miller et al. 2010), but not the user interfaces nor the user behaviour.

Tools were also developed to translate the counterexamples produced by the model checkers back to Simulink and Stateflow models (Cofer 2012), since for large systems it can be difficult to determine the cause of the violation of the property only by examining counterexamples (Whalen et al. 2008).

1.6.11.2 Verification

The technique is applied to several case studies in avionics (Cofer 2012; Combéfis 2013; Miller 2009; Miller et al. 2010; Whalen et al. 2008). The first application of the NuSMV model checker to an actual product at Rockwell Collins is the mode logic of the FCS 5000 Flight Control System (Miller 2009): 26 errors are found in the mode logic.

The largest and most successful application is the Rockwell Collins ADGS-2100 (Adaptive Display and Guidance System Window Manager), a cockpit system that provides displays and display management software for commercial aircraft (Miller et al. 2010). The Window Manager (WM) ensures that data from different applications are displayed correctly on the display panel. A set of properties that formally expresses the WM requirements (i.e. functional properties) is developed in the CTL and LTL temporal logic: 563 properties are developed and verified, and 98 design errors are found and corrected.

The approach is also applied to an adaptive flight control system prototype for unmanned aircraft modelled in Simulink (Cofer 2012; Whalen et al. 2008). During the analysis, over 60 functional properties are verified, and 10 model errors and 2 requirement errors are found in relatively mature models.

These applications to the avionics domain demonstrate that the approach scales well. Even if the approach does not take user interfaces into account, it is a good example of formal methods applied to safety-critical systems. In addition, further investigations of the usage of compositional verification are conducted (Cofer et al. 2008; Murugesan et al. 2013), to enhance the proposed techniques.

1.6.12 Loer and Harrison et al. (Germany 2000–2006)

1.6.12.1 Modelling

Another approach to verifying interactive systems is proposed in Loer and Harrison (2002, 2006), also with the goal of making model checking more accessible to software engineers. The authors claim that in the avionics and automotive domains, requirements are often expressed as statechart models (Loer and Harrison 2002). With statecharts, a complex system can be specified as a number of potentially hierarchical state machines that describe functional or physical subsystems and run in parallel (Loer and Harrison 2000). Such parallelism could represent *concurrent behaviour*. The ordering of events which change the machine from one state to another can be used to represent *qualitative time*. Furthermore, in the statechart semantics, *time* is represented by a number of execution steps, allowing to express the formulation "within n steps from the current state…" (Loer and Harrison 2000).

To introduce formal verification in the process, they propose an automatic translation from statechart models (created with the Statemate toolkit) to the input language of the SMV model checker, which is relatively robust and well supported

(Loer and Harrison 2006). Such translation is part of the IFADIS toolbox, which also provides guided process of property specifications and a trace visualization to facilitate the result analysis of the model checker.

Concerning the modelling coverage of the approach, the authors describe five pre-defined elements in which the formal model is structured (Loer and Harrison 2000):

- **Control elements**: Description of the widgets of the UIs;
- **Control mechanism**: Description of the system functionality;
- **Displays**: Description of the output elements;
- **Environment**: Description of relevant environmental properties;
- **User tasks**: Sequence of user actions that are required to accomplish a certain task.

Therefore, their model covers the three aspects we are analysing: the user, UIs, and the functional core. However, aspects such as *multi-modality* and *multi-touch* interactions are not covered.

1.6.12.2 Verification

The properties can be verified using Cadence SMV or NuSMV model-checking tools. Depending on the type of property, the model checker can output traces that demonstrate why a property holds or not (Loer and Harrison 2006).

The property editor helps designers to construct temporal-logic properties by making patterns available and helping the process of instantiation (Loer and Harrison 2006). Temporal-logic properties can be specified either in LTL (linear temporal logic) or in CTL (computational tree logic). The following usability properties can be verified:

- **Reachability (Loer and Harrison 2000)**: Are all the states reachable or not?
- **Robustness (Loer and Harrison 2000)**: Does the system provide fallback alternatives in the case of a failure? or, alternatively, are the guards for unsafe states foolproof?
- **Recoverability (Loer and Harrison 2000)**: Does the system support undo and redo?
- **Visibility of system status (Loer and Harrison 2000)**: Does the system always keep the users informed about what is going on, through appropriate feedback within reasonable time?
- **Recognition rather than recall (Loer and Harrison 2000)**: Is the user forced to remember information from one part of the dialogue to another?
- **Behavioural consistency (Loer and Harrison 2006)**: Does the same input always yield the same effect?

In particular, the reachability property here is classified as a usability property because it is defined as generic property, which can be applied to any interactive system (i.e. "are all the states reachable or not?"). This is in contrast to the classification of the reachability property, for instance, where it is classified as a functional property because it expresses what can be done at the UI, and how can it be done, which is something that is usually defined in the system requirements.

Although the approach is not applied to many case studies (i.e. only to the avionics domain Loer and Harrison 2006), several reasons indicate that the approach scales well to real-life applications. The approach is supported by a tool that provides a translation from engineering models (statecharts) to formal models (SMV specifications), a set of property patterns to facilitate the specification of properties, and a trace visualizer to interpret the counterexamples generated by the model checker. It is used in the case study described in Loer and Harrison (2006), and an evaluation shows that the tool improves the usability of model checking for non-experts (Loer and Harrison 2006).

1.6.13 Thimbleby et al. (United Kingdom 1987–2015)

1.6.13.1 Modelling

In the healthcare domain, several investigations of medical device user interfaces have been conducted at Swansea University and Queen Mary University of London. Specifically, investigations are conducted on interactive hospital beds (Acharya et al. 2010), for user interfaces of drug infusion pumps (Cauchi et al. 2012a; Masci et al. 2014a, 2015; Thimbleby and Gow 2008), and interaction issues that can lead to serious clinical consequences.

Infusion pumps are medical devices used to deliver drugs to patients. Deep investigation has been done of the data entry systems of such devices (Cauchi et al. 2012b, 2014; Gimblett and Thimbleby 2013; Li et al. 2015; Masci et al. 2011; Oladimeji et al. 2011, 2013; Thimbleby 2010; Thimbleby and Gimblett 2011; Tu et al. 2014). If a nurse makes an error in setting up an infusion (for instance, a number ten times larger than the necessary for the patient's therapy), the patient may die. Under-dosing is also a problem: if a patient receives too little of a drug, recovery may be delayed or the patient may suffer unnecessary pain (Masci et al. 2011).

The authors report several issues with the data entry system of such pumps (Masci et al. 2014a). Several issues are detected (Masci et al. 2014a) using the approach depicted in Fig. 1.7. In this approach, the C++ source code of the infusion pump is manually translated into a specification in the PVS formal language ([a] in Fig. 1.7).

Concerning the modelling coverage, the approach deals with the display and functionality of the devices, but does not cover the modelling of the users

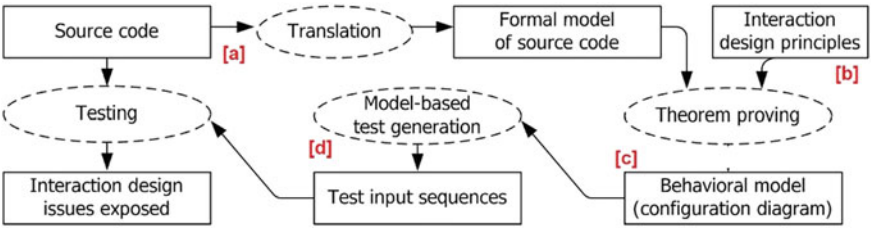


Fig. 1.7 Verification approach using PVS, adapted from Masci et al. (2014a)

interacting with such devices. In addition, no references were found describing the modelling of *concurrent behaviour, multi-modality,* nor *multi-touch interactions.*

1.6.13.2 Verification

Usability properties such as consistency of actions and feedback are formalized ([b] in Fig. 1.7) as invariants to be established using theorem proving:

- **Consistency of actions**: The same user actions should produce the same results in logically equivalent situations;
- **Feedback**: It ensures that the user is provided with sufficient information on what actions have been done and what result has been achieved.

A behavioural model is then extracted ([c] in Fig. 1.7), in a mechanized manner, from the PVS formal specification. This model captures the control structure and behaviour of the software related to the handling user interactions. Theorem proving is used to verify that the behavioural model satisfies the usability properties. Lastly, the behavioural model is exhaustively explored to generate a suite of test sequences ([d] in Fig. 1.7) (Masci et al. 2014a).

A similar approach is described in Masci et al. (2013a), in which the PVS specification is automatically discovered (Gimblett and Thimbleby 2010; Thimbleby 2007a) from reversely engineering the infusion pump software. Besides, functional properties are extracted from the safety requirements provided by the US medical device regulator FDA (Food and Drug Administration), to make sure that the medical device is reasonably safe before entering the market (Masci et al. 2013a). This approach allows *quantitative time* to be modelled and property such as "The pump shall issue an alert if paused for more than t minutes" to be verified (Masci et al. 2013a).

The same FDA safety requirements are used to verify a PVS formal model of another device, the *Generic Patient Controlled Analgesia* (GPCA) infusion pump (Masci et al. 2013a). In this work, the authors propose the usage of formal methods for rapid prototyping of user interfaces. Once verified, the formal model of the infusion pump is automatically translated into executable code through the PVS

code generator, providing a prototype of the GPCA user interface from a verified model of the infusion pump.

An approach to integrating PVS executable specifications and Stateflow models is proposed in Masci et al. (2014b), aiming at reducing the barriers that prevent non-experts from using formal methods. It permits Stateflow models to be verified, avoiding the hazards of translating design models created in different tools.

All the work mentioned in this subsection is based on the PVS theorem prover. Nevertheless, model checking can also be used in order to formally verify medical devices (Masci et al. 2011, 2015; Thimbleby 2007b). For example, the authors model the Alaris GP in Masci et al. (2015), and the B-Braun Infusomat Space infusion pumps in the higher-order logic specification language SAL (Symbolic Analysis Laboratory) (De Moura et al. 2004). Afterwards, model checking is applied to verify the predictability of user interfaces, a usability property expressed in the LTL temporal logic. Predictability is defined in Masci et al. (2011) as "if users look at the device and see that it is in a particular display state, then they can predict the next display state of the device after a user interaction".

The maturity of the approach described here and its applications to numerous case studies are evidence that the approach scales well to real-life applications.

1.6.14 Palanque et al. (France 1990–2015)

1.6.14.1 Modelling

In Palanque and Bastide (1995), another approach is proposed to modelling and verifying interactive systems with a different formalism: Petri nets (Petri 1962). Being a graphical model, Petri nets might be easier to understand than textual descriptions.

Originally, the work was targeting at modelling, implementation, and simulation of the dialogue part of event-driven interfaces (Bastide and Palanque 1990); it nowadays covers the modelling of the entire interactive system. Early notation was called Petri nets with Objects (Bastide and Palanque 1990) (which belongs to the high-level Petri nets class) and was an extension of Petri nets in order to manipulate tokens which were references to objects (in the meaning the object-oriented paradigm). This has been further extended over the years to the ICO (Interactive Cooperative Object) formalism (Navarre et al. 2009) which permits applications to be prototyped and tested but also can be fully implemented by integrating Java code in the models.

A system described using ICOs is modelled as a set of objects that cooperate to perform the system tasks. ICO uses concepts borrowed from the object-oriented formalism (such as inheritance, polymorphism, encapsulation, and dynamic instantiation) to describe the structural or static aspects of systems, such as its attributes and the operations it provides to its environment.

Through the Petri nets with Objects formalism, *data, state,* and *events*, as well as *concurrent behaviour,* can be modelled. *Dynamic reconfiguration* of user interfaces are proposed in Navarre et al. (2008), allowing operators to continue interacting with the interactive system even though part of the hardware side of the user interface is failing. Besides, the formalism also allows the modelling of low-level behaviour of interaction techniques including *multi-touch interactions* (Hamon et al. 2013). In the multi-touch context, *new fingers* are detected at execution time. Thus, the description language must be able to receive dynamically created objects. In Petri nets, this can be represented by the creation/destruction of tokens associated with the objects (Hamon et al. 2013). The notation has been used to model WIMP interfaces (Bastide and Palanque 1990) and post-WIMP ones including multi-modal interaction with speech (Bastide et al. 2004), virtual reality (Navarre et al. 2005), or bimanual interactions (Navarre et al. 2009).

Once the system is modelled using the ICO formalism, it is possible to apply model checking to verify usability and functional properties. How modelling, verification, and execution are performed using Petshop (for Petri net Workshop) (Bastide et al. 2002, 2004) (the CASE tool supporting the ICO formal description technique) is illustrated in Chap. 20 of this book.

This work allows the integration of two different representations: tasks models and interactive systems models as described in Palanque et al. (1996). This integration was first done by describing task models constructs as Petri net structures. This was first done using UAN notation (Hix and Hartson 1993) as demonstrated in Palanque et al. (1996), allowing for the verification of compatibility between the two representations using Petri net-based bisimulation (Bourguet-Rouger 1988). Then, connection with CTT notation (Paternó et al. 1997) was made using scenarios as a connection artefact (Navarre et al. 2001). Such work has been extended by developing integration of models at the syntactic level allowing for co-simulation of models (Barboni et al. 2010) and more recently using the HAMSTERS notation that allows structuring of models (Martinie et al. 2011a) and enables explicit representation of data, errors, and knowledge in task models (Fahssi et al. 2015). This work focusses on the use of multiple models jointly as presented in Martinie et al. (2014).

1.6.14.2 Verification

For instance, the following usability properties can be verified (Palanque and Bastide 1995):

- **Predictability**: The user is able to foresee the effects of a command.
- **Deadlock freedom**: The impossibility for a user to get into a state where no actions can be taken.
- **Reinitiability**: This is the ability for the user to reach the initial state of the system.

- **Exclusion of commands**: This means the commands which must never be offered at the same time (or, on the contrary, must always be offered simultaneously).
- **Succession of commands**: The proper order in which commands may be issued; for instance, a given command must or must not be followed by another one, immediately after or with some other commands in between.
- **Availability**: A command is offered all the time, regardless of the state of the system (e.g. a help command).

The specification is verified using Petri net property analysis tools (Palanque et al. 1996). In order to automate the process of property verification, the ACTL can be used to express the properties, which are then proved by model checking the Petri net marking graph (Palanque et al. 1999).

The ICO approach also permits user's cognitive behaviour to be modelled by a common Petri net for system, device, and user (Moher et al. 1996). As previously mentioned, Petshop supports the design of interactive systems according to the ICO methodology. Alternatively, the Java PathFinder model checker is used to verify a set of properties on a safety-critical application in the interactive cockpit systems modelled with ICO (Boyer and Moore 1983).

The approach has been applied to case studies and real applications in safety-critical systems not only in the space domain (see, for instance, Bastide et al. 2003, 2004; Boyer and Moore 1983; Palanque et al. 1997) but also in the air traffic management and more recently aircraft cockpits. These case studies and the maturity of the tools show that the approach scales well to real-life applications.

However, the verification based on the Petri net properties has limitations exposed in Navarre et al. (2009). The analysis is usually performed on the underlying Petri net (a simplified version of the original Petri net). A drawback is that properties verified on the underlying Petri net are not necessarily true on the original Petri net. Thus, the results of the analysis are essentially indicators of potential problems in the original Petri net. This is due to the fact that the team involved in the ICO notation has not extended work for properties verifications in Petri nets to encompass extensions (e.g. the use of preconditions).

1.6.15 Aït-Ameur et al. (France 1998–2014)

1.6.15.1 Modelling

An alternative approach is proposed in Aït-Ameur et al. (1998b, 1999, 2003a), this time relying on theorem proving using the B method (Abrial 1996) to specify the interactive system. The approach permits task models to be validated. Task models can be used to describe a system in terms of tasks, subtasks, and their temporal relationships. A task has an initial state and a final state and is decomposed in a sequence of several subtasks.

The approach uses the B method for representing, verifying, and refining specifications. The authors use the set of events to define a transition system that permits the dialogue controller of the interactive system to be represented. The CTT (Concur Task Tree) notation (Paternó et al. 1997) is used to represent task models. In Aït-Ameur et al. (2003a), only the CTT operator called "sequence" between tasks is covered. In further work (Aït-Ameur et al. 2005a, b, 2009), the authors describe how every CTT construction can be formally described in Event B (including the *concurrency* operator) allowing to translate, with generic translation rules, every CTT construction in Event B, which is the event-based definition of B method.

The case study described in Aït-Ameur et al. (1998a) shows that the approach covers the modelling of users, UIs, and the functional core. In this approach, *qualitative time* can be modelled using the PRE THEN substitution, which allows one to order operations (Aït-Ameur et al. 1998b). *Multi-touch interactions* are also covered by modelling each finger as a machine, and by integrating these machines (using the EXTENDS clause) in another machine that would represent the whole hand used for interaction (Aït-Ameur et al. 1998b). However, there is no possibility to account for quantitative time which is needed for the description of fine grain interaction in multi-modal interactions.

1.6.15.2 Verification

This usage of Event B to encode CTT task models is described in several case studies (Aït-Ameur et al. 2006; Aït-Ameur and Baron 2004, 2006; Cortier et al. 2007). In particular, the approach is used to verify Java/Swing user interfaces (Cortier et al. 2007), from which Event B models are obtained. Such Event B models encapsulate the UI behaviour of the application. Validation is achieved with respect to a task model that can be viewed as a specification. The task model is encoded in Event B, and assertions ensure that suitable interaction scenario is accepted by the CTT task model. Demonstrating that the Event B formal model behaves as intended comes to demonstrate that it is a correct refinement of the CTT task model.

Moreover, the following usability properties can be verified:

- **Robustness (Aït-Ameur et al.** 2003b): These properties are related to system dependability.
- **Visibility (Aït-Ameur et al.** 1999): It relates to feedback and information delivered to the user.
- **Reachability (Aït-Ameur et al.** 1999): These properties express what can be done at the user interface and how can it be done.
- **Reliability (Aït-Ameur et al.** 1999): It concerns the way the interface works with the underlying system.
- **Behavioural properties (Aït-Ameur and Baron** 2006): They characterize the behaviour of the UI suited by the user.

The proof of these properties is done using the invariant and assertion clauses of the B method, together with the validation of specific aspects of the task model (i.e. functional properties), thus permitting a full system task model to be validated. The Atelier B tool is used for an automatic proof obligation generation and proof obligation checking (Aït-Ameur 2000).

In order to compare this theorem proving-based approach to model-checking-based approaches, the authors show how the same case study is tackled using both theorem proving (with Event B) and model checking (with Promela/SPIN) (Aït-Ameur et al. 2003b). The authors conclude that both techniques permit the case study to be fully described and that both permit robustness and reachability properties to be verified. The proof process of the Event B-based approach is not fully automatic, but it does not suffer from the state-space explosion of model-checking techniques. The Promela-SPIN-based technique is fully automatic, but limited to finite-state systems on which exhaustive exploration can be performed. The authors conclude that a combined usage of both techniques would strengthen the verification of interactive systems.

An integration of the approach with testing is also presented in Aït-Ameur et al. (2004). Here, the informal requirements are expressed using the semi-formal notation UAN (Hix and Hartson 1993) (instead of CTT), and the B specifications are manually derived from this notation. To validate the formal specification, the authors use a data-oriented modelling language, named EXPRESS, to represent validation scenarios. The B specifications are translated into EXPRESS code (the B2EXPRESS tool Aït-Sadoune and Aït-Ameur 2008). This translation gives data models that represent specification tests and permits Event B models to be animated.

The approach is applied to several case studies in the avionics domain (Aït-Ameur et al. 2014; Jambon et al. 2001). Specifically, the authors illustrate how to explicitly introduce the context of the systems in the formal modelling (Aït-Ameur et al. 2014). The approach is also applied to the design and validation of *multimodal* interactive systems (Aït-Ameur et al. 2006, 2010a; Aït-Ameur and Kamel 2004). The numerous case studies and the maturity of the approach suggest that it might scale to real-life applications even though no evidence was given in the papers.

1.6.16 Bowen and Reeves (New Zealand 2005–2015)

1.6.16.1 Modelling

The main focus of the approach proposed by Bowen and Reeves is the use of lightweight models of interfaces and interactions in conjunction with formal models

of systems in order to bridge the gap between the typical informal UI design process and formal methods (Bowen and Reeves 2007a). UIs are formally described using a presentation model (PM) and a presentation and interaction model (PIM), while the underlying system behaviour is specified using the Z language (ISO 2002). The lightweight models allow UI and interaction designers to work with their typical artefacts (prototypes, storyboards, etc.), and a relation (PMR) is created between the behaviours described in the UI models and the formal specification in Z which gives a formal meaning to (and therefore the semantics of) the presentation models. The formal semantics also then enables a description of refinement to be given which can guide the transformation from model to implementation. Again, this is given at both formal and informal levels so can be used within both design and formal processes (Bowen and Reeves 2006).

While the primary use of the models is during the design process (assuming a specification-first approach), it is also possible to reverse-engineer existing systems into the set of models. Most examples given of this rely on a manual approach, which can be error-prone or lead to incomplete models. Some work has been done on supporting an automated approach. Using a combination of dynamic and static analysis of code and Java implementations has been investigated, and this allows UI widgets and some behaviours to be identified which can be used to partially construct models and support their completion (Bowen 2015).

The presentation model uses a simple syntax of tuples of labels, and the semantics of the model is based on set theory. It is used to formally capture the meaning of an informal design artefact such as a scenario, a storyboard, or a UI prototype by describing all possible behaviours of the windows, dialogues, or modes of the interactive system (Bowen and Reeves 2007a). In order to extend this to represent dynamic UI behaviour, a PIM (presentation and interaction model) is used. This is essentially a finite-state machine where each state represents one window, dialogue, or mode (i.e. individual presentation model) of the system with the transitions representing navigational behaviours. The PIM can be visualized using the μCharts language (Reeve and Reeves 2000) which has its own Z semantics (and which is based on statecharts) and which therefore enables a complete model of all parts of the system to be created and ultimately translated into Z.

Although the approach mainly focuses on modelling the user interfaces, the presentation model can also be used to model the user operations. The main goal is to ensure that all user operations described in the formal specification have been described in the UI design, and again, the PMR is used to support this. The models can also be used post-implementation to support testing. A set of abstract tests can be generated from the presentation models which describe the requirements of the UI widgets and behaviours, and these can then be concretized into testing frameworks such as JUnit and UISpec4 J (Bowen and Reeves 2013).

1.6.16.2 Verification

The approach has been applied to several case studies. In Bowen and Reeves (2007b), the authors use the models in the design process of UIs for the PIMed tool, a software editor for presentation models and PIMs. However, in this work, no automated verification of the presentation model and the PIMs of the editor is proposed. The formal models are manually inspected. For example, in order to verify the deadlock freedom property, the authors use a manual walk-through procedure in the PIMs. In later work (e.g. Bowen and Reeves 2012), the verification is automated by the ProZ tool, allowing both usability properties and functional properties to be verified using model checking. The kinds of usability properties that can be verified are:

- **Total reachability**: One can get to any state from any other state.
- **Deadlock freedom**: A user cannot get into a state where no action can be taken.
- **Behavioural consistency**: Controls with the same behaviour have the same name.
- **Minimum memory load on user**: Does not have to remember long sequences of actions to navigate through the UI.

Another case study to which these formal models have been applied relates to a safety-critical system in the healthcare domain (Bowen and Reeves 2012). Again, the verification is supported using ProZ. The authors model a syringe pump, a device commonly used to deliver pain-relief medication in hospitals and respite care homes. The device has ten widgets, which include the display screen, eight soft keys, and an audible alarm (*multi-modality).* Temporal safety properties and invariants (to check boundary values) are verified against the formal models using ProZ and LTL.

Typically, the generated PIMs are relatively small. This is because the number of states and transitions are bounded by the number of windows/dialogues/modes of the system rather than individual behaviours as seen in other uses of finite-state machines. This abstraction of presentation models into states of PIMs prevents state explosion and enables a combination of both manual and automatic verification (as appropriate) with reasonably low overhead. The presentation models and system specification are created manually as part of the development and specification process. While the creation of models of systems can be seen as an additional overhead to a development process, the benefits provided by both the creation and the use of the models more than compensates for this later in the process. Once the presentation models and PMR have been created, the PIM can be automatically generated and translation between the models, μCharts, and Z is automated using several tools. The individual models can be used independently or in combination

to verify different aspects of the system under consideration. Most recently, this work has focussed on safety-critical systems, and an example of this is given in Chap. 6.

1.6.17 Weyers et al. (Germany 2009–2015)

1.6.17.1 Modelling

In Weyers (2012) and Weyers et al. (2012a), a formal modelling approach has been proposed, which is based on a combination of the use of a domain-specific and visual modelling language called FILL with an accompanied transformation algorithm mapping FILL models onto a well-established formal description concept: Petri nets. In contrast to the works by Bastide and Palanque (1990) who extended basic coloured Petri nets to the ICO formalism, the modelling is done in a domain-specific description, which is not used directly for execution or verification. Instead, it is transformed into an existing formalism called reference net (Kummer 2002) (a special type of Petri net) providing a formal semantic definition for FILL, making FILL models executable using existing simulators (e.g. Renew Kummer et al. 2000) and offering the possibility for the application of verification and validation techniques for Petri net-based formalisms. As a modelling approach, FILL has been used in research on collaborative learning systems in which students created an interface for simulating a cryptographic algorithm (Weyers et al. 2009, 2010b).

As the work focuses on the model-based creation of interactive systems and less on the creation of formal models used for their verification as done in various related works, Weyers (2015) extended the basic approach with concepts from software development. The main extension, which is described in Chap. 5 of this book, addresses a component-based modelling as it offers reusability of certain components and capabilities to structure the overall model by means of functional and conceptual entities. The latter enables the modeller to create more complex (scalable) models and to be able to split the model into semantically meaningful parts. It further offers the description of complex user interfaces, which are not restricted to basic graphical user interfaces but include multi-user interfaces as well as mobile and other interaction devices. This is covered by a software infrastructure that offers capabilities to run mobile devices with models generated using FILL and its associated transferred reference nets. All is embedded into a coherent software tool called UIEditor (Weyers 2012).

An application context in which the component-based model description plays a central role is that of gaze guiding as a job aid for the control of technical processes (Kluge et al. 2014; Weyers et al. 2015). Gaze guiding as a method refers to a technique for visualizing context-dependent visual aids in the form of gaze guiding

tools into graphical user interfaces that guide the user's attention and support her or him during execution of a control task. Gaze guiding tools are visual artefacts that are added to a graphical user interface in the case the user is expected to apply a certain operation (according to a previously defined standard operating procedure), but the system does not recognize this awaited input. This work facilitates the component-based description as the model describing the behaviour of the gaze guiding is embedded as a component into the user interface description. The model specifies in which cases or context gaze guiding tools are faded into the user interface.

1.6.17.2 Model Reconfiguration and Formal Rewriting

As the work by Weyers et al. does not focus on the verification of interactive systems but on the modelling and creation of flexible and executable model-based descriptions of such systems, a formal approach for the reconfiguration and rewriting of these models has been developed. This enables the specification of formal adaptation of the models according to user input or algorithmic specifications and makes the models flexible. In this regard, the main goal is to maintain the formal integrity of a model during an adaptation. Formal integrity refers to the requirement that an adaptation approach needs to be formally well defined as well and keeps the degree of formality on the same level with that of the model description. This should prevent any gaps in the formalization during adaptation of a model and thus prevent the compromise of any following verification, testing, or debugging of the rewritten model. Therefore, Weyers et al. (2010a, 2014) developed a reconfiguration concept based on pushouts, a concept known from category theory for the rewriting of reference nets. Together with Stückrath, Weyers extended a basic approach for the rewriting of Petri nets based on the so-called double-pushout approach to a method for coloured Petri nets equipped with a rewriting of XML-based specification of inscriptions (Stückrath and Weyers 2014).

The application of formal rewriting that is driven by the user has been investigated in the context of the monitoring and control of complex technical and safety-critical systems (Burkolter et al. 2014; Weyers et al. 2012a, b). In these works, the reconfiguration of a given user interface for controlling a simplified nuclear power plant was reconfigured by the user according to his or her own needs as well as to the standard operating procedures which were presented. These adaptations of the user interface include a change not only in the visual layout of widgets but also in the functional behaviour of the interface using the rewriting of the underlying reference net model. Operations were offered to the user, e.g., to generate a new widget which triggers a combination of two existing operations. For example, it was possible to combine the opening and closing of different valves into one new operation, which was accessible through a new single widget, e.g. a button. By pressing this button, the user was able to simultaneously open and close

the dedicated valves with one click instead of two. Weyers et al. were able to show that this individualization of a user interface reduces errors in the accompanied control task. A detailed introduction into the rewriting concept is presented in Chap. 10 of this book.

1.6.18 Combéfis et al. (Belgium 2009–2013)

1.6.18.1 Modelling

In Combéfis (2013), a formal framework for reasoning over system and user models is proposed, and the user models can be extracted from user manuals. Furthermore, this work proposes the automatic generation of user models. Using his technique, "adequate" user models can be generated from a given initial user model. Adequate user models capture the knowledge that the user must have about the system, i.e. the knowledge needed to control the system, using all its functionality and avoiding surprises. This generated user model can be used, for instance, to improve training manuals and courses (Combefis and Pecheur 2009).

In order to compare the system and the user model and to verify whether the user model is adequate to the system model, both models should be provided. With this goal, in this approach system and user are modelled with enriched labelled transition systems called HMI LTS (Combéfis et al. 2011a). In HMI LTS, three kinds of actions are defined (Combéfis et al. 2011a):

- **Commands**: Actions triggered by the user on the system;
- **Observations**: Actions triggered by the system, but that the user can observe; and
- **Internal actions**: Actions that are neither controlled nor observed by the user.

To be considered "adequate", user models are expected to follow two specific properties: full control and mode-preserving. Intuitively, a user model allows full control of a system if at any time, when using the system according to the user model (Combefis and Pecheur 2009): the commands that the user model allows are exactly those available on the system; and the user model allows at least all the observations that can be produced by the system (Combefis and Pecheur 2009). A user model is said to be mode-preserving according to a system, if and only if, for all possible executions of the system the users can perform with their user model, given the observation they make, the mode predicted by the user model is the same as the mode of the system (Combefis and Pecheur 2009). Model checking is used to verify both properties over the user model.

Concerning the approach's coverage regarding modelling, the users, the user interfaces, and the functional core are modelled and compared to each other, the user model being extracted from the user manual describing the system. However, there is no indication that the approach supports *concurrent behaviour, multi-modality,* or *multi-touch* interactions.

1.6.18.2 Verification

The verification goal of this approach is to verify whether the user model is adequate to the system model, rather than to verify properties over the system model. In order to automatically generate adequate user models, the authors propose a technique based on a derivative of weak bisimulation, in which equivalence checking is used (Milner 1980). This is called "minimization of a model modulo an equivalence relation". Intuitively, using equivalence checking, they generate a user model U_2 from the initial user model U_1; that is, U_2 is equivalent to U_1 with respect to specific equivalence relations introduced by the authors.

Two equivalence relations are proposed: full-control equivalence and mode-preserving equivalence. Full-control equivalence distinguishes commands and observations: two equivalent states must allow the same set of commands, but may permit different sets of observations. Minimization modulo this equivalence produces a minimal user model that permits full control of the system. A mode-preserving equivalence is then derived from the full-control equivalence, by adding an additional constraint that the modes of two equivalent states must be the same (Combefis and Pecheur 2009). Using these equivalence relations, the authors can generate mode-preserving fully controlled user models, which can then be used to design user interfaces and/or training manuals. Both properties (i.e. mode-preserving and full control) and their combination are interesting because they propose that different levels of equivalence can be shown between system models.

A tool named jpf-hmi has been implemented in Java and uses the Java Pathfinder model checker (Combéfis et al. 2011a), to analyse and generate user models. The tool produces an LTS corresponding to one minimal fully controlled mental model, or it reports that no such model exists by providing a problematic sequence from the system (Combéfis et al. 2011a).

The approach is applied to several examples that are relatively large (Combéfis et al. 2011b). In the healthcare domain, a machine that treats patients by administering X-ray or electron beams is analysed with the approach, which detects several issues in the system. In the avionics domain, the approach is applied to an autopilot system of a Boeing airplane (Combéfis 2013), and a potential-mode confusion is identified. These are evidence that the approach scales well to real-life applications.

1.6.19 Synthesis

This section presents a representative list of approaches to verifying interactive systems with respect to the specifications, i.e. general statements about the behaviour of the system, which are represented here as desired properties, and analysed afterwards using formal methods. The approaches diverge on the formalisms they use for the description of interactive systems and for the specification of properties.

Some authors use theorem proving to perform verification, which is a technique that can handle infinite-state systems. Even though a proof done by a theorem prover is ensured to be correct, it can quickly become a hard process (Bumbulis et al. 1995b): the process is not fully automated, and user guidance is needed regarding the proof strategy to follow. Simulation can also be used to assess the quality of interactive systems. Simulation provides an environment for training the staff before starting their daily activities (Martinie et al. 2011b). However, simulated environments are limited in terms of training, since it is impossible to drive operators into severe and stressful conditions even using a full-scale simulator (Niwa et al. 2001). Simulation explores a part of the system state space and can be used for disproving certain properties by showing examples of incorrect behaviours. To the contrary, formal techniques such as model checking and equivalence checking consider the entire state space and can thus prove or disprove properties for all possible behaviours (Garavel and Graf 2013).

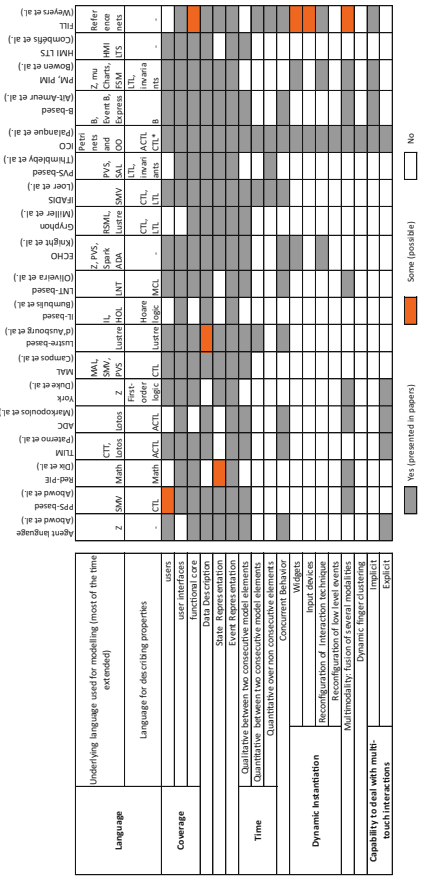
The presented approaches allow either *usability* or *dependability* properties to be verified over the system models. We believe that in the case of safety-critical systems, the verification approach should cover both such properties, due to the ergonomic aspects covered by the former and the safety aspects covered by the latter. Some approaches cover the modelling of the users, the user interfaces, and the functional core.

1.6.20 Summary

A representative list of approaches for describing and assessing the quality of interactive systems has been presented in this chapter.

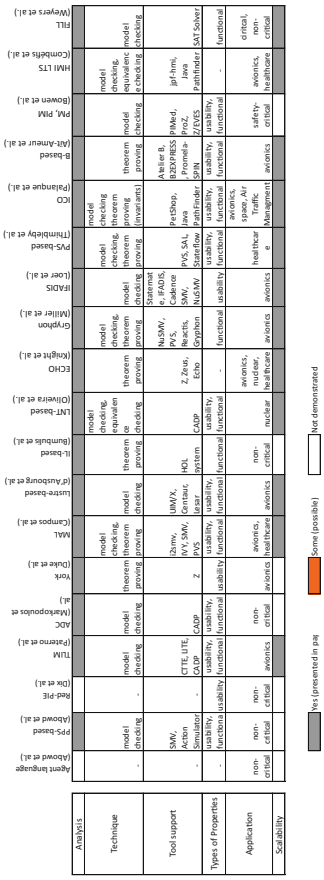
Different formalisms are used in the system modelling (and property modelling, when applied). Numerous case studies have shown that each formalism has its strengths. The criteria to choose one over another would be more related with the knowledge and experience of the designers in the formalisms. Different formal techniques are employed, such as model checking, equivalence checking, and theorem proving. Most of the works presented here are tool supported, even though some authors still use manual inspection of the models to perform verification.

Table 1.1 Modelling coverage of the approaches



Tables 1.1 and 1.2 below summarize these approaches: the former gives an overview of the modelling coverage of the approaches, and the latter an overview of their verification capabilities. It is important to note that those tables are only meant at summarizing what has been presented. The cells of Table 1.1 are coloured according to the information found in papers. An empty cell does not mean that this aspect cannot be addressed by the notation; it only means that no paper has made that information explicit. Indeed, the presentation in the sections usually contains more details than the summary table.

Table 1.2 Verification capabilities of the approaches



References

Abowd GD (1991) Formal aspects of human-computer interaction. Dissertation, University of Oxford

Abowd GD, Dix AJ (1992) Giving undo attention. Interact Comput 4(3):317–342

Abowd GD, Wang H-M, Monk AF (1995) A formal technique for automated dialogue development. In: Proceedings of the 1st conference on designing interactive systems: processes, practices, methods, & techniques. ACM, pp 219–226

Abrial J-R (1996) The B-book: Assigning Programs to Meanings. Cambridge University Press, New York

Acharya C, Thimbleby HW, Oladimeji P (2010) Human computer interaction and medical devices. In: Proceedings of the 2010 British computer society conference on human-computer interaction, pp 168–176

Aït-Ameur Y, Girard P, Jambon F (1998a) A uniform approach for specification and design of interactive systems: the B method. In: Proceedings of the fifth international eurographics workshop on design, specification and verification of interactive systems, pp 51–67

Aït-Ameur Y, Girard P, Jambon F (1998b) 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, pp 333–352

Aït-Ameur Y, Girard P, Jambon F (1999) Using the B formal approach for incremental specification design of interactive systems. In: Engineering for human-computer interaction. Springer, pp 91–109

Aït-Ameur Y (2000) Cooperation of formal methods in an engineering based software development process. In: Proceedings of integrated formal methods, second international conference, pp 136–155

Aït-Ameur Y, Baron M, Girard P (2003a) Formal validation of HCI user tasks. In: Software engineering research and practice, pp 732–738

Aït-Ameur Y, Baron M, Kamel N (2003b) Utilisation de Techniques Formelles dans la Modelisation d'Interfaces Homme-machine. Une Experience Comparative entre B et Promela/SPIN. In: 6th International Symposium on Programming and Systems, pp 57–66

Aït-Ameur Y, Kamel N (2004) A generic formal specification of fusion of modalities in a multimodal HCI. In: Building the information society. Springer, pp 415–420

Aït-Ameur Y, Baron M (2004) Bridging the gap between formal and experimental validation approaches in HCI systems design: use of the event B proof based technique. In: International symposium on leveraging applications of formal methods, pp 74–80

Aït-Ameur Y, Breholee B, Girard P, Guittet L, Jambon F (2004) Formal verification and validation of interactive systems specifications. In: Human error, safety and systems development. Springer, pp 61–76

Aït-Ameur Y, Baron M, Kamel N (2005a) Encoding a process algebra using the event B method. Application to the validation of user interfaces. In: Proceedings of 2nd IEEE international symposium on leveraging applications of formal methods, pp 1–17

Aït-Ameur Y, Idir A-S, Mickael B (2005b) Modelisation et Validation formelles d'IHM: LOT 1 (LISI/ENSMA). Technical report. LISI/ENSMA

Aït-Ameur Y, Aït-Sadoune I, Mota J-M, Baron M (2006) Validation et Verification Formelles de Systemes Interactifs Multi-modaux Fondees sur la Preuve. In: Proceedings of the 18th International Conference of the Association Francophone d'Interaction Homme-Machine, pp 123–130

Aït-Ameur Y, Baron M (2006) Formal and experimental validation approaches in HCI systems design based on a shared event B model. Int J Softw Tools Technol Transf 8(6):547–563

Aït-Sadoune I, Aït-Ameur Y (2008) Animating event B models by formal data models. In: Proceedings of leveraging applications of formal methods, verification and validation, pp 37–55

Aït-Ameur Y, Baron M, Kamel N, Mota J-M (2009) Encoding a process algebra using the event B method. Int J Softw Tools Technol Transfer 11(3):239–253

Aït-Ameur Y, Boniol F, Wiels V (2010a) Toward a wider use of formal methods for aerospace systems design and verification. Int J Softw Tools Technol Transf 12(1):1–7

Aït-Ameur Y, Aït-Sadoune I, Baron M, Mota J-M (2010b) Verification et Validation Formelles de Systemes Interactifs Fondees sur la Preuve: Application aux Systemes Multi-Modaux. J Interact Pers Syst 1(1):1–30

Aït-Ameur Y, Gibson JP, Mery D (2014) On implicit and explicit semantics: integration issues in proof-based development of systems. In: Leveraging applications of formal methods, verification and validation. Specialized techniques and applications. Springer, pp 604–618

Barboni E, Ladry J-F, Navarre D, Palanque PA, Winckler M (2010) Beyond modelling: an integrated environment supporting co-execution of tasks and systems models. In: Proceedings of ACM EICS conference, pp 165–174

Basnyat S, Palanque PA, Schupp B, Wright P (2007) Formal socio-technical barrier modelling for safety-critical interactive systems design. Saf Sci 45(5):545–565

Bass L, Little R, Pellegrino R, Reed S, Seacord R, Sheppard S, Szezur MR (1991) The ARCH model: seeheim revisited. In: User interface developpers' workshop
Bastide R, Palanque PA (1990) Petri net objects for the design, validation and prototyping of user-driven interfaces. In: IFIP INTERACT 1990 conference, pp 625–631
Bastide R, Navarre D, Palanque PA (2002) A model-based tool for interactive prototyping of highly interactive applications. In: CHI extended abstracts, pp 516–517
Bastide R, Navarre D, Palanque PA (2003) A tool-supported design framework for safety critical interactive systems. Interact Comput 15(3):309–328
Bastide R, Navarre D, Palanque PA, Schyn A, Dragicevic P (2004) A model-based approach for real-time embedded multimodal systems in military aircrafts. In: Proceedings of the 6th international conference on multimodal interfaces, pp 243–250
Beck K (1999) Extreme programming explained: embrace change. Addison-Wesley
Booch G (2005) The unified modelling language user guide. Pearson Education, India
Bourguet-Rouger A (1988) External behaviour equivalence between two petri nets. In: Proceedings of concurrency. Lecture notes in computer science, vol 335. Springer, pp 237–256
Bowen J, Reeves S (2006) Formal refinement of informal GUI design artefacts. In: Proceedings of 17th Australian software engineering conference, pp 221–230
Bowen J, Reeves S (2007a) Formal models for informal GUI designs. Electr Notes Theor Comput Sci 183:57–72
Bowen J, Reeves S (2007b) using formal models to design user interfaces: a case study. In: Proceedings of the 21st British HCI group annual conference on HCI, pp 159–166
Bowen J, Reeves S (2012) Modelling user manuals of modal medical devices and learning from the experience. In: Proceedings of ACM SIGCHI symposium on engineering interactive computing systems, pp 121–130
Bowen J, Reeves S (2013) UI-design driven model-based testing. Innov Syst Softw Eng 9(3):201–215
Bowen J (2015) Creating models of interactive systems with the support of lightweight reverse-engineering tools. In: Proceedings of the 7th ACM SIGCHI symposium on engineering interactive computing systems, pp 110–119
Boyer RS, Moore JS (1983) Proof-checking, theorem proving, and program verification. Technical report, DTIC document
Brat G, Martinie C, Palanque P (2013) V&V of lexical, syntactic and semantic properties for interactive systems through model checking of formal description of dialog. In: Proceedings of the 15th international conference on human-computer interaction, pp 290–299
Bumbulis P, Alencar PSC, Cowan DD, Lucena CJP (1995a) Combining formal techniques and prototyping in user interface construction and verification. In: Proceedings of 2nd eurographics workshop on design, specification, verification of interactive systems, pp 7–9
Bumbulis P, Alencar PSC, Cowan DD, de Lucena CJP (1995b) A framework for machine-assisted user interface verification. In: Proceedings of the 4th international conference on algebraic methodology and software technology, pp 461–474
Burkolter D, Weyers B, Kluge A, Luther W (2014) Customization of user interfaces to reduce errors and enhance user acceptance. Appl Ergon 45(2):346–353
Campos JC, Harrison MD (1997) Formally verifying interactive systems: a review. In: Proceedings of design, specification and verification of interactive systems, pp 109–124
Campos JC (1999) Automated deduction and usability reasoning. Dissertatoin, University of York
Campos JC, Harrison MD (2001) Model checking interactor specifications. Autom Softw Eng 8 (3–4):275–310
Campos JC, Harrison MD (2007) Considering context and users in interactive systems analysis. In: Proceedings of the joint working conferences on engineering interactive systems, pp 193–209
Campos JC, Harrison MD (2008) Systematic analysis of control panel interfaces using formal tools. In: Interactive systems. Design, specification, and verification. Springer, pp 72–85
Campos JC, Harrison MD (2009) Interaction engineering using the IVY tool. In: Proceedings of the 1st ACM SIGCHI symposium on engineering interactive computing systems, pp 35–44

Campos JC, Harrison MD (2011) Modelling and analysing the interactive behaviour of an infusion pump. Electron Commun EASST 45

Cauchi A, Gimblett A, Thimbleby HW, Curzon P, Masci P (2012a) Safer "5-key" number entry user interfaces using differential formal analysis. In: Proceedings of the 26th annual BCS interaction specialist group conference on people and computers, pp 29–38

Cauchi A, Gimblett A, Thimbleby HW, Curzon P, Masci P (2012b) Safer "5-key" number entry user interfaces using differential formal analysis. In: Proceedings of the 26th annual BCS interaction specialist group conference on people and computers, pp 29–38

Cauchi A, Oladimeji P, Niezen G, Thimbleby HW (2014) Triangulating empirical and analytic techniques for improving number entry user interfaces. In: Proceedings of ACM SIGCHI symposium on engineering interactive computing systems, pp 243–252

Champelovier D, Clerc X, Garavel H, Guerte Y, Lang F, Serwe W, Smeding G (2010) Reference manual of the LOTOS NT to LOTOS translator (version 5.0). INRIA/VASY

Chen P (1976) The entity-relationship model—toward a unified view of data. ACM Trans Database Syst 1(1):9–36

Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8(2):244–263

Cofer D, Whalen M, Miller S (2008) Software model checking for avionics systems. In: Digital avionics systems conference, pp 1–8

Cofer D (2010) Model checking: cleared for take-off. In: Model checking software. Springer, pp 76–87

Cofer D (2012) Formal methods in the aerospace industry: follow the money. In: Proceedings of the 14th international conference on formal engineering methods: formal methods and software engineering. Springer, pp 2–3

Cofer D, Gacek A, Miller S, Whalen MW, LaValley B, Sha L (2012) Compositional verification of architectural models. In: Proceedings of the 4th international conference on NASA formal methods. Springer, pp 126–140

Combefis S, Pecheur C (2009) A bisimulation-based approach to the analysis of human-computer interaction. In: Proceedings of the 1st ACM SIGCHI symposium on engineering interactive computing systems, pp 101–110

Combéfis S, Giannakopoulou D, Pecheur C, Feary M (2011a) A formal framework for design and analysis of human-machine interaction. In: Proceedings of the IEEE international conference on systems, man and cybernetics, pp 1801–1808

Combéfis S, Giannakopoulou D, Pecheur C, Feary M (2011b) Learning system abstractions for human operators. In: Proceedings of the international workshop on machine learning technologies in software engineering, pp 3–10

Combéfis S (2013) A formal framework for the analysis of human-machine interactions. Dissertation, Universite catholique de Louvain

Cortier A, d'Ausbourg B, Aït-Ameur Y (2007) Formal validation of java/swing user interfaces with the event B method. In: Human-computer interaction. Interaction design and usability. Springer, pp 1062–1071

Coutaz J (1987) PAC, an object oriented model for dialogue design. In: Bullinger H-J, Shackel B (eds) Human computer interaction INTERACT'87, pp 431–436

Curzon P, Blandford A (2004) Formally justifying user-centred design rules: a case study on post-completion errors. Proc IFM 2004:461–480

d'Ausbourg B (1998) Using model checking for the automatic validation of user interface systems. In: Proceedings of the fifth international eurographics workshop on the design, specification and verification of interactive systems, pp 242–260

d'Ausbourg B, Seguin C, Durrieu G, Roche P (1998) Helping the automated validation process of user interfaces systems. In: Proceedings of the 20th international conference on software engineering, pp 219–228

d'Ausbourg B (2002) Synthetiser I'Intention d'un Pilote pour Definir de Nouveaux Équipements de Bord. In: Proceedings of the 14th French-speaking conference on human-computer Interaction, pp 145–152

De Moura L, Owre S, Rueß H, Rushby J, Shankar N, Sorea M, Tiwari A (2004) SAL 2. In: Proceeidngs of computer aided verification, pp 496–500
Degani A, Heymann M (2002) Formal verification of human-automation interaction. Hum Fact J Hum Fact Ergon Soc 44(1):28–43
Dey T (2011) A comparative analysis on modelling and implementing with MVC architecture. In: Proceedings of the international conference on web services computing, vol 1, pp 44–49
Dix AJ (1988) Abstract, generic models of interactive systems. In: Proceedings of fourth conference of the British computer society human-computer interaction specialist group. University of Manchester, pp 63–77
Dix AJ, Harrison MD, Cunciman R, Thimbleby HW (1987) Interaction models and the principled design of interactive systems. In: Proceedings of the 1st European software engineering conference, pp 118–126
Dix AJ (1991) Formal methods for interactive systems. Academic Press, London
Dix AJ (1995) Formal Methods. In: Monk A, Gilbert N (eds) Perspectives on HCI: diverse approaches. Academic Press, London, pp 9–43
Dix AJ (2012) Formal methods. In: Soegaard M, Dam R (eds) Encyclopedia of human-computer interaction
Doherty G, Campos JC, Harrison MD (1998) Representational reasoning and verification. Formal Aspects Comput 12:260–277
Duke DJ, Harrison MD (1993) Abstract interaction objects. Comput Graph Forum 12:25–36
Duke DJ, Harrison MD (1995) Event model of human-system interaction. Softw Eng J 10(1):3–12
Elder MC, Knight J (1995) Specifying user interfaces for safety-critical medical systems. In: Proceedings of the 2nd annual international symposium on medical robotics and computer assisted surgery, pp 148–155
Fahssi R, Martinie C, Palanque PA (2015) Enhanced task modelling for systematic identification and explicit representation of human errors. In: Proceedings of INTERACT, pp 192–212
Fields B, Wright P, Harrison M (1995) Applying formal methods for human error tolerant design. In: Software engineering and human-computer interaction. Springer, pp 185–195
Foley JD, Wallace VL (1974) The art of natural graphic man-machine conversation. Comput Graph 8(3):87
Garavel H, Graf S (2013) formal methods for safe and secure computer systems. Federal office for information security
Garavel H, Lang F, Mateescu R, Serwe W (2013) CADP 2011: a toolbox for the construction and analysis of distributed processes. Int J Softw Tools Technol Transf 15(2):89–107
Gimblett A, Thimbleby HW (2010) User interface model discovery: towards a generic approach. In: Proceedings of the 2nd ACM SIGCHI symposium on engineering interactive computing system, EICS 2010, Berlin, Germany, pp 145–154
Gimblett A, Thimbleby HW (2013) Applying theorem discovery to automatically find and check usability heuristics. In: Proceedings of the 5th ACM SIGCHI symposium on engineering interactive computing systems, pp 101–106
Hallinger P, Crandall DP, Seong DNF (2000) Systems thinking/systems changing & a computer simulation for learning how to make school smarter. Adv Res Theor School Manag Educ Policy 1(4):15–24
Hamilton D, Covington R, Kelly J, Kirkwood C, Thomas M, Flora-Holmquist AR, Staskauskas MG, Miller SP, Srivas MK, Cleland G, MacKenzie D (1995) Experiences in applying formal methods to the analysis of software and system requirements. In: Workshop on industrial-strength formal specification techniques, pp 30–43
Hamon A, Palanque PA, Silva JL, Deleris Y, Barboni E (2013) Formal description of multi-touch interactions. In: Proceedings of the 5th ACM SIGCHI symposium on engineering interactive computing systems, pp 207–216
Hardin DS, Hiratzka TD, Johnson DR, Wagner L, Whalen MW (2009) Development of security software: a high assurance methodology. In: Proceedings of 11th international conference on formal engineering methods, pp 266–285
Harrison M, Thimbleby HW (eds) (1990) Formal methods in HCI. Cambridge University Press

Harrison MD, Duke DJ (1995) A review of formalisms for describing interactive behaviour. In: software engineering and human-computer interaction. Springer, pp 49–75
Harrison MD, Masci P, Campos JC, Curzon P (2013) Automated theorem proving for the systematic analysis of an infusion pump. Electron Commun EASST69
Harrison MD, Campos JC, Masci P (2015) Reusing models and properties in the analysis of similar interactive devices, pp 95–111
Hix D, Hartson RH (1993) Developing user interfaces: ensuring usability through product process. Wiley, New York
ISO/IEC (1989) LOTOS—a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807
IEC ISO (2001) Enhancements to LOTOS (E-LOTOS). International Standard 15437
ISO/IEC (2002) 13568, Information technology—Z formal specification notation—syntax, type system and semantics
Jambon F, Girard P, Aït-Ameur Y (2001) Interactive system safety and usability enforced with the development process. In: Proceedings of 8th IFIP international conference on engineering for human-computer interaction, pp 39–56
Kieras DE, Polson PG (1985) An approach to the formal analysis of user complexity. Int J Man Mach Stud 22:94–365
Kluge A, Greve J, Borisov N, Weyers B (2014) Exploring the usefulness of two variants of gaze-guiding-based dynamic job aid for performing a fixed sequence start up procedure after longer periods of non-use. Hum Fact Ergon 3(2):148–169
Knight JC, Kienzle DM (1992) Preliminary experience using Z to specify a safety-critical system. In: Proceedings of Z user workshop, pp 109–118
Knight JC, Brilliant SS (1997) Preliminary evaluation of a formal approach to user interface specification. In: The Z formal specification notation. Springer, pp 329–346
Knight JC, Fletcher PT, Hicks BR (1999) Tool support for production use of formal techniques. In: Proceedings of the world congress on formal methods in the development of computing systems, pp 242–251
Kummer O, Wienberg F, Duvigneau M, Köhler M, Moldt D, Rölke H (2000) Renew—the reference net workshop. In: Proceedings of 21st international conference on application and theory of Petri nets-tool demonstrations, pp 87–89
Kummer O (2002) Referenznetze. Universität Hamburg, Disseration
Li K-Y, Oladimeji P, Thimbleby HW (2015) Exploring the effect of pre-operational priming intervention on number entry errors. In: Proceedings of the 33rd annual ACM conference on human factors in computing systems, pp 1335–1344
Loer K, Harrison MD (2000) Formal interactive systems analysis and usability inspection methods: two incompatible worlds? In: Interactive systems: design, specification, and verification, pp 169–190
Loer K, Harrison MD (2002) Towards usable and relevant model checking techniques for the analysis of dependable interactive systems. In: Proceedings of automated software engineering, pp 223–226
Loer K, Harrison MD (2006) An integrated framework for the analysis of dependable interactive systems (IFADIS): its tool support and evaluation. Autom Softw Eng 13(4):469–496
Lutz RR (2000) Software engineering for safety: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 213–226
Mancini R (1997) Modelling interactive computing by exploiting the undo. Dissertation, University of Rome
Markopoulos P (1995) On the expression of interaction properties within an interactor model. In: Interactive systems: design, specification, and verification, pp 294–310
Markopoulos P, Rowson J, Johnson P (1996) Dialogue modelling in the framework of an interactor model. In: Pre-conference proceedings of design specification and verification of interactive systems, vol 44, Namur, Belgium
Markopoulos P (1997) A compositional model for the formal specification of user interface software. Dissertation, University of London

Markopoulos P, Johnson P, Rowson J (1998) Formal architectural abstractions for interactive software. Int J Hum Comput Stud 49(5):675–715

Martinie C, Palanque PA, Navarre D, Winckler M, Poupart E (2011) Model-based training: an approach supporting operability of critical interactive systems. In: Proceedings of ACM EICS conference, pp 53–62

Martinie C, Palanque PA, Winckler M (2011) Structuring and composition mechanisms to address scalability issues in task models. In: Proceedings of IFIP TC 13 INTERACT, pp 589–609

Martinie C, Navarre D, Palanque PA (2014) A multi-formalism approach for model-based dynamic distribution of user interfaces of critical interactive systems. Int J Hum Comput Stud 72(1):77–99

Masci P, Ruksenas R, Oladimeji P, Cauchi A, Gimblett A, Li Y, Curzon P, Thimbleby H (2011) On formalising interactive number entry on infusion pumps. Electron Commun EASST 45

Masci P, Ayoub A, Curzon P, Harrison MD, Lee I, Thimbleby H (2013a) Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In: Proceedings of the 5th ACM SIGCHI symposium on engineering interactive computing systems, pp 81–90

Masci P, Zhang Y, Jones PL, Oladimeji P, D'Urso E, Bernardeschi C, Curzon P, Thimbleby H (2014a) Formal verification of medical device user interfaces using PVS. In: Proceedings of the 17th international conference on fundamental approaches to software engineering. Springer, pp 200–214

Masci P, Zhang Y, Jones PL, Oladimeji P, D'Urso E, Bernardeschi C, Curzon P, Thimbleby H (2014b) Combining PVSio with stateflow. In: Proceedings of NASA formal methods—6th international symposium, pp 209–214

Masci P, Ruksenas R, Oladimeji P, Cauchi A, Gimblett A, Li AY, Curzon P, Thimbleby HW (2015) The benefits of formalising design guidelines: a case study on the predictability of drug infusion pumps. Innov Syst Softw Eng 11(2):73–93

Mateescu R, Thivolle D (2008) A model checking language for concurrent value-passing systems. In: Cuellar J, Maibaum T, Sere K (eds) Proceedings of the 15th international symposium on formal methods. Springer, pp 148–164

Merriam NA, Harrison MD (1996) Evaluating the interfaces of three theorem proving assistants. In: Proceedings of DSV-IS conference. Springer, pp 330–346

Miller SP, Tribble AC, Whalen MW, Heimdahl MPE (2006) Proving the shalls. Int J Softw Tools Technol Transf 8(4–5):303–319

Miller SP (2009) Bridging the gap between model-based development and model checking. In: Tools and algorithms for the construction and analysis of systems. Springer, pp 443–453

Miller SP, Whalen MW, Cofer DD (2010) Software model checking takes off. Commun ACM 53 (2):58–64

Milner R (1980) A calculus of communicating systems. Springer

Moher T, Dirda V, Bastide R (1996) A bridging framework for the modelling of devices, users, and interfaces. Technical report

Murugesan A, Whalen MW, Rayadurgam S, Heimdahl MPE (2013) Compositional verification of a medical device system. In: Proceedings of the 2013 ACM SIGAda annual conference on high integrity language technology, pp 51–64

Navarre D, Palanque PA, Paterno F, Santoro C, Bastide R (2001) A tool suite for integrating task and system models through scenarios. In: Proceedings of the 8th international workshop on interactive systems: design, specification, and verification-revised papers. Springer, pp 88–113

Navarre D, Palanque PA, Bastide R, Schyn A, Winckler M, Nedel LP, Freitas CMDS (2005) A formal description of multimodal interaction techniques for immersive virtual reality applications. In: Proceedings of the 2005 IFIP TC13 international conference on human-computer interaction. Springer, pp 170–183

Navarre D, Palanque PA, Basnyat S (2008) A formal approach for user interaction reconfiguration of safety critical interactive systems. In: Computer safety, reliability, and security. Springer, pp 373–386

Navarre D, Palanque PA, Ladry J-F, Barboni E (2009) ICOs: a model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Trans Comput Hum Interact 16(4):1–56

Niwa Y, Takahashi M, Kitamura M (2001) The design of human-machine interface for accident support in nuclear power plants. Cogn Technol Work 3(3):161–176

Oladimeji P, Thimbleby HW, Cox AL (2011) Number entry interfaces and their effects on error detection. In: Proceedings of human-computer interaction—INTERACT 2011—13th IFIP TC 13 international conference, pp 178–185

Oladimeji P, Thimbleby HW, Cox AL (2013) A performance review of number entry interfaces. In: Proceedings of human-computer interaction—INTERACT 2013—14th IFIP TC 13 international conference, pp 365–382

Oliveira R, Dupuy-Chessa S, Calvary G (2014) Formal verification of UI using the power of a recent tool suite. In: Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems, pp 235–240

Oliveira R (2015) Formal specification and verification of interactive systems with plasticity: applications to nuclear-plant supervision. Dissertation, Université Grenoble Alpes

Oliveira R, Dupuy-Chessa S, Calvary G (2015a) Verification of plastic interactive systems. i-com 14(3):192–204

Oliveira R, Dupuy-Chessa S, Calvary G (2015b) Equivalence checking for comparing user interfaces. In: Proceedings of the 7th ACM SIGCHI symposium on engineering interactive computing systems, pp 266–275

Oliveira R, Dupuy-Chessa S, Calvary G (2015c) Plasticity of user interfaces: formal verification of consistency. In: Proceedings of the 7th ACM SIGCHI symposium on engineering interactive computing systems, pp 260–265

OMG (2010) Systems modelling language (OMG SysML™), version 1.2

Palanque PA, Bastide R, Sengès V (1995) Validating interactive system design through the verification of formal task and system models. In: Proceedings of IFIP WG 2.7 conference on engineering human computer interaction, pp 189–212

Palanque PA, Bastide R (1995) Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In: Interactive systems: design, specification, and verification. Springer, pp 383–400

Palanque PA, Bastide R, Senges V (1996) Validating interactive system design through the verification of formal task and system models. In: Proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction, pp 189–212

Palanque PA, Paternó F (eds) (1997) Formal methods in HCI. Springer

Palanque PA, Bastide R, Paternó F (1997) Formal specification as a tool for objective assessment of safety-critical interactive systems. In: Proceedings of the IFIP TC13 international conference on human-computer interaction, pp 323–330

Palanque PA, Farenc C, Bastide R (1999) Embedding ergonomic rules as generic requirements in a formal development process of interactive software. In: Human-computer interaction INTERACT'99: IFIP TC13 international conference on human-computer interaction, pp 408–416

Palanque PA, Winckler M, Ladry J-F, ter Beek M, Faconti G, Massink M (2009) A formal approach supporting the comparative predictive assessment of the interruption-tolerance of interactive systems. In: Proceedings of ACM EICS 2009 conference, pp 46–55

Payne SJ, Green TRG (1986) Task-action grammars: a model of mental representation of task languages. Hum Comput Interact 2(2):93–133

Park D (1981) Concurrency and automata on infinite sequences. In: Proceedings of the 5th GI-conference on theoretical computer science. Springer, pp 167–183

Parnas DL (1969) On the use of transition diagrams in the design of a user interface for an interactive computer system. In: Proceedings of the 24th national ACM conference, pp 379–385

Paternó F, Faconti G (1992) On the use of LOTOS to describe graphical interaction. People and computers VII. Cambridge University Press, pp 155–155

Paternó F (1994) A Theory of user-interaction objects. J Vis Lang Comput 5(3):227–249

Paternó F, Mezzanotte M (1994) Analysing MATIS by interactors and ACTL. Technical report

Paternó F, Mezzanotte M (1996) Formal verification of undesired behaviours in the CERD case study. In: Proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction, pp 213–226

Paternó F (1997) Formal reasoning about dialogue properties with automatic support. Interact Comput 9(2):173–196

Paternó F, Mancini C, Meniconi S (1997) ConcurTaskTrees: a diagrammatic notation for specifying task models. In: Proceedings of the IFIP TC13 international conference on human-computer interaction, pp 362–369

Paternó F, Santoro C (2001). Integrating model checking and HCI tools to help designers verify user interface properties. In: Proceedings of the 7th international conference on design, specification, and verification of interactive systems. Springer, pp 135–150

Paternó F, Santoro C (2003) Support for reasoning about interactive systems through human-computer interaction designers' representations. Comput J 46(4):340–357

Petri CA (1962) Kommunikation mit Automaten. Dissertation, University of Bonn

Pfaff G, Hagen P (eds) (1985) Seeheim workshop on user interface management systems. Springer, Berlin

Reeve G, Reeves S (2000) μCharts and Z: Hows, whys, and wherefores. In: Integrated formal methods: second international conference. Springer, Berlin

Reisner P (1981) Formal grammar and human factors design of an interactive graphics system. IEEE Trans Softw Eng 7(2):229–240

Schwaber K (2004) Agile project management with scrum. Microsoft Press

Sifakis J (1979) Use of petri nets for performance evaluation. In: Beilner H, Gelenbe E (eds), Proceedings of 3rd international symposium on modelling and performance of computer systems

Shepherd A (1989) Analysis and training in information technology tasks. In Diaper D (ed) Task analysis for human-computer interaction, pp 15–55

Sufrin B (1982) Formal specification of a display-oriented text editor. Sci Comput Program 1:157–202

Sousa M, Campos J, Alves M, Harrison M, (2014) Formal verification of safety-critical user interfaces: a space system case study. In: Proceedings of the AAAI spring symposium on formal verification and modelling in human machine systems, pp 62–67

Spivey MJ (1989) The Z notation: a reference manual. Prentice-Hall, Upper Saddle River

Strunk EA, Yin X, Knight JC (2005) ECHO: a practical approach to formal verification. In: Proceedings of the 10th international workshop on formal methods for industrial critical systems, pp 44–53

Stückrath J, Weyers, B (2014) Lattice-extended coloured petri net rewriting for adaptable user interface models. Electron Commun EASST 67(13):13 pages. http://journal.ub.tu-berlin.de/eceasst/article/view/941/929

Thevenin D, Coutaz J (1999) Plasticity of user interfaces: framework and research agenda. In: Sasse A, Johnson C (eds) Proceedings of interact, pp 110–117

Thimbleby H (2007a) Interaction walkthrough: evaluation of safety critical interactive systems. In: Interactive systems. Design, specification, and verification. Springer, pp 52–66

Thimbleby H (2007b) User-centered methods are insufficient for safety critical systems. In: Proceedings of the 3rd human-computer interaction and usability engineering of the Austrian computer society conference on HCI and usability for medicine and health care. Springer, pp 1–20

Thimbleby H, Gow J (2008) Applying graph theory to interaction design. In: Gulliksen J, Harning MB, Palanque P, Veer GC, Wesson J (eds) Engineering interactive systems. Springer, pp 501–519

Thimbleby H (2010) Think! interactive systems need safety locks. J Comput Inf Technol 18 (4):349–360

Thimbleby HW, Gimblett A (2011) Dependable keyed data entry for interactive systems. Electron Commun EASST 45

Tu H, Oladimeji P, Li KY, Thimbleby HW, Vincent C (2014) The effects of number-related factors on entry performance. In: Proceedings of the 28th international BCS human computer interaction conference, pp 246–251

Turchin P, Skii R (2006) History and mathematics. URSS

Turner CS (1993) An investigation of the therac-25 accidents. Computer 18:9I62/93, 0700–001830300

van Glabbeek RJ, Weijland WP (1996) Branching time and abstraction in bisimulation semantics. J ACM 43(3):555–600

Wang H-W, Abowd G (1994) A tabular interface for automated verification of event-based dialogs. Technical report. DTIC Document

Wegner P (1997) Why interaction is more powerful than algorithms. Commun ACM 40(5):80–91

Weyers B, Baloian N, Luther W (2009) Cooperative creation of concept keyboards in distributed learning environments. In: Borges MRS, Shen W, Pino JA, Barthès J-P, Lou J, Ochoa SF, Yong J (eds) Proceedings of 13th international conference on CSCW in design, pp 534–539

Weyers B, Luther W, Baloian N (2010a) Interface creation and redesign techniques in collaborative learning scenarios. J Futur Gener Comput Syst 27(1):127–138

Weyers B, Burkolter D, Kluge A, Luther W (2010) User-centered interface reconfiguration for error reduction in human-computer interaction. In: Proceedings of the third international conference on advances in human-oriented and personalized mechanisms, technologies and services, pp 52–55

Weyers B, Luther W, Baloian N (2012a) Cooperative reconfiguration of user interfaces for learning cryptographic algorithms. J Inf Technol Decis Mak 11(6):1127–1154

Weyers B (2012) Reconfiguration of user interface models for monitoring and control of human-computer systems. Dissertation, University of Duisburg-Essen. Dr. Hut, Berlin

Weyers B, Burkolter D, Luther W, Kluge A (2012b) Formal modelling and reconfiguration of user interfaces for reduction of human error in failure handling of complex systems. J Hum Comput Interact 28(1):646–665

Weyers B, Borisov N, Luther W (2014) Creation of adaptive user interfaces through reconfiguration of user interface models using an algorithmic rule generation approach. Int J Adv Intell Syst 7(1&2):302–336

Weyers B (2015) FILL: formal description of executable and reconfigurable models of interactive systems. In: Proceedings of the workshop on formal methods in human computer interaction, pp 1–6

Weyers B, Frank B, Bischof K, Kluge A (2015) Gaze guiding as support for the control of technical systems. Int J Inf Syst Crisis Resp Manag 7(2):59–80

Whalen M, Cofer D, Miller S, Krogh BH, Storm W (2008) Integration of formal analysis into a model-based software development process. In: Formal methods for industrial critical systems. Springer, pp 68–84

Yin X, Knight JC, Nguyen EA, Weimer W (2008) Formal verification by reverse synthesis. In: Proceedings of international conference on computer safety, reliability, and security, pp 305–319

Yin X, Knight J, Weimer W (2009a) Exploiting refactoring in formal verification. In: Proceedings of dependable systems & networks, pp 53–62

Yin X, Knight JC, Weimer W (2009b) Exploiting refactoring in formal verification. In: Proceedings of the 2009 IEEE/IFIP international conference on dependable systems and networks, pp 53–62

Yin X, Knight JC (2010) Formal verification of large software systems. In: Proceedings of second NASA formal methods symposium, pp 192–201