Specifying And Prototyping Dynamic Human-Computer Interfaces For Stochastic Applications

C.W. Johnson
Department of Computer Science, University of York,
Heslington, York, United Kingdom, YO1 5DD.
E-mail: johnson@minster.york.ac.uk,
Telephone: (0904) 433376, Fax: (0904) 432767.

ABSTRACT

Formal methods are increasingly being used to support the software engineering of complex systems. A number of limitations restrict the utility of these techniques for the design of human-computer interfaces. Firstly, formal notations frequently abstract away from the temporal properties that affect usability. Secondly, specifications often fail to consider the stochastic, or probabilistic, behaviours that characterise human-computer interaction with distributed and concurrent applications. This paper proposes techniques to overcome these limitations. It is argued that temporal logic provides a means of representing sequential and concurrent properties of interaction. It is also demonstrated that logic can be used to explicitly represent assumptions about operator responses to high and low risk events. In order to support the validation of these assumptions we have implemented a tool which exploits Monte Carlo techniques to directly derive prototype simulations from temporal logic specifications of interactive systems.

Keywords: *formal methods; prototyping; Monte Carlo simulation.*

1 Introduction

In recent years there has been an increasing interest in the use of formal methods to support the software engineering of complex systems. NASA are applying mathematical specification techniques to assess the design of their Advanced Orbiting System [17]. Formal analysis is a necessary part of many design projects commissioned by the United Kingdom's Ministry of Defence [15]. Companies, including IBM [9] and Mitsubishi [13], are investing considerable resources in the application of 'industrial strength' specification techniques. Formal methods have also been applied to support interface development. For instance, Sufrin and He have extended the Z schema calculus to represent interaction with a text-editor [20]. Harrison and Thimbleby have recruited formal methods to specify high level requirements for many different interactive systems [8]. Took has developed a formal framework for the representation of graphical interfaces [21].

1.1 Time

Many specification techniques seem ill-equipped to support the development of human-computer interfaces to multi-user, multi-process applications. Formal notations such as Z abstract away from the temporal properties that can determine the quality of interaction. Delays in responses from application processes and other users can lead to breakdown and referential failure. Some user commands have an instant impact upon application processes whilst the effects of others may only be visible after many hours of operation [1]. In such situations, users have time for planned and coordinated responses. In other situations, users must respond immediately in order to avert potential disasters [12]. It is, therefore, extremely important that formal notations can represent temporal properties if they are to support interface development for dynamic applications.

1.2 Probability

Many specification languages also abstract away from the stochastic properties that characterise human-computer interaction with distributed and concurrent applications. A stochastic process is defined to be a series of operations whose results can best be described in terms of probabilities [6]. In manufacturing processes and telecommunications networks, designers can only specify traces of interaction with respect to the probable or anticipated behaviour of other systems or users. For instance, operator commands to start pumping will not always be successful if pumps fail once every two days. This has important implications for interface development. Wickens argues that users devise polling strategies to exploit the probabilistic behaviour of their system [23]. Less attention will be allocated to the observation of low probability errors, greater attention is allocated to the detection of high probability errors. If Wickens is correct then it is vital that designers can identification high and low risk situations during the development of human computer interfaces to complex and dynamic applications.

1.3 Overview

Sections 2 and 3 briefly review the strengths and weaknesses of two widespread techniques for probabilistic risk assessment. It is concluded that neither provides adequate support for the development of human-computer interfaces. Section 4 argues that temporal logics can be used to avoid these limitations. Section 5 continues this argument and shows how PCTL (Probabilistic Computation Tree Logic) can represent stochastic properties of human-computer interfaces. It is demonstrated that logic can be used to explicitly represent assumptions about operator responses to high and low risk events. Section 6 provides a means of validating the usability assumptions that can be embodied within logic specifications. The implementation of the Risklog simulation tool is described. This exploits PCTL specifications to directly derive prototype interfaces which mimic the probable behaviour of application processes. Section 7 presents the conclusions which can be derived from this work. Areas for future work are also proposed.

2 Markov Models

A number of formal notations might be recruited to support the development of human-computer interfaces. For example, Markov models provide a quantitative means of representing the ways in which events interact. Figure 1 illustrates an extremely simple model which relates maintenance rates, r_repair, and failure rates, r_fail. Davies uses this model in order to produce predictive equations about system behaviour [6]. For instance, the following formula relates the probability that a component is working, $p_working$, to its failure and repair rates at time t:

$$|p_working|_t = |\frac{1}{r_fail + r_repair} r_repair + r_fail\ e^{-(r_fail + r_repair)}|_t \quad (1)$$

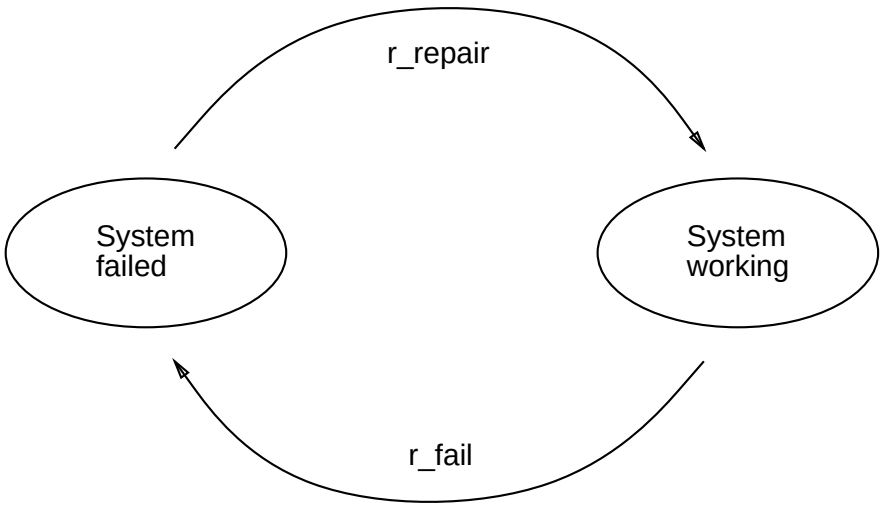

Figure 1: A simple Markov model

Figure 1 is extremely simple. It does, however, illustrate the point that the equations derived from Markov models provide a basis for predicting system behaviour. For instance, if a coolant pump fails six times every year and can be repaired once every two days then there is a 0.96 probability that the component is working at any particular time. A limitation of this approach is that operator responses to system failures are represented implicitly in the transitions between system states. Users must intervene to transform a coolant pump from the failed to the working state. Explicitly representing operators as components of a Markov model begs the question: what states can a user be in? In anticipation of the results of current research in the field of cognitive science, designers must seek alternative approaches.

3 Fault-Tree Analysis

Fault-trees are in widespread use amongst systems engineers. There are a number of industry standards which guide their industrial application. For instance, the fault-tree notation used in Figure 2 conforms to guidelines that have been drawn up by the European Federation of Chemical Engineers [5] and the United States' Nuclear Regulatory Commission [22]. The bottom-right branch of this fault-tree represents the risk that an operator will attempt to withdraw control-rods whose interlocks have been disabled. Figure 2 also shows how probabilities can be associated with the nodes of a fault-tree. Unfortunately, fault-trees provide an inadequate representation of time. The European Federation of Chemical Engineering's International Study Group On Risk Analysis concludes:

> "Fault-trees have difficulties with event sequences... parts of systems where sequence is important are, therefore, usually modelled using techniques more adept at incorporating such considerations" [5]

Gate conditions are assumed to hold simultaneously or at some time after their branch conjunctions and disjunctions. The leaves of a tree are assumed to hold simultaneously or at some time before their parents. This lack of temporal information is a significant limitation because event sequences affect the quality of interaction between an operator and their system. Determining the order in which displays

are presented and commands are issued forms an important stage in the development of man-machine interfaces.

4 CTL And Time

Logic suffers from similar problems to fault-trees. For example, the following formula states that an error message is received by a user if *pump_1* fails and user 1 confirms an error message:

$$warning_received \Leftarrow pump_1_failed \wedge \\ display_pump_1_warning \wedge user_1_warning_acknowledge \quad (2)$$

An warning is received if *pump_1* fails, a warning is displayed and *user_1* acknowledges the warning.

There is no notion of time in classical logic; formula (2) would be true if *user_1* responded before the error actually occurred. This is clearly not what the designer intended. This limitation can be avoided by using formalisms which explicitly represent time, such as Clarke and Emerson's Computation Tree Logic (CTL) [4]. Appendix A briefly introduces the syntax and semantics of CTL. Intuitive readings for the logic will be given throughout the remainder of the paper.

Designers can use CTL's X (read as 'next') operator to represent the sequential information that was missing in formula (2). The A operator can be read as 'in all traces of interaction'. For example, the following clause specifies that under all possible circumstances *user_1* responds in the next state after pump 1 fails:

$$always_receive_error \Leftarrow pump_1_failed \wedge \\ display_pump_1_warning \wedge AX\ user_1_warning_acknowledge \quad (3)$$

Information about an error is always received if *pump_1* fails and a warning is displayed in the present interval and in the immediate next interval of all possible traces of interaction *user_1* acknowledges the warning.

The previous formula shows how CTL can represent assumptions about operator responses to system failures. Designers can use such clauses to reason about potential interfaces. For example, it may not be possible to guarantee that *user_1* will always be able to immediately repond to an error in *pump_1*. Given divided attention and fatigue there may exist future traces in which the response is delayed. Designers can only specify that there exist certain traces of interaction during which *user_1* can immediately acknowledge the warning. This requirement can be represented using the E (read as 'in some traces of interaction') operator. The following clause specifies that in some traces of interaction *user_1* acknowledges the error in the next state after pump 1 fails:

$$possibly_receive_error \Leftarrow pump_1_failed \wedge \\ display_pump_1_warning \wedge EX\ user_1_warning_acknowledge \quad (4)$$

Information about an error is possibly received if *pump_1* fails and a warning is displayed in the present interval and in the immediate next interval of some traces of interaction *user_1* acknowledges the warning.

The choice of presentation strategy can profoundly affect the usability of an interface. The previous clauses illustrate how logic specifications provide a means of constructing a design without forcing com-

mitment to a particular implementation early in the development process. It is possible for designers to construct alternative designs in terms of high level abstractions rather than low level device primitives. These details can be added gradually as development progresses. For instance, the previous clause does not require that the warning display should continue to be presented until the user acknowledges it. Such presentation requirements can be represented using the $\mathcal{U}$ (read as 'until') operator:

$$\begin{aligned} &continue_display \Leftarrow \\ &\quad pump_1_failed \wedge E[display_pump_1_warning\ \mathcal{U} \\ &\quad user_1_warning_acknowledge \wedge user_1_start_stand_by_pump] \end{aligned} \tag{5}$$

Information about an error continues to be displayed if in some traces of interaction a warning is presented until *user_1* acknowledges the message and issues a command to start the reserve pumping system.

The $\mathcal{U}$ operator can also be used to represent requirements for multi-user systems. For instance, it might be required that a number of operators should coordinate their responses before the stand-by pumping equipment is started:

$$\begin{aligned} &cooperation \Leftarrow \\ &\quad pump_1_failed \wedge display_pump_1_warning \wedge A[\neg stand_by_pump_on\ \mathcal{U} \\ &\quad user_1_start_stand_by_pump \wedge user_2_start_stand_by_pump] \end{aligned} \tag{6}$$

There is cooperation if *pump_1* fails and a warning is presented in the present interval and in all possible traces of interaction the stand-by pump is not started until both *user_1* and *user_2* issue commands to start the reserve pumping system.

The stand-by pump is not turned on until both *user_1* and *user_2* provide input to start it. This clause can easily be generalised to more than two operators using additional clauses: *user_3*; *user_4* and so on [10].

5 PCTL And Probability

In many applications, such as distributed or concurrent systems, component failures can complicate operator tasks. Designers can only specify the probable behaviour of human computer interfaces to such applications. The following two clauses illustrate the need to consider probabilistic information during interface development:

$$\begin{aligned} &acknowledge_error \Leftarrow pump_1_failed \wedge \\ &\quad display_pump_1_warning \wedge EF\ user_1_warning_acknowledge \end{aligned} \tag{7}$$

$$\begin{aligned} &acknowledge_error \Leftarrow pump_1_failed \wedge \\ &\quad display_pump_1_warning \wedge EF\ user_2_warning_acknowledge \end{aligned} \tag{8}$$

The first clause states that an error is acknowledged if pump 1 fails and a warning is displayed in the present interval and eventually in some trace of interaction user 1 provides input to acknowledge the failure. The second clause states that an error is acknowledged if pump 1 fails and a warning is displayed in the present interval and eventually in some trace of interaction user 2 provides input to acknowledge the failure.

In order to determine whether operators can satisfy either of these requirements it is important that designers can represent and reason about the probability of $pump_1_failed$. If it is an extremely rare event then additional presentation resources must be devoted to $display_pump_1_warning$ so that users 1 and 2 are likely to observe the warning. Less presentation resources may be necessary in order to present information about high probability errors which can be anticipated by system operators [23].

PCTL (Probabilistic Computation Tree Logic) extends CTL in order to represent high and low probability situations which require operator intervention. $[f\ P\ y]$ intuitively means that the probability of f being true in the next state is y where y is a value in the range [0.0,1.0]. Appendix B briefly introduces the syntax and semantics of PCTL. The following clause illustrates the use of this notation. Figures from Associated Octel predict 76 failures per operating year for compressors [6]. If state transitions, occur every second then designers can specify that the probability of pump 1 failing in the next state is 0.964×10^{-6}:

$$[pump_1_failed\ P\ 0.000000964] \tag{9}$$

The probability of pump 1 failing in the next state is 0.964×10^{-6}.

If an interface is being developed for an existing system then designers may already have access to data about the behaviour of application processes. In other circumstances, designers must exploit generic data for system behaviour. It is important to note that there are many well developed techniques for the combination and analysis of such probabilities. The interested reader is directed to Bacchus' work for a more detailed discussion [2]. Given such data, it is possible for designers to reason about the behaviour of interfaces to stochastic processes. We know from formula (9) that there is a 0.964×10^{-6} probability of pump 1 failing in the next second, therefore, there is at least a 0.964×10^{-6} probability that $user_1$ and $user_2$ will be presented with a $pump_warning$ given clauses (7) and (8). Such analyses can help designers to allocate display resources. For instance, blow-back valves prevent materials from being forced back into a pump's inlet supply. Designers might identify a probability of one failure every 0.2×10^5 seconds:

$$[blow_back_1_failed\ P\ 0.000002] \tag{10}$$

The probability of $blow_back$ valve 1 failing in the next state is 2×10^{-5}.

As with pump failures, designers might allocate display resources to warn $user_1$ and $user_2$ of a blow back failure:

$$acknowledge_error \Leftarrow blow_back_1_failed \wedge \\ display_blow_back_1_warning \wedge EF\ user_1_warning_acknowledge \tag{11}$$

$$acknowledge_error \Leftarrow blow_back_1_failed \wedge \\ display_blow_back_1_warning \wedge EF\ user_2_warning_acknowledge \tag{12}$$

The first clause states that an error is acknowledged if blow back valve 1 fails and a warning is displayed in the present interval and eventually in some trace of interaction user 1 provides input to acknowledge the warning. The second clause states that an error is acknowledged if blow back valve 1 fails and a warning is displayed in the present interval and eventually in some trace of interaction user 2 provides input to acknowledge the warning.

From these formulae and the previous analysis of clause (9), interface designers can draw a number of conclusions about the behaviour of a potential interface. For any user, $blow_back$ and $pump$ failures will

be rare events. Considerable perceptual cues will, therefore, be required to alert users to these failures. The low probability of both events might persuade interface designers to present additional diagnostic information. In Norman's terms, sufficient context must be provided 'in the world' to ensure that operators are not forced to rely upon 'in the head' recollections of emergency procedures for rare failures [18]. It is vital that such requirements should emerge early in development when sensing devices can be allocated to meet the informational requirements of system operators. The previous clauses also capture the observation that a *blow_back* failure is more probable than a pump failure. Designers might use this information when developing training material and system documentation. Resources may be allocated in proportion to the perceived probability of particular system behaviours.

6 Prototyping

Using PCTL to identify high and low probability behaviours does not guarantee that operators will be able to detect and respond to particular displays. The Risklog prototyping tool has been developed to provide empirical support for the assumptions that may be embodied within PCTL specifications.

6.1 Risklog And Time

Design requirements, expressed in formal notations, provide the non-formalist with little idea of what it would be like to interact with a system. Prototypes provide a far better impression of the look and feel of a final implementation. They can be shown to system operators. They are amenable to experimental analysis. They can be used to determine the cognitive, perceptual, physiological and sociological demands that a control system might place upon its users. Several research groups have provided means of deriving partial implementations from temporal logic specifications [16]. For example, Prolog has been used to build the Tokio meta-interpreter [7]. Tokio can be used to implement *cooperation* (6) using the following clause:

$$risklog_cooperation : -pump_1_failed,$$
$$display_pump_1_warning, \quad (\neg stand_by_pump_on \; \mathcal{U}$$
$$user_1_start_stand_by_pump, user_2_start_stand_by_pump) \tag{13}$$

There is cooperation if *pump_1* fails and a warning is presented in the present interval and the stand-by pump is not started until both *user_1* and *user_2* issue commands to start the reserve pumping system.

There are, however, some CTL clauses which cannot be executed so easily. For instance, *continue_display* (5) only holds for some traces of interaction. Such clauses represent weak requirements which may or may not be fulfilled. They do little to direct the execution of a prototype. Designers must either omit such clauses from an executable specification or strengthen them to require that they must be satisfied by an implementation. Our intuition is that this process forms a necessary part of the refinement towards implementation.

6.2 Risklog And Probability

In the previous clause, if were asserted that *pump_1* failed during the first run of a prototype then it would also fail for every other run. Designers must change the assertions made for many different intervals of interaction in order to ensure that operators are faced with traces that accurately model the probable behaviour of a safety-critical system. Risklog extends Tokio by exploiting Monte Carlo techniques to avoid such limitations. The term 'Monte Carlo' refers to the use of random numbers to determine which

of a number of possible paths a simulation must follow. By analogy, if Risklog is faced with a choice of possible behaviours then it makes its decision by flipping a coin. The use of probabilities is analogous to weighting the coin in favour of a particular path or paths. In our example, the probability of *pump_1* failing, 0.964×10^{-6}, is compared with a random number in the range [0.0, 1.0]. If the random number is less than or equal to the probability then the fact is assumed to be true; the pump fails. If the random number is greater than the probability then the fact is assumed to be false; the pump does not fail. If a probability is increased then there is a greater chance that it will be larger than the random number generated by *simulate*. From this it follows that the corresponding paths of behaviour will more readily be chosen by Risklog:

$$risklog_simulate_cooperation : -simulate(pump_1_failed),$$
$$display_pump_1_warning, \quad (\neg stand_by_pump_on\ \mathcal{U}$$
$$user_1_start_stand_by_pump, user_2_start_stand_by_pump) \tag{14}$$

There is cooperation if risklog uses Monte Carlo techniques to simulate *pump_1* failing and a warning is presented in the present interval and the stand-by pump is not started until both *user_1* and *user_2* issue commands to start the reserve pumping system.

If propositions contain probabilistic information then Risklog provides a suite of facilities for determining which risk assessments hold for a particular interval. The Monte Carlo suite also calls the random number generators used by *simulate*. A $random_num(Num)$ predicate is true if Num unifies with a random number in the range [0.0,1.0]. This is generated using the linear congruent method [19]. The algorithm is seeded using the time at which the trace is started. These facilities are implemented using an interface between Prolog and the C programming language.

6.3 Risklog And Interaction

The clauses derived from formal risk analyses provide the non-formalist with an extremely poor impression of the 'look and feel' of potential implementations. Structured graphics systems provide means of introducing the graphical presentation details necessary in order to develop prototypes from the abstract clauses of interval temporal logic. Complex images can be represented in terms of their component parts. For example, a *reactor_error* display might consist of an icon representing a *rod_trip* as well as a message to *display_pump_1_warning*:

$$part(reactor_error, rod_trip_icon). \tag{15}$$
$$part(reactor_error, display_pump_1_warning). \tag{16}$$

The first clause states that the *rod_trip_icon* is part of the *reactor_error* display. The second clause states that the *display_pump_1_warning* is part of the *reactor_error* display.

Figure 3 illustrates how the image of these parts can be described in terms of regions: a background region; a text region and a *rod_trip_icon* region. The *rod_trip_icon* region might be further decomposed into sub-regions containing lines, squares or circles. Each region has properties, such as size and position, attributes, such as font and pattern, and a behaviour, such as whether or not it is selectable. Elements of the resulting region structures can be directly incorporated into Risklog simulations:

$$risklog_present_cooperation : -simulate(pump_1_failed),$$
$$display(reactor_error), part(reactor_error, display_pump_1_warning),$$
$$(\neg stand_by_pump_on\ \mathcal{U}\ user_1_start_stand_by_pump,$$
$$user_2_start_stand_by_pump) \quad (17)$$

There is cooperation if Risklog uses Monte Carlo techniques to simulate *pump_1* failing and a warning display is presented and that display contains an error message about the failure in *pump_1* in the present interval and the stand-by pump is not started until both *user_1* and *user_2* issue commands to start the reserve pumping system.

The resulting Risklog architecture is illustrated in Figure 4. A companion paper in this volume provides greater detail on the techniques that are used to implement graphical interfaces for stochastic, multi-user applications [11].

7 Conclusion And Further Work

In this paper we have argued that PCTL can be used to specify temporal properties of human-computer interfaces. This notation has been enhanced so that designers can reason about operator interaction with stochastic processes. It has been argued that the resulting PCTL specifications provide little impression of what it would be like to interact with a potential implementation. The Risklog prototyping tool has been developed to avoid such limitations. Monte Carlo techniques are used to simulate low and high risk traces of interaction. These simulations can be shown to potential users and are amenable to experimental analysis.

There is an important limitation with Risklog's low probability decision procedure; high probability behaviours are less likely to occur. It is, therefore, difficult to simulate the mixture of high and low probability failures which often frustrate the use of complex computer systems. Current work is exploring tolerances, or bands, within which low probability propositions will occasionally be evaluated as true irrespective of the random numbers generated by Monte Carlo techniques. Low probability, high cost failures, typically, have a greater impact upon safety than high probability, low cost errors. Further work intends to enhance Risklog to consider the costs of failures when simulating situations that require operator intervention. Utility functions will be introduced into PCTL specifications [3].

Acknowledgements

Thanks are due to the members of the Department of Computer Science, University of York and, in particular, to Prof. M.D. Harrison and Prof. J. McDermid for comments on early drafts of this paper.

A The Syntax and Semantics Of CTL

p denotes an atomic proposition and is a formula, f_i denotes a formula or sub-formula. $f_1 \wedge f_2$ is a formula denoting conjunction, $\neg f$ is a formula denoting negation. $EX\ F_j f$ is a formula; f is true in the next state of possible future F_j. $A[f_1 \mathcal{U} f_2]$ is a formula; there exists a prefix for all futures such that f_2 is true in the last state of the prefix and f_1 holds at all other states in that prefix. $E[f_1 \mathcal{U} f_2]$ is a formula; there exists a prefix for a future such that f_2 is true in the last state of the prefix and f_1 holds at all other states in that prefix.

Clarke and Emerson introduce a model for the semantics of CTL [4]. $M = (S, F_1, ..., F_k, L)$ where: S is a countable set of states; $F_i \subset S \times S$ is a relation on S giving the transitions over a possible future i; L is an assignment of true atomic propositions in each state. Let $F = F_1 \cup F_2 ... \cup F_k$ and $\forall s \in S, \exists s' \in S(s, s') \in F$. A *trace* is an infinite sequence of states $(s_0, s_1, s_2, ...) \in S^\omega$ where $\forall i (s_i, s_{i+1}) \in F$. To any structure M and state $s \in S$ of M there corresponds a tree of possible future states with root labelled s_0 such that $s \rightarrow s_i$ is a branch in the tree iff $(s, s_i) \in F$. $M, s_0 \models f$ denotes that formula f is true in state s_0 of future tree M. The M is omitted if it is understood by the preceding context. The $\models$ relation can be defined inductively:

- $s_0 \models p$ iff $p \in L(s_0)$;
- $s_0 \models \neg f$ iff not$(s_0 \models f)$;
- $s_0 \models f_1 \wedge f_2$ iff $s_0 \models f_1$ and $s_0 \models f_2$;
- $s_0 \models EX\ F_k f$ iff
$\exists s_1 \in S(s_0, s_1) \in F_k \wedge s_1 \models f$;
- $s_0 \models A[f_1 \mathcal{U} f_2]$ iff for all traces
$\exists i[i \geq 0 \wedge s_i \models f_2 \wedge \forall j(0 \leq j \wedge j < i \wedge s_j \models f_1)]$;
- $s_0 \models E[f_1 \mathcal{U} f_2]$ iff for some trace
$\exists i[i \geq 0 \wedge s_i \models f_2 \wedge \forall j(0 \leq j \wedge j < i \wedge s_j \models f_1)]$;
- $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$;
- $f_1 \Rightarrow f_2 \equiv \neg f_1 \vee f_2$;
- $EF f_1 \equiv E[true \mathcal{U} f_1]$ states that for some trace there exists a state in which f_1 is true;
- $AG f_1 \equiv \neg EF \neg f_1$ states that f_1 is true for all states in all traces;
- $AX f \equiv \neg EX \neg f_1$ states that f_1 holds in all next states.

B Syntax and Semantics of PCTL

The syntax of PCTL extends CTL with an additional path formula. $[f\ P\ y]$ intuitively means that the probability of f being true in the next state of all possible futures is y where y is a value in the range [0.0,1.0]. Informally, the probability of a proposition is given by the number of possible futures in which that proposition is true divided by the number of all possible futures. For instance, if *pump_1* fails in two out of ten possible states then the probability of *pump_1* failing in the next state is 2/10 or 0.2. More formally, we can state that $s_0 \models [f\ P\ y]$ iff:

$$\frac{\#\{\forall F_k \in F | (s_0, s_1) \in F_k \wedge s_1 \models f \bullet F_k\}}{\#\{\forall F_j \in F | (s_0, s_1) \in F_j \bullet F_j\}} = y$$

Real-time failure rates are represented in PCTL by linking all state transitions to a clock, $\forall i (s_i, s_{i+1}) \in F$. Ladkin has developed similar techniques for grounding Allen's convex interval logic in real-time [14].

References

[1] J.L. Alty and J. Mullin. Dialogue specification in the GRADIENT dialogue system. In A. Sutcliffe and L. Macaulay, editors, *People And Computers V*, pages 151 – 168. Cambridge University Press, Cambridge, United Kingdom, 1989.

[2] F. Bacchus. *Representing And Reasoning With Probabilistic Knowledge.* The MIT Press, Cambridge, United States of America, 1990.

[3] A. Burns. The HCI component of dependable real-time systems. *The Software Engineering Journal*, 6(4):168–174, July 1991.

[4] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In D. Kozen, editor, *Logic of Programs 1981 - Proceedings*, LNCS 131, pages 52–71. Springer-Verlag, Berlin, FDR, 1982.

[5] A.P. Cox, editor. *Risk Analysis In The Process Industries: The Report Of The International Study Group On Risk Analysis.* EFCE No. 45. The Institute Of Chemical Engineers, Rugby, United Kingdom, 1985.

[6] K.R. Davies. Techniques for the identification and assessment of major accident hazards. In J. Burgoyne, editor, *The Assessment And Control Of Major Hazards*, pages 289–308. Pergamon Press, Oxford, United Kingdom, 1985.

[7] M. Fujita, S. Kono, H. Tanaka, and T. Moto-Aka. Tokio: Logic programming based on temporal logic and its compilation to PROLOG. In E. Shapiro, editor, *Third International Conference On Logic Programming*, LNCS 225, pages 695–708. Springer-Verlag, Berlin, FDR, 1986.

[8] M.D. Harrison and H.W. Thimbleby, editors. *Formal Methods In Human Computer Interaction.* Cambridge University Press, Cambridge, United Kingdom, 1989.

[9] A. Jack. It's hard to explain but Z is much clearer than English. *The Financial Times*, page 22, 12 April 1992.

[10] C.W. Johnson. Applying temporal logic to support the specification and prototyping of concurrent multi-user interfaces. In D. Diaper and N. Hammond, editors, *People And Computers VI: Usability Now*, pages 145–156. Cambridge University Press, Cambridge, United Kingdom, 1991.

[11] C.W. Johnson. A formal approach to the presentation of CSCW systems. In J.L. Alty, D. Diaper, and S. Guest, editors, *People And Computers VIII*, pages 335–352. Cambridge University Press, Cambridge, United Kingdom, 1993.

[12] C.W. Johnson and M.D. Harrison. Using temporal logic to support the specification and prototyping of interactive control systems. *International Journal Of Man-Machine Studies*, 36:357–385, 1992.

[13] K. Katsuyama, F. Sato, T. Nakakawaji, and T. Mizuno. OSI testing environment based on standardized formalisations. In J. Quemada, J. Mañas, and E. Vazquez, editors, *Formal Description Techniques III*, pages 287–294, North Holland, 1991. Elsevier.

[14] P. Ladkin. Primitives and units for time specifications. In T. Kehler, S.Rosenschein, R. Filman, and P.F. Patel-Schneider, editors, *Proceedings Of The Fifth Annual Conference On Artificial Intelligence*, pages 354–359. Morgan Kaufman, Los Altos, United States of America, 1986.

[15] Ministry Of Defence. *Requirements for the Procurement of Safety Critical Software*, MOD DEF-STAN 0055, London, United Kingdom, 1991.

[16] B. Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, United Kingdom, 1986.

[17] National Aeronautic and Space Administration. *Advanced Orbiting Systems - Architectural Specification For The CCSDS Secretariat*, Washington DC, United States of America, 1989.

[18] D.A. Norman. The 'problem' with automation : Inappropriate feedback and interaction not 'over-automation'. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 137–145. Clarendon Press, Oxford, United Kingdom, 1990.

[19] R. Sedgewick. *Algorithms*. Addison Wesley, Wokingham, United Kingdom, 1988.

[20] B. Sufrin and J. He. Specification, refinement and analysis of interactive processes. In M. D. Harrison and H. W. Thimbleby, editors, *Formal methods in Human Computer Interaction*, pages 153–200. Cambridge University Press, Cambridge, United Kingdom, 1990.

[21] R. Took. Integrating inheritance and composition in an objective presentation model for multiple media. In F.H. Post and W. Barth, editors, *EUROGRAPHICS '91*, pages 291–303. Elsevier Science Publications, North Holland, Netherlands, 1991.

[22] W.E. Vesely. The fault tree handbook. Technical Report USNRC NUREG 0492, United States' Nuclear Regulatory Commission, Washington DC, United States of America, January 1981.

[23] C.D. Wickens. *Engineering Psychology And Human Performance*. C.E. Merrill Publishing Company, London, United Kingdom, 1984.

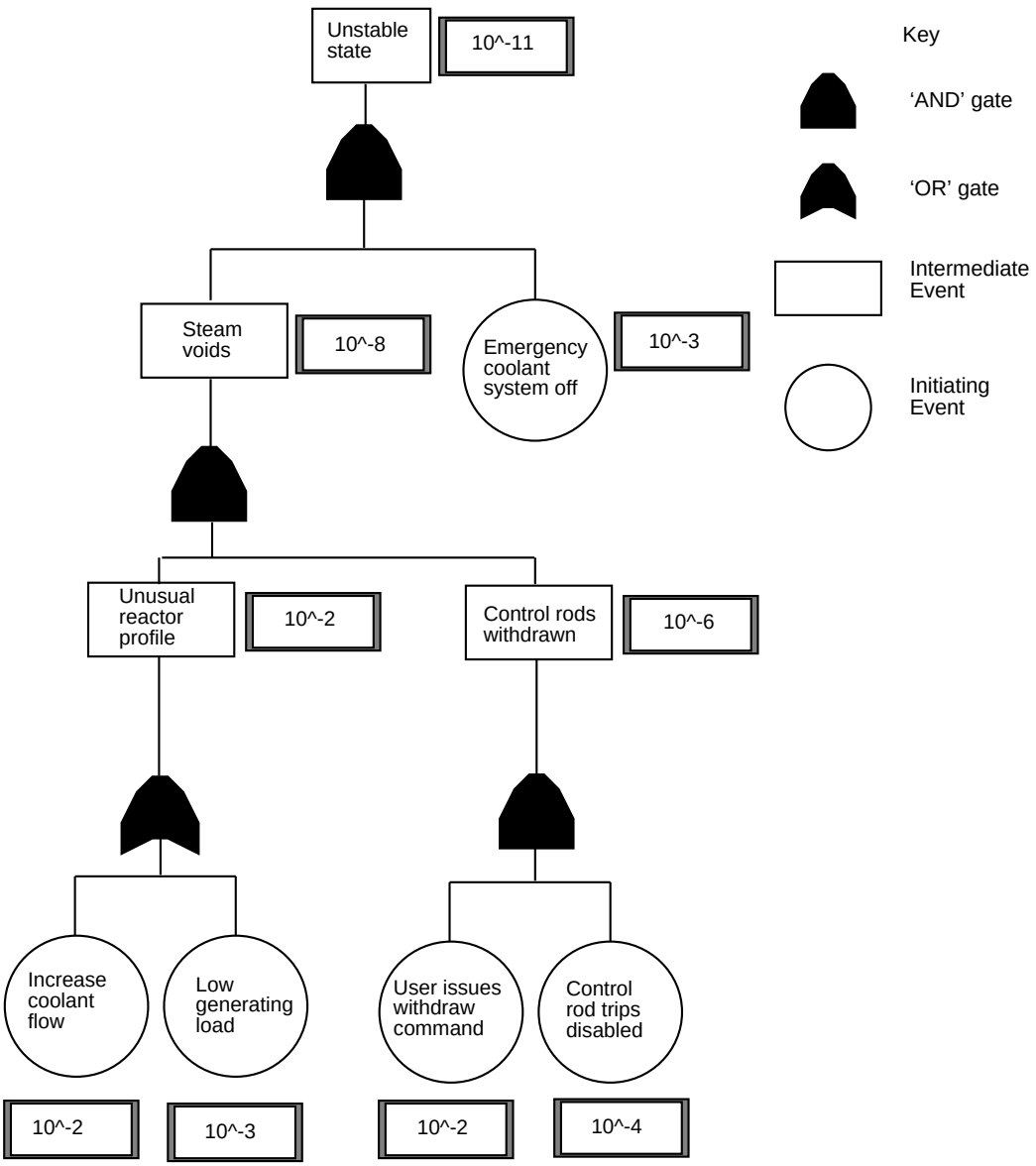

Figure 2: Fault-tree Representing Quantified Probabilities

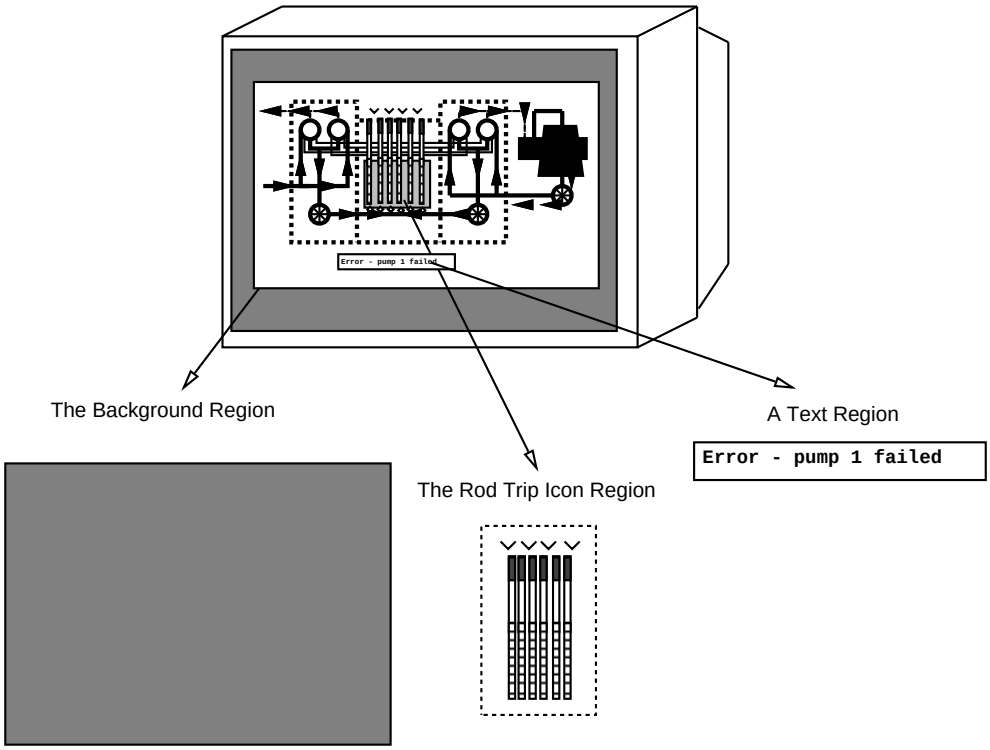

Figure 3: The region decomposition for part of the *reactor_unstable_warning*

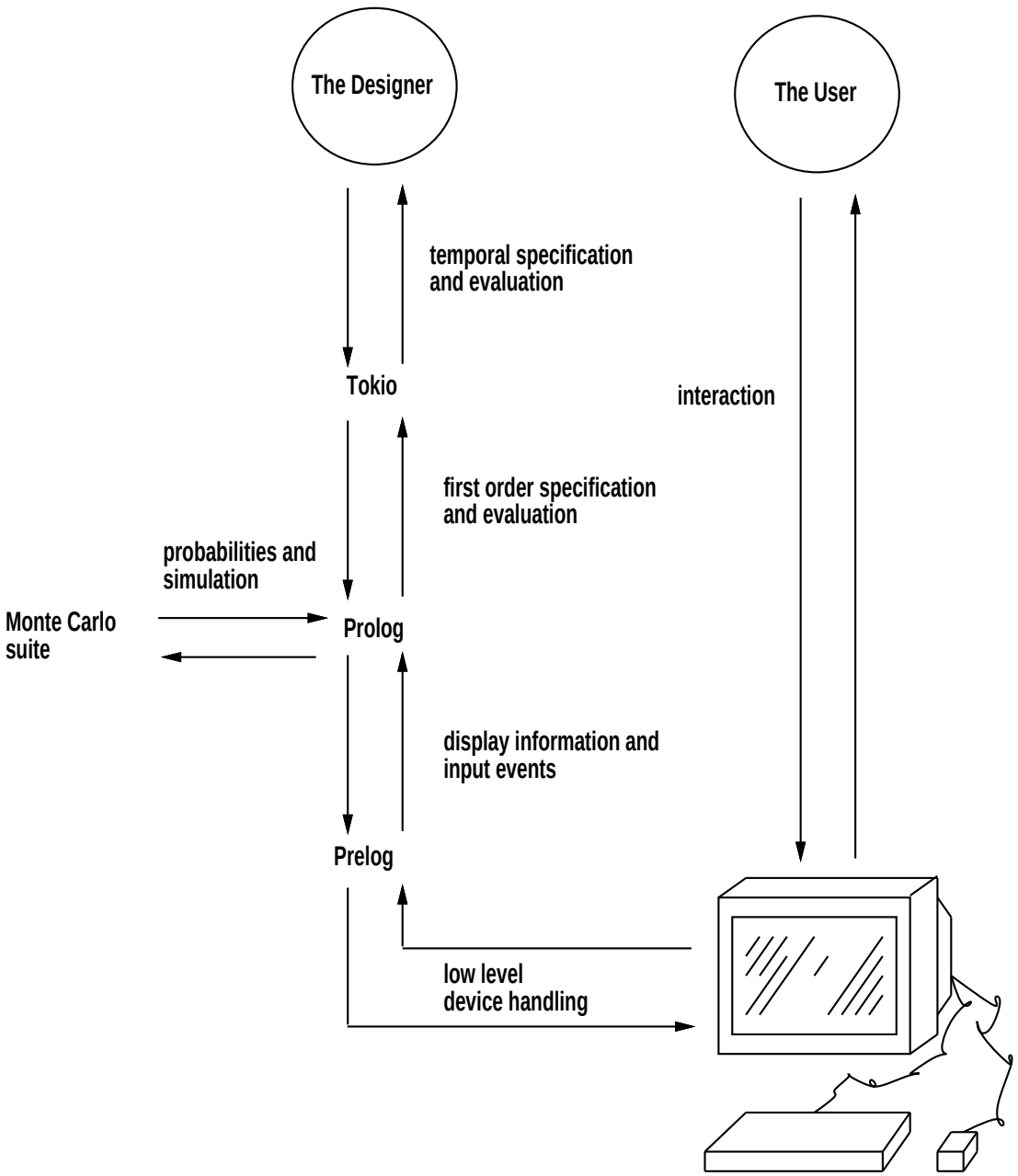

Figure 4: The extended Risklog architecture