COMPUTING SCIENCE

Formal Techniques for Requirements Analysis for Safety-Critical Systems

R. de Lemos, A. Saeed and T. Anderson

TECHNICAL REPORT SERIES

No. 468 January, 1994
Formal Techniques for Requirements Analysis for Safety-Critical Systems

R. de Lemos, A. Saeed and T. Anderson

Abstract

Formal support for the different activities performed during requirements analysis demands the utilisation of a set of formal notations and techniques whose features and expressive power match the characteristics of the activities. Selecting an appropriate formal technique for an activity allows emphasis to be placed on pertinent characteristics of the system, enabling the technique to work to its own strengths. In order to facilitate the utilization of different formal techniques, in this paper, we introduce an event/action model (E/A model) as a common foundation for models of system behaviour. To show the flexibility of the E/A model, we incorporate its concepts into two different classes of formalisms.
Bibliographical details

DE LEMOS, Rogério Sergio Neves

Formal Techniques for Requirements Analysis for Safety-Critical Systems
[By] R. de Lemos, A. Saeed and T. Anderson


(University of Newcastle upon Tyne, Computing Science, Technical Report Series, no. 468)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
SABED, Amer
ANDERSON, Thomas

Abstract

Formal support for the different activities performed during requirements analysis demands the utilisation of a set of formal notations and techniques whose features and expressive power match the characteristics of the activities. Selecting an appropriate formal technique for an activity allows emphasis to be placed on pertinent characteristics of the system, enabling the technique to work to its own strengths. In order to facilitate the utilization of different formal techniques, in this paper, we introduce an event/action model (E/A model) as a common foundation for models of system behaviour. To show the flexibility of the E/A model, we incorporate its concepts into two different classes of formalisms.

About the author

Mr. R. de Lemos is a Research Associate in the Departments of Computing Science and Chemical and Process Engineering at the University of Newcastle upon Tyne.

Dr. A. Saeed is a Research Associate in the DCSC, Department of Computing Science at the University of Newcastle upon Tyne.

Professor T. Anderson is a Professor in the Department of Computing Science at the University of Newcastle upon Tyne.

Suggested keywords

FORMAL TECHNIQUES
OPERATIONAL FORMALISMS
PROPERTY-ORIENTED FORMALISMS
REQUIREMENTS ANALYSIS
SAFETY-CRITICAL SYSTEMS
TIMELINESS REQUIREMENTS

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.6425 658.47
U.D.C. 681.322.06 519.718
Formal Techniques for Requirements Analysis for Safety—Critical Systems

Rogério de Lemos\textsuperscript{1,2}, Amer Saeed\textsuperscript{3} and Tom Anderson\textsuperscript{1,3}

\textsuperscript{1}Department of Computing Science
\textsuperscript{2}Department of Chemical and Process Engineering
\textsuperscript{3}BAe Dependable Computing Systems Centre
University of Newcastle upon Tyne, NE1 7RU, UK

Abstract

Formal support for the different activities performed during requirements analysis demands the utilisation of a set of formal notations and techniques whose features and expressive power match the characteristics of the activities. Selecting an appropriate formal technique for an activity allows emphasis to be placed on pertinent characteristics of the system, enabling the technique to work to its own strengths. In order to facilitate the utilization of different formal techniques, in this paper, we introduce an event/action model (E/A model) as a common foundation for models of system behaviour. To show the flexibility of the E/A model, we incorporate its concepts into two different classes of formalisms.

Keywords: safety—critical systems, requirements analysis, timeliness requirements, formal techniques, property—oriented formalisms, operational formalisms.

1. Introduction

For formal support to be effective in system development it is essential to examine the demands imposed by the context in which application is envisaged. If formal support is considered in isolation, more emphasis may be placed on the mathematical properties of a formal notation than a method to guide its application, rendering the support impractical. In this paper, we focus on the provision of formal support for the requirements stage of system development, for the class of process control systems.

In addition to the usual features that should be provided by formal support, such as an unambiguous notation, checks for consistency and completeness, a number of specific features arise from the class of systems under consideration.

1. Compatibility with the underlying models of control theory, such as differential equations and variables that are functions of time.

2. Support traceability (by formal refinement and verification) between the results of system safety techniques, such as hazard identification and software specifications.

3. Promote the analysis of distinct properties of system behaviour from different perspectives (e.g. timing and reliability).

During requirements analysis, the broad range of information that must be encoded and analysed suggests two alternatives for the basis of formal support: the employment of a
single wide spectrum notation, such as Durational Calculus [1], [2] or extended CCS [3]; or a number of specialized notations, such as RTL [4] and Statecharts [5]. In the case of wide spectrum notations features 1 and 2 are supported, in the sense that it is possible with appropriate extensions to relate to the models of control theory and relate the results of system safety analysis to software specifications. However, because of the many interacting features such notations tend to be complex making it difficult to extract a suitable subset of the notation for specific analysis, working against feature 3. On the other hand, a suitable set of specialized notations will support features 1 and 3, in the sense that an appropriate formalism can be selected for a related class of properties, and the different notations permit a selective approach to the analysis of the requirements. However, an inconvenience of this approach is that difficulties arise when attempting to link specifications expressed in different formalisms which works against feature 2.

In order to facilitate the systematic analysis of the requirements, from the safety perspective, we have proposed a framework [6]. The structure of the framework follows from the analysis of the system that identifies the key components and their interactions, establishing the domains of analysis and their inter-relationships, respectively. This process is conducted recursively, each decomposition leading to a lower level of abstraction. The framework is defined by associating its phases with the domains of analysis, and the ordering of the phases with the identified inter-relationships.

The formal support for the framework consists of formal notations and techniques that are used to represent and reason about the behaviour and properties of a system. For the framework, we advocate, when necessary, the utilisation of more than one formalism. Although formal techniques have been classified in a number of different ways, in the context of our proposed approach two classes of formalisms are identified: property-oriented and operational. The degree of application of one class of formalism versus the other is related to the level of abstraction being considered: at higher levels of abstraction there is a natural tendency to use property-oriented formalisms, whereas at lower levels operational formalisms dominate.

The rest of the paper is presented as follows. Firstly, a structure for the class of process control systems is defined, and a classification of system variables presented. In section 3, we introduced the E/A model and its corresponding PEA notation to be employed in the description of system behaviour. Section 4 presents an extract of a case study and its analysis from the perspective of the PEA notation. Section 5 describes how the E/A model is represented in propriety-oriented and operational formalisms, and illustrates their application on the case study. Finally, section 6 contributes with some concluding remarks.

2. System Structure
A system structure is obtained by performing successive refinements on the system and its components (recursively, a component can be considered to be another system). Refinement ceases when a component is considered to be atomic. To the refinement process we associate levels of abstraction, which facilitate the representation of a partial view of the system, suppressing detail that is irrelevant to that view. For the class of process control systems, at the first refinement performed to the system structure, three basic components are defined: the physical process or plant, the controller, and the operator.

A detailed description of the interaction between components consists of the definition of the interface between the components (structural part) and the behaviour observed at that interface (dynamic part). In order to model the behaviour of a system, we start out by introducing the following system variables: input variables \( V_u \) and output variables \( V_y \) which describe the external behaviour of the system, and state variables \( V_x \) which describe the state of the system. A variable is represented by a function which maps time \( (T_v) \) into a set of values \( (V_v) \) of the variable, that is, \( v_i(t) \): \( T_v \rightarrow V_v \).

The passage of time is represented by a time structure \( (T,<,\Delta,+,\cdot) \), where \( T \) is a non-empty set of time points, \( < \) a strict total order, \( \Delta (\in \mathbb{R}^+) \) the granularity, and \( + \) the addition operation on \( T \). Depending on \( \Delta \) the time structure can either be dense (isomorphic to the reals or rational numbers) or discrete (isomorphic to the integers) [7].

Both the value and time domains of a variable are partitioned into disjoint subsets: anticipated and unanticipated elements. For the value domain of a variable, the anticipated subset \( (V_v^a) \) contains those values that are expected to occur, while the unanticipated subset \( (V_v^u) \) contains the remainder \( V_v = V_v^a \cup V_v^u \) (A similar partition is made in the time domain.) For some variables, it will also be necessary to stipulate constraints on the dynamic behaviour of the values of the variables with respect to time, e.g. maximum rate of change in the value of a variable.

3. Behaviour Description

3.1. Event/Action Model (E/A Model)

In order to describe the behaviour of real-time safety-critical systems, which exhibit both continuous and discrete behaviours, we introduce the event/action model (E/A model). The E/A model provides a set of primitive concepts which enable the modelling of the system behaviour, in terms of the system predicates (predicate over system variables). In the approach taken, those variables which are continuous have their behaviour discretised according to imposed thresholds and discontinuities that they are subjected to.

The E/A model is based on primitive concepts such as events, actions and states, and the concept of a time structure (or timeline). The state of a system is the information that, together with the system input, determines the behaviour of the system. A transition
represents a transformation in the system state. The system state is modified by the occurrence of events and the execution of actions. An event is a temporal marker of no duration which causes or marks a transition. An action is the basic unit of activity which implies duration. The duration of an interval is the time distance between the two events that define the interval. Apart from events, actions and states which describe the behaviour of process control systems, the E/A model also takes into account the timing uncertainties associated with them.

The motivation for selecting these primitive concepts is twofold: they have been used as primitives in several real-time specification languages [8], [4], and they have meaningful interpretations at different levels of abstraction. These concepts provide flexibility, enabling descriptions to be given of system behaviour ranging from the activities of the physical entities of the plant to the temporal ordering of the computational tasks of the control system. The main features of the E/A model are: the primitive concepts can be expressed in different classes of formalisms, both discrete and dense time structures are supported, and timing constraints can be depicted graphically.

3.1.1. Primitive Functions of the E/A Model

The primitive concepts of the E/A model are related to the timeline by two types of primitive functions: point and interval. A point function is a temporal marker of no duration, represented as a cut in the timeline, which models events. An interval function denotes a duration, represented as a contiguous section of the timeline, which models states and actions. The timing uncertainties associated with point and interval functions can be represented in terms of a utility function.

The definition of the primitive functions is realised in terms of system predicates in both value and time domains. The primitive functions have time and instance number as parameters, the instance number specifies the number of times for which the function has been true. The instance number is convenient in describing the behaviour of discrete variables, or continuous variables that are discretised, because it allows references to the past and future behaviours of the variable.

State and Transition Predicates

To express relationships between system variables we define a system predicate which is a predicate built using standard mathematical (e.g. + and −), logical (e.g. ∧ and ¬) and relational operators (e.g. < and ≤), and a free variable for each \( v_i \) of the type \( V_v \), and a free time variable \( t \) of the type \( T_v \), and constants of the type \( V_v \) or \( T_v \). System predicates are used to define both state and transition predicates. A state predicate is defined from a system predicate by associating to the former a superscript that denotes an instance number. The first instance of a state predicate is defined as follows:
\( \forall t \in T: [\ sp(t) \iff sp(t) \land (\exists t_1 \in T: t_1 \leq t \land (\forall t_2 \in T: t_1 < t_2 \Rightarrow \neg sp(t_2)) \land (\forall t_3 \in T: t_1 \leq t_3 \leq t \Rightarrow sp(t_3))] \).

The ith instance of a state predicate, represented by the index \( i \) (\( i > 1 \)), is defined as follows:

\( \forall i \in I^+: i > 1 \Rightarrow \)

\( \forall t \in T: [sp(i) \iff sp(t) \land (\exists t_1, t_2, t_3 \in T: t_1 < t_2 < t_3 < t \land \forall t_4 \in T: t_1 \leq t_4 < t_2 \Rightarrow sp(t_4)^{i-1} \land \forall t_5 \in T: t_2 < t_5 < t_3 \Rightarrow \neg sp(t_5) \land \forall t_6 \in T: t_3 \leq t_6 \leq t \Rightarrow sp(t_6)] \).

In order to capture a transition predicate we introduce the double bar operator "||". Figure 1 depicts the possible transition predicates that can be obtained from a system predicate. By applying the operator "||" to a system predicate we capture the first point in time at which a system predicate becomes true or false. This transition, known as a closed transition, is defined as follows:

\( \forall t \in T: [ ||^c (sp(t)) \iff sp(t) \land (\exists t_1 \in T: t_1 \leq t \Rightarrow \forall t_2 \in T: t_1 < t_2 < t \Rightarrow \neg sp(t_2))] \).

By applying the operator "||" to a system predicate we capture the last point in time just before a system predicate becomes true or false. This transition, known as open transition, is defined as follows:

\( \forall t \in T: [ ||^o (sp(t)) \iff \neg sp(t) \land (\exists t_1 \in T: \forall t_2 \in T: t_1 < t_2 > t \Rightarrow sp(t_2))] \).

![Figure 1. Transitions over system predicates.](image)

In order to capture the instance number of a transition, we introduce the bar operator with an index \( i \) added to it "\( |i| \)". The first instance of a closed transition is defined as follows:

\( \forall t \in T: [|^c (sp(t)) \iff ||^c (sp(t)) \land (\forall t_1 \in T: t_1 < t \Rightarrow \neg ||^c (sp(t_1))] \).

The ith instance of a closed transition, represented by the index \( i \) (\( i > 1 \)), is defined as follows:
\[ \forall t \in T: \forall i \in I^+: \ [i > 1 \Rightarrow \mathbf{I}^c(t) \Leftrightarrow \mathbf{I}^c(sp(t)) \land (\exists t_1 \in T: t_1 < t \land \mathbf{I}^c_{t-1}(sp(t_1)) \land (\forall t_2 \in T: t_1 < t_2 < t \Rightarrow \neg \mathbf{I}^c(sp(t_2))))]. \]

The definition of the single bar operator for the open transition is similar to the one for the closed transition. In the sequel, unless otherwise mentioned, we use two abbreviations in the notation: a bar operator without a superscript will refer to a closed transition, and a bar operator without a subscript will refer to the first instance of a transition. For example, the transition \( \mathbf{I}(sp(t)) \) captures the first instance of the system predicate \( sp(t) \) when the predicate becomes true.

In the definition of the primitive functions, conditions (state and transition predicates) capture the value domain properties of the functions, and time points capture the time domain properties of the functions. In order to distinguish the value domain from the time domain definitions, the subscript "V" or "T", respectively, is added to the name of the function. The definitions of the primitive functions, in the value domain, will be made only in terms of the closed transition although the open transition could have been used instead. As a consequence, the upper and lower boundaries of a time interval will be respectively closed ("["") and open ("("), however other combinations could be employed depending on the type of transitions being considered.

**Point Function**

A point function \( E(t,i) \) is a function which maps the timeline \( T \) the set of all of its points) and the number of instances of an event \( I^+ \) the set of the nonnegative integer numbers) into a Boolean.

In the value domain the definition of the point function is in terms of the ith instance of a condition which is specified by a transition predicate:

\[ E_V(t,i): T \times I^+ \rightarrow B \]
\[ \forall t \in T: \forall i \in I^+: [E_V(t,i) \Leftrightarrow \mathbf{I}(sp_E(t))]. \]

In the time domain the definition of the point function is in terms of the time point constants \( t_E^i \) at which an event has occurred for the ith time:

\[ E_T(t,i): T \times I^+ \rightarrow B \]
\[ \forall t \in T: \forall i \in I^+: [E_T(t,i) \Leftrightarrow t = t_E^i] \]
\[ t_E^i \in T_E = \{t_E^1, t_E^2, \ldots\}. \]

Instead of the notation \( (t_E^i) \), the occurrence function \( @(E,i) \) from RTL could have been employed [4].

The value and time domain definitions of a point function are alternative forms for describing the occurrence of the same event.
∀t ∈ T; ∀i ∈ I+: [E_r(t,i) ↔ E_T(t,i)].

A relative time representation of the timeliness requirements of a sequence of events can impose one of three basic types of timing constraints [9]:

- **minimum** — no less than t time units must elapse between the occurrence of two events;
- **maximum** — no more than t time units must elapse between the occurrence of two events;
- **durational** — exactly t time units must elapse between two events.

In a similar way as presented for RTL [4], in the following we state two *monotonicity* properties which are associated with the point function:

- **uniqueness property** — at most one time point can be associated with each occurrence of an event, i.e. the same instance number of an event cannot happen at two distinct time points:

  \[∀t, t' ∈ T; ∀i ∈ I^+: [E(t,i) ∧ E(t',i) → t = t'].\]

- **ordering property** — if the ith occurrence of an event happens, then the previous occurrences of same event must have happened earlier, hence two distinct occurrences of the same event must happen at different time points:

  \[∀t, t' ∈ T; ∀i, j ∈ I^+: [E(t,i) ∧ E(t',j) ∧ i < j → t < t'].\]

For discrete timelines, in order to observe the occurrence of all instances of the same event, the granularity of the timeline should be smaller than the smallest time interval between the occurrence of any two events. If the same event occurs more than once between two ticks of the timeline, only one of the occurrences will be observed.

**Interval Function**

An *interval function* A(t,i) is a function which maps the timeline and the number of instances of an action (or a state) into a Boolean. (In the following, the definition of the interval function will be restricted to the execution of an action, however it could be extended to the holding of states.) An action is manifested in the system by its associated events: the *start event* that marks the initiation of an action, and the *finish event* that marks the completion of an action.

In the value domain, the start and finish events are defined, respectively, as the *start condition* and the *finish condition*. The start condition defines the condition that triggers the execution of an action, it is defined as follows:
\[ \uparrow A_V(t) : T \rightarrow B \]
\[ \forall t \in T: [\uparrow A_V(t) = \langle (sp_{tA}(t)) \rangle]. \]

The finish condition defines the condition in which an action terminates its execution, it is defined as follows:

\[ \downarrow A_V(t) : T \rightarrow B \]
\[ \forall t \in T: [\downarrow A_V(t) = \langle (sp_{tA}(t)) \rangle]. \]

Apart from the start and finish conditions, the invariant condition is another necessary condition for the execution of an action. This condition is expressed in terms of a state predicate that has to hold true while the action is being executed:

\[ A_V^{\text{INV}}(t) : T \rightarrow B \]
\[ \forall t \in T: [A_V^{\text{INV}}(t) = (sp_{tA}(t))]. \]

The instance number of the conditions does not match with the instance number of the transition and state predicate because the transition and state predicates on their own are not sufficient for an action to start or finish its execution. It might be the case, for example, that although the transition predicate representing the start condition of an action holds true, the action does not start its execution because the invariant condition associated with that action does not hold true.

In the value domain, the interval function is defined in terms of the transition predicates corresponding to the start and finish conditions, and the state predicate corresponding to invariant condition. This is represented as follows:

\[ A_V(t) : T \rightarrow B \]
\[ \forall t \in T: [A_V(t) = \exists t_1, t_2 \in T: t_1 \leq t < t_2 \land \langle (sp_{tA}(t_1)) \rangle \land \langle (sp_{tA}(t_2)) \rangle \land \]
\[ \forall t_3 \in T: t_1 \leq t_3 < t_2 \Rightarrow sp_{tA}(t_3) \land (\neg sp_{tA}(t_3))]. \]

The first instance of an interval function is defined as follows:

\[ A_V(t, 1) : T \times I^+ \rightarrow B \]
\[ \forall t \in T: [A_V(t, 1) = A_V(t) \land (\exists t_1 \in T: t_1 \leq t) \land \]
\[ (\forall t_2 \in T: t_2 < t_1 \Rightarrow \neg A_V(t_2)) \land (\forall t_3 \in T: t_1 \leq t_3 \leq t \Rightarrow A_V(t_3))]. \]

The ith instance of an index function is defined as follows:

\[ A_V(t, i) : T \times I^+ \rightarrow B \]
\[ \forall i \in I^+: i > 1 \Rightarrow \]
\[ \forall t \in T: [A_V(t, i) = A_V(t) \land (\exists t_1, t_2, t_3 \in T: t_1 < t_2 < t_3 < t) \land \]
\[ \forall t_4 \in T: t_1 \leq t_4 < t_2 \Rightarrow A_V(t_4, i-1) \land \forall t_5 \in T: t_2 < t_5 < t_3 \Rightarrow \neg A_V(t_5) \land \]
\[ \forall t_6 \in T: t_3 \leq t_6 < t \Rightarrow A_V(t_6)]. \]
In the time domain, the start and finish events are defined as temporal markers, respectively, the *start time* and the *finish time*. The definition of the start and finish times follows directly from the time domain definition of the point function. They are defined in terms of time point constants of the form \( t_{\uparrow A}^i \) and \( t_{\downarrow A}^i \) at which the respective events \( \uparrow A_T(t,i) \) and \( \downarrow A_T(t,i) \) have occurred for the \( i \)th time. The *start time* refers to the time point at which the start event of an action occurs:

\[
\uparrow A_T(t,i) : T \times I^+ \rightarrow B
\]

\[
\forall t \in T : \forall i \in I^+ : [\uparrow A_T(t,i) \iff t = t_{\uparrow A}^i] \quad t_{\uparrow A}^i \in T_{\uparrow A} = \{ t_{\uparrow A}^1, t_{\uparrow A}^2, \ldots \}.
\]

The *finish time* refers to the time point at which the finish event of an action occurs:

\[
\downarrow A_T(t,i) : T \times I^+ \rightarrow B
\]

\[
\forall t \in T : \forall i \in I^+ : [\downarrow A_T(t,i) \iff t = t_{\downarrow A}^i] \quad t_{\downarrow A}^i \in T_{\downarrow A} = \{ t_{\downarrow A}^1, t_{\downarrow A}^2, \ldots \}.
\]

In the time domain, the interval function \( A_T(t,i) \) is defined, as follows, in terms of the start and finish times of an action:

\[
A_T(t,i) : T \times I^+ \rightarrow B
\]

\[
\forall t \in T : \forall i \in I^+ : [A_T(t,i) \iff t_{\uparrow A}^i \leq t < t_{\downarrow A}^i].
\]

Similar to the point function definition, the value and time domain definitions of an interval function are alternative forms for describing the execution of the same action:

\[
\forall t \in T : \forall i \in I^+ : [A_V(t,i) \iff A_T(t,i)].
\]

The *execution time* of an action is the duration of the interval during which the action is executed.

In the following we state the two properties that are associated with the interval function:

- **duration property** – the finish event of an action cannot precede the start event of the action:

\[
\forall t, t' \in T : \forall i \in I^+ : [\downarrow A(t,i) \Rightarrow \uparrow A(t',i) \land t > t'].
\]

If the start and finish events, occur at the same time point, implying a durationless action, then the action is modelled by a point function.

- **ordering property** – two instances of the same action cannot be executed at the same time:

\[
\forall t, t' \in T : \forall i, j \in I^+ : [\downarrow A(t,i) \land \uparrow A(t',j) \land i < j \Rightarrow t < t'].
\]

**Utility Function**

The specification of time uncertainties for the two functions defined above can be represented by means of a *utility function* \( U(t,i) \), or value function [10]. We are concerned
with the class of utility functions which are typically found in real-time safety-critical systems — *discrete* or *critical utility functions*. In these functions the utility, or usefulness, can only assume maximum and minimum values — hence \( U(t,i) \) is a Boolean function.

The utility function \( U_E(t,i) \) associated with the point function, which models the occurrence of an event, maps the timeline and the instance number of an event into the usefulness of its occurrence. The maximum usefulness in the occurrence of an event is obtained from the time interval established from the following two time attributes:

- *earliest occurring time* (eot) — the first point in time at or after which an event can occur;
- *latest occurring time* (lot) — the last point in time before which the event can occur.

An event either occurs during the interval of time established by these two time points or it is assumed not to have occurred. A utility function representing the usefulness of the occurrence of an event can be defined as follows:

\[
U_E(t,i) : T \times I^+ \to B
\]

\[
\forall t \in T : \forall i \in I^+ : \{ U_E(t,i) \Leftrightarrow (t_{E_{eot}}^i \leq t < t_{E_{lot}}^i) \}
\]

\[
t_{E_{eot}}^i \in T_{E_{eot}} = \{ t_{E_{eot}}^1, t_{E_{eot}}^2, \ldots \}
\]

\[
t_{E_{lot}}^i \in T_{E_{lot}} = \{ t_{E_{lot}}^1, t_{E_{lot}}^2, \ldots \}.
\]

The utility function \( U_A(t,i) \) associated with the interval function, which models the execution of an action, maps the timeline and the instance number of an action into the usefulness of its execution. The maximum usefulness in the execution of an action is established by the two time intervals associated with the utility functions of the start and finish events of the action. The time attributes of the utility function associated with the execution of an action are the following:

- *earliest starting time* (est) — the first point in time at or after which an action can start its execution;
- *latest starting time* (lst) — the last point in time before which an action can start its execution;
- *earliest finishing time* (eft) — the first point in time at or after which an action can finish its execution;
- *latest finishing time* (lf) — the last point in time before which an action can finish its execution.

The time attributes est and lf are also referred as “delay” and “deadline”, respectively. These time attributes are expressed by step utility functions, respectively positive and negative step functions.
A utility function representing the usefulness in the execution of an action is defined as follows, in terms of the utility functions of the start and finish events of an action:

\[ U_{tA}(t', i) : T \times I^+ \rightarrow B \]
\[ \forall t' \in T : \forall i \in I^+ : [U_{tA}(t', i) \Leftrightarrow (i_{A_{t'}}^{t'} \leq t' < i_{A_{i}}^{t'})] \]
\[ i_{A_{t'}}^{t'} \in T_{A_{t'}} = \{i_{A_{t'}}^{1}, i_{A_{t'}}^{2}, ...\} \]
\[ i_{A_{i}}^{t'} \in T_{A_{i}} = \{i_{A_{i}}^{1}, i_{A_{i}}^{2}, ...\} \]

\[ U_{tA}(t'', i) : T \times I^+ \rightarrow B \]
\[ \forall t'' \in T : \forall i \in I^+ : [U_{tA}(t'', i) \Leftrightarrow (i_{A_{t''}}^{t''} \leq t'' < i_{A_{i}}^{t''})] \]
\[ i_{A_{t''}}^{t''} \in T_{A_{t''}} = \{i_{A_{t''}}^{1}, i_{A_{t''}}^{2}, ...\} \]
\[ i_{A_{i}}^{t''} \in T_{A_{i}} = \{i_{A_{i}}^{1}, i_{A_{i}}^{2}, ...\} \]

\[ U_{A}(t, i) : T \times I^+ \rightarrow B \]
\[ \forall t, t', t'' \in T : \forall i \in I^+ : [U_{A}(t, i) \Leftrightarrow U_{tA}(t', i) \land U_{tA}(t'', i)]. \]

3.1.2. E/A Model and Time Structures

One of the characteristics of the E/A model is that it can support both dense and discrete time structures. However, depending on the time structure there are some dissimilarities on how the occurrence of an event is observed. We say that an event is observed when a point on the timeline is associated with its occurrence.

In dense time structures the occurrence and observation of an event is always simultaneous. Thus, for distinct events we associate distinct time points, except for the cases where events occur simultaneously. In discrete time structures, once an event occurs, it is not possible to observe an event itself, only its consequence(s) may be observed. Hence an event cannot be observed between any two points on the timeline; the point we associate with the observation of an event might not be coincident with the actual occurrence of the event.

Depending on the granularity of a discrete time structure, an action might be represented either by the point function or interval function. If we consider, for example, a time structure \( T_2 \) which contains another time structure \( T_1 \), as shown in figure 2, then an action \( A(t, 1) \) when measured in \( T_2 \) has a duration of less than \( \Delta(T_1) \), but when measured in \( T_1 \) it becomes durationless. However, for another instance of the same action \( - A(t, 2) \), we can have a different situation, as shown in figure 2. In summary, an action of fixed duration in a time structure \( (T_2) \), may assume different durations in another time structure \( (T_1) \) with a greater granularity \( - \Delta(T_1) > \Delta(T_2) \), depending on the time points associated with the respective starting and finishing events of the action. (In order to maintain compatibility with the close/open interval being employed, we have opted to observe the occurrence of an event at the time point before its occurrence.)
3.2. Predicate Event/Action Notation (PEA Notation)

The concern in the E/A model was to identify a set of primitive concepts and define a set of primitive functions that formalise these concepts. In order to express the primitive concepts of the E/A model concisely and to facilitate formal analysis of the system behaviour, we introduce the Predicate Event/Action notation (PEA notation). In the following, the primitive functions of the E/A model will be defined as primitive predicates of the PEA notation.

3.2.1. Primitive Predicates of the PEA Notation

Point Predicate

For the value domain definition of the point predicate the notation “E(i)” is introduced to denote the ith instance of an event E. The point predicate is true iff there exists a time point t at which the point function $E_V(t,i)$ holds true:

$$\forall i \in I^+: [E(i) \iff \exists t \in T: E_V(t,i)].$$

For the time domain definition of the point predicate the notation “E(i)@t” is introduced to denote that the ith instance of an event E occurs at a time point t. The event predicate is true iff at the time point t the event E has occurred for the ith time:

$$\forall t \in T: \forall i \in I^+: [E(i)@t \iff E_T(t,i)].$$

Interval Predicate

For the value domain definition of the interval predicate the notation “$A(i)\uparrow A, A_{INV}, \downarrow A$” is introduced to denote the ith execution of action A. The start condition “$\uparrow A$” and finish condition “$\downarrow A$” are transition predicates, and the invariant condition “$A_{INV}$” is a state predicate. An action is executed between “$\uparrow A$” and “$\downarrow A$” while “$A_{INV}$” holds true. The conditions associated with an interval predicate are defined as follows:

$$\forall i \in I^+: [\uparrow A(i) \iff \exists t_{\uparrow A} \in T: \uparrow A_V(t_{\uparrow A}, i)],$$

$$\forall i \in I^+: [A_{INV}(i) \iff \exists t \in T: A_{V^{INV}}(t, i)],$$

and
\[ \forall i \in I^+: [\downarrow A(i) \Leftrightarrow \exists t_{tA} \in T: \downarrow A(t_{tA},i)]. \]

The interval predicate is true iff for all time points between \( t_{tA} \) and \( t_{tA} \) the interval function \( A(t,i) \) holds true. The interval predicate is defined in the value domain as follows:

\[ \forall i \in I^+: [A(i)@t_{tA} \Leftrightarrow A_T(t_{tA},i)]. \]

For the time domain definition of the interval predicate the notation "\( A(i)@t_{tA} \)" is introduced to denote the \( i \)th execution of an action \( A \) between the time points \( t_{tA} \) and \( t_{tA} \).

These time points represent the times at which the events associated with action \( A \) have occurred: the start event at time \( t_{tA} \) and the finish event at time \( t_{tA} \). The two events are respectively defined as follows, in terms of the point function:

\[ \forall t_{tA} \in T: \forall i \in I^+: [A(i)@t_{tA} \Leftrightarrow A_T(t_{tA},i)], \text{ and} \]

\[ \forall t_{tA} \in T: \forall i \in I^+: [\downarrow A(i)@t_{tA} \Leftrightarrow \downarrow A_T(t_{tA},i)]. \]

The interval predicate is true iff for all time points between \( t_{tA} \) and \( t_{tA} \) the interval function \( A_T(t,i) \) holds true. The interval predicate is defined in the time domain as follows:

\[ \forall t_{tA}, t_{tA} \in T: \forall i \in I^+: [A(i)@t_{tA},t_{tA} \Leftrightarrow \forall t \in T: (t_{tA} \leq t < t_{tA} \Rightarrow A_T(t,i))]. \]

**Utility Predicate**

In the following we define the utility predicates that will be employed in the reasoning of time uncertainties.

The utility predicate "\( U_E(i)@\langle t_{col}, t_{lot} \rangle \)" is introduced to denote the time uncertainty associated with the \( i \)th occurrence of event \( E \). The utility predicate is true iff for all time points between \( t_{col} \) and \( t_{lot} \) the utility function \( U_E(t,i) \) holds true. The utility predicate representing the usefulness in the occurrence of an event is represented as:

\[ \forall t_{col}, t_{lot} \in T: \forall i \in I^+: [U_E(i)@\langle t_{col}, t_{lot} \rangle \Leftrightarrow \forall t \in T: (t_{col} \leq t < t_{lot} \Rightarrow U_E(t,i))]. \]

For the utility predicate of an action \( A \), the notation "\( U_A(i)@\langle t_{A_{col}}, t_{A_{lot}} \rangle \)" is introduced to denote the time uncertainty associated with the \( i \)th execution of the action. The utility predicate is true iff for all time points between \( t_{A_{col}} \) and \( t_{A_{lot}} \), the utility function \( U_A(t',i) \) holds true, and for all time points between \( t_{A_{col}} \) and \( t_{A_{lot}} \) the utility function \( U_A(t'',i) \) holds true. The utility predicate representing the usefulness in the execution of an action is represented as:

\[ \forall t_{A_{col}}, t_{A_{lot}} \in T: \forall i \in I^+: [U_A(i)@\langle t_{A_{col}}, t_{A_{lot}} \rangle \Leftrightarrow \forall t' \in T: (t_{A_{col}} \leq t' < t_{A_{lot}} \Rightarrow U_A(t',i)) \land \forall t'' \in T: (t_{A_{col}} \leq t'' < t_{A_{lot}} \Rightarrow U_A(t'',i))]. \]
3.2.2. Operators of the PEA Notation

In order to compose point and interval predicates of the PEA notation, a set of logical operators are defined in both value and time domains. In the following, we define some of the PEA notation operators in term of the functions of the E/A model.

From two standard logical operators, negation ($\neg$) and conjunction ($\land$), other logical operators can be defined, such as disjunction ($\lor$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). As an example, we define, in the value domain, the conjunction of two interval predicates as follows:

$$\forall i, j \in I^+: A(i)(\uparrow A, A_{INV}; \downarrow A) \land B(j)(\uparrow B, B_{INV}; \downarrow B) \overset{def}{=} \exists t \in T. A_V(t, i) \land B_V(t, j).$$

Four additional logical operators were defined over the interval predicates of the PEA notation: choice ($+$), meet ($<$), overlap ($\parallel$) and disjoint ($\searrow$). As an example, we define, in both value and time domains, the meet and overlap operators.

The meet operator ($<$):

$$\forall i, j \in I^+: A(i)(\uparrow A, A_{INV}; \downarrow A) < B(j)(\uparrow B, B_{INV}; \downarrow B) \overset{def}{=} A(i)(\uparrow A, A_{INV}; \downarrow A) \land B(j)(\uparrow B, B_{INV}; \downarrow B) \land A(i) \Rightarrow B(j);$$

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i, j \in I^+: A(i)(\oplus(t_{\uparrow A}, t_{\downarrow A}) < B(j)(\oplus(t_{\uparrow B}, t_{\downarrow B}) \overset{def}{=} A(i)(\oplus(t_{\uparrow A}, t_{\downarrow A}) \land B(j)(\oplus(t_{\uparrow B}, t_{\downarrow B}) \land (t_{\downarrow A} = t_{\uparrow B}).$$

The overlap operator ($\parallel$):

$$\forall i, j \in I^+: A(i)(\uparrow A, A_{INV}; \downarrow A) \parallel B(j)(\uparrow B, B_{INV}; \downarrow B) \overset{def}{=} A(i)(\uparrow A, A_{INV}; \downarrow A) \land B(j)(\uparrow B, B_{INV}; \downarrow B) \lor \uparrow A(i) \Rightarrow \downarrow B(j)(\uparrow B, B_{INV}; \downarrow B) \lor \uparrow B(j) \Rightarrow A(i)(\uparrow A, A_{INV}; \downarrow A);$$

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i, j \in I^+: A(i)(\oplus(t_{\uparrow A}, t_{\downarrow A}) \parallel B(j)(\oplus(t_{\uparrow B}, t_{\downarrow B}) \overset{def}{=} (A(i)(\oplus(t_{\uparrow A}, t_{\downarrow A}) \land B(j)(\oplus(t_{\uparrow B}, t_{\downarrow B}) \land ((t_{\uparrow B} \leq t_{\uparrow A} < t_{\downarrow B}) \lor (t_{\uparrow A} \leq t_{\uparrow B} < t_{\downarrow A})).$$

Apart from the above two sets of operators, we define another special operator that is only applicable to the time domain: the duration operator (or $\delta-$operator) measures the time distance between two time points (associated with the occurrence of events). The duration between the occurrence of the events $E(i)@t_E$ and $F(i)@t_F$ is given by the duration operator, as follows:

$$\delta_{(t_E, t_F)}: T \times T \rightarrow T$$

$$\delta_{(t_E, t_F)} = t_F - t_E$$
The minimum and maximum durations between the occurrence of two events is obtained from their respective utility predicates, by using the following two variations of the duration operator:

\[
\begin{align*}
\delta_{(i,j)}^{\text{min}} : T \times T &\rightarrow T \\
\delta_{(i,j)}^{\text{max}} : T \times T &\rightarrow T \\
\delta_{(i,x,t,x,f)}^{\text{min}} & = t_{1}E_{\text{out}} - t_{1}E_{\text{out}} \\
\delta_{(i,x,t,x,f)}^{\text{max}} & = t_{1}E_{\text{out}} - t_{1}E_{\text{out}}
\end{align*}
\]

If the two time points of the duration operator refer to the time of occurrence of the start and finish events of an action, then the notation \(\delta_{A(i)}\) can be used instead of \(\delta_{(i,x,t,x,f)}\).

3.2.3. The PEA Notation and other Techniques

Of the many possible formal techniques that could be employed in the context of the proposed approach, the PEA notation is compared with three similar techniques: RTL [4], TRIO [11] and VVSL [12]. RTL captures temporal properties of a system in terms of events and actions. To permit the mechanical reasoning of RTL formulae the expressiveness of the logic was restricted. Specifications in RTL are built using an occurrence function, which relate the occurrence of an event to a time point, this makes it difficult to separate the analysis of the value and time domains. TRIO is a first-order temporal logic language for specifying and verifying timing requirements; its proof theory and its executability can be mathematically defined. The language provides operators that allow the truth or falsity of a proposition at particular time instants. TRIO can accommodate either dense or discrete time structures. However, TRIO does not provide mechanisms for modularizing complex specifications, and it is hard to read and understand. VVSL defines operations in terms of temporal predicates over state variables, in addition to the pre-condition and post-condition of VDM, VVSL introduces an inter-condition which imposes constraints during execution (i.e. acts as a state-invariant). However, unlike the PEA notation, VVSL does not permit quantitative timing analysis.

4. Extract of a Case Study: Controlling the Temperature in a Nuclear Reactor

To clarify some concepts introduced so far, an extract of a case study based on a simplified nuclear reactor control system was selected as an example [13]. Specifically, the example will serve to illustrate how the concepts of the E/A model can be incorporated within different classes of formalisms in order to describe system behaviour.

The example involves a system used to control the temperature of a nuclear reactor, the rods of the reactor have to be moved down when the temperature reaches the pre-defined threshold (5000K). The activity of moving down the rods takes 20 time units, and the start of consecutive movements of the rods should be at least 30 time units apart. To simplify the concepts, we make the (strong) assumption that the movement of the rods is not constrained by any physical limitation.
The Case Study from the PEA Notation Perspective

In a very simplified form, the nuclear plant can be defined, in the PEA notation, as the parallel composition of the actions describing the physical process and safety controller.

\[ \forall i \in I^+: \text{Nuclear} \_ \text{Plant}(i) \implies \text{Physical} \_ \text{Process}(i) \parallel \text{Safety} \_ \text{Controller}(i). \]

The physical process is defined by the sequential composition of actions representing the down movement of the rods (\(PDMovRods(i)\)) and no movement of the rods (\(PNDMovRods(i)\)).

\[ \forall i \in I^+: \text{Physical} \_ \text{Process}(i) \iff \]
\[ PNDMovRods(i) < PDMovRods(i) < PDMovRods(i + 1). \]

In order to capture the timing requirements imposed on the actions the \(\delta\) operator can be employed to specify that the execution of \(PDMovRods(i)\) should take exactly 20 units of time, and consecutive executions should be at least 30 units of time apart:

\[ \forall i \in I^+: \delta_{PDMovRods(i)} = 20. \]
\[ \forall i \in I^+: \delta_{\text{min}}^{\text{min}}(r_{\text{PDMovRods}}^{\text{PDMovRods}}, r_{\text{PDMovRods}}^{\text{PDMovRods}}) = 30. \]

The safety controller is defined by sequential composition of actions representing the down movement of the rods (\(CoDMovRods(i)\)), no movement (\(CoNDMovRods(i)\)), and a wait action on the down movement of the rods (\(CoWMovRods(i)\)).

\[ \forall i \in I^+: \text{Safety} \_ \text{Controller}(i) \iff \]
\[ CoNDMovRods(i) < CoDMovRods(i) < CoWMovRods(i) < CoNDMovRods(i + 1). \]

In sections 5.1 and 5.2, we show how the PEA notation specifications can be used as templates for specifying the behaviour in a property-oriented formalism (THL) and an operational formalism (ER nets).

5. Formalisation of the E/A Model

In this section, we illustrate how the E/A model can be formalised by incorporating the primitive concepts into existing formalisms. The general approach adopted is to define the primitive functions in terms of the primitives of the formalisms, and then impose restrictions over these primitives to capture the basic properties of the point and interval functions. This approach to formalisation is applied to the classes of formalisms identified in this paper: property-oriented and operational.
Property-oriented formalisms specify behaviour in terms of the properties that are exhibited by a system. A property-oriented specification is axiomatic, and hence has a conjunctive nature which allows properties to be added or removed (during analysis) without the need to reconstruct the full specification. A property-oriented specification should only state the necessary constraints on system behaviour (no explicit interrelationships are imposed between parts of the specification) to minimise the restrictions imposed on possible implementations. Temporal Logic [14] and Timed History Logic (THL) [15] are examples of property-oriented formalisms. Operational formalisms specify behaviour by constructing an executable model of the system in terms of mathematical structures such as tuples, relations, functions, sets and sequences. In our approach, we are only concerned with those operational formalisms that explicitly specify non-determinism and concurrency. Statecharts [5] and Environment/Relationship nets (ER nets) [16] are examples of such operational formalisms.

5.1. Property-Oriented Formalisms

In the following we present the primitive functions of the E/A model in terms of THL [15]. THL is a logical formalism, based on the notion of system histories. A history $H$ of a system is a function of the form $H: T \rightarrow \Gamma$, where $T$ is a closed time interval representing the operational lifetime of the system and $\Gamma$ the state space of the system (i.e. the cross product of the set of values for all the variables in a state vector $SV$). The set of all “possible” histories of a system are defined as the universal history set $\Gamma H$. For a history $H$ the sequence of values taken by a variable $v_i$ of $SV$ is denoted by the function $H(v_i): T \rightarrow V_{v_i}$.

Specifications are expressed by restricting the set of histories by two sorts of relations. Invariant relations are formulated as predicates built using free value variables for each $v_i$ of $SV$. A history satisfies an invariant relation if and only if the predicate is satisfied at every time point within $T$. History relations are formulated as predicates built using two free time variables $T_0$, $T_1$ and free function variables for each $v_i$ of $SV$ ($T_0$ and $T_1$ should be interpreted as being universally quantified over $T$). A history satisfies a history relation if and only if the predicate holds for every pair of points $T_0 < T_1$ within $T$.

The THL description of an E/A model of a system is obtained by extending the state space $\Gamma$, in terms of the primitive functions. That is, for a system with $q$ point functions and $r$ interval functions the state vector is extended by the following variables:

$$\{ E_1(t), \ldots, E_q(t), \uparrow A_1(t), \ldots, \uparrow A_r(t), \downarrow A_1(t), \downarrow A_r(t), A_1(t), \ldots, A_r(t) \}.$$  

The above functions are of the form $E(t): T \rightarrow B \times I^+$. However, to be consistent with the E/A model the following convention is adopted to introduce the parameter "$i$":

$$\forall H \in \Gamma H: \forall t \in T: \forall i \in I^+: [H.E(t,i) \equiv H.E(t) = (\text{true},i)].$$
Axioms are then introduced into the THL description, to ensure that the uniqueness and ordering properties of point functions and the duration and ordering properties of interval functions are satisfied for all well-defined histories. For example, we say that a history \( H \) is well-defined for a point function \( E \) if and only if:
\[
\forall t, t' \in T: \forall i \in I^+: [H.E(t,i) \land H.E(t',i) \Rightarrow t = t'].
\]

The double bar operator "\( \overline{\circ} \)" of the E/A model is defined in THL as the following history relation:
\[
\forall t \in T: \overline{\circ} (sp(t)) \iff \\
\forall T_1 \in T: (T_1 = t \land sp(T_1) \land \exists t_1 \in T: \forall t_2 \in T: t_1 < t_2 < T_1 \Rightarrow \neg sp(t_2)).
\]

The Case Study from the Property-Oriented Formalism Perspective

The behaviour of the nuclear plant is specified in THL by imposing invariant and history relations over the start and finish conditions of the actions of the physical process and safety controller.

Physical Process

Pr1. The initial condition of the physical process is that \( PNDMovRods \) holds true.
\[
\forall T_1 \in T: PNDMovRods(1)(T_1) \iff (T_1 = 0).
\]

Pr2. The rods must start to move down if and only if the temperature \( PTemp \) is above the threshold (5000K) and 30 time units have elapsed since the previous time the rods started to move down, the execution of the action should take exactly 20 time units.
\[
\forall T_1 \in T: \uparrow PDMovRods(1)(T_1) \iff \\
(PTemp > 5000)(T_1) \land \forall t \in T: t < T_1 \Rightarrow \neg (PTemp > 5000)(t).
\]

Pr3. \( PNDMovRods \) immediately follows the action \( PDMovRods \), as specified by the sequential composition of the actions in the PEA notation description of Physical Process.
\[
\forall i \in I^+: i > 1 \Rightarrow \uparrow PNDMovRods(i) \iff \downarrow PDMovRods(i - 1).
\]
\[
\forall i \in I^+: \downarrow PNDMovRods(i) \iff \uparrow PDMovRods(i).
\]

Safety Controller
In this example, we assume that provided the reactor temperature ($P_{temp}$) is within its anticipated range then it is equal to the thermometer reading ($C_{temp}$).

**Cr1.** The initial condition of the safety controller is that $CoNDMovRods$ holds true.

$$\forall T_1 \in T: CoNDMovRods(1)(T_1) \iff (T_1 = 0).$$

**Cr2.** The safety controller must start to move the rods down at the instant the temperature ($C_{temp}$) rises above the threshold (5000K) and 30 time units have elapsed since the previous time the rods started to move down, the execution of the action $CoDMovRods$ should take exactly 20 time units.

$$\forall T_1 \in T: \uparrow CoDMovRods(1)(T_1) \iff$$

$$(C_{temp} > 5000)(T_1) \land \forall t \in T: t < T_1 \Rightarrow \neg (C_{temp} > 5000)(t).$$

$$\forall T_1 \in T: \forall i \in I^+: i > 1 \Rightarrow \uparrow CoDMovRods(i)(T_1) \iff$$

$$(C_{temp} > 5000)(T_1) \land \exists t_1 \in T: (t_1 \leq T_1 - 30 \land \uparrow CoDMovRods(i-1)(t_1) \land$$

$$\forall t \in T: t_1 + 30 \leq t < T_1 \Rightarrow \neg (C_{temp} > 5000)(t)).$$

$$\forall T_1 \in T: \forall i \in I^+: \downarrow CoDMovRods(i)(T_1 + 20) \Rightarrow \uparrow CoDMovRods(i)(T_1).$$

**Cr3.** $CoWMovRods$ immediately follows action $CoDMovRods$, and is of duration exactly 10 time units.

$$\forall i \in I^+: \uparrow CoWMovRods(i) \iff \downarrow CoDMovRods(i).$$

$$\forall T_1 \in T: \forall i \in I^+: \downarrow CoWMovRods(i)(T_1 + 10) \iff \uparrow CoWMovRods(i)(T_1).$$

**Cr4.** $CoNDMovRods$ must immediately follow $CoWMovRods$ and precede $CoDMovRods$, as specified by the sequential composition of the actions in the PEA notation description of Safety_Controller.

$$\forall i \in I^+: i > 1 \Rightarrow \uparrow CoNDMovRods(i) \iff \downarrow CoWMovRods(i-1).$$

$$\forall i \in I^+: \downarrow CoNDMovRods(i) \iff \uparrow CoDMovRods(i).$$

### 5.2. Operational Formalisms

In the following we present the primitive functions of the E/A model in terms of ER nets [16]. ER nets are high-level Petri nets where tokens carry information (functions associating values to variables), and transitions are augmented with predicates (describing which input tokens can participate in a firing and which possible tokens are produced by the firing). The timing constraints are expressed as relational expressions (predicates) which must be satisfied for a transition to become enabled.

The representation of the E/A model primitive functions in terms of ER nets is straightforward because of the similarity between the firing rules of a transition and the properties.
associated with the occurrence of events. The timing properties defined for both point and interval functions of the E/A model are captured by the timing predicates to be associated with the transitions.

The point function in the E/A model corresponds in ER nets to the firing of a transition. Figure 3(a) shows the representation of the point function in terms of ER nets. The label $E_\nu(t,i)$ represents the system predicate that defines the occurrence of an event. When transition $tr$ fires the token $tok$ is removed from predicate $P1$ and placed in $P2$ with a timestamp $P2.time (= t'_k)$ corresponding to the time at which the transition fired. The instance number of an event is captured in the predicate of $tr$, by increasing the value of indices $i$. Time uncertainties can also be modelled by associating a utility function with the occurrence of the event, once transition $tr$ is enabled, it is allowed to fire only within the time interval established by the timing relation associated with the transition.

![Diagram](image)

$$tr : P1.time + t_{\text{const}} \leq P2.time < P1.time + t_{\text{rel}}$$

(a)

$$tr1 : P1.time + t_{\text{rel}} \leq P2.time < P1.time + t_{\text{rel}}$$

$$tr2 : P2.time + t_{\text{rel}} \leq P3.time < P2.time + t_{\text{rel}}$$

(b)

Figure 3. The ER nets representation of the E/A model:
(a) point function;
(b) interval function.

The representation of the E/A model interval function follows directly from the point function representation. The labels $\uparrow A_\nu(t,i)$ and $\downarrow A_\nu(t,i)$ on transitions $tr1$ and $tr2$ represent, respectively, the start and finish events that are associated with the execution of an action, and place $P2$ denotes the execution of the action. The invariant condition associated with the point function ($A_\nu^{\text{INV}}(t,i)$) is part of the system predicate of transition $tr1$. The violation of $A_\nu^{\text{INV}}(t,i)$ while, for example, an action is being executed, can be represented by a choice on $P2$ between transition $tr2$ and another transition which negates the invariant condition. Figure 3(b) shows the representation of the interval function in terms of ER nets. Time
uncertainties can also be modelled by associating a utility function with the occurrence of each event that defines the execution of an action.

The Case Study from the Operational Formalism Perspective

In order to exemplify the utilization of the E/A model primitive functions as modelling concepts for an operational formalism, in the following, we discuss the ER net model of the movement of the rods to control the temperature of a nuclear reactor. The ER net model is shown in figure 4 and includes models of the physical process and the safety controller. The name and the role of the places of the ER net correspond to the actions specified by the property-oriented formalism, presented in the previous section. The relations associated with the transitions of the ER net are defined in figure 4, and correspond to the conditions of the actions defined in THL. In the ER net model, the place CiTemp contains the last measurement of the reactor temperature which is periodically updated from the simulated values contained in PSimTemp. The subnet containing PDMovRods and PNDMovRods is obtained from the PEA notation formula that defines the sequence of actions that are associated with the rods; the subnet containing CoDMovRods, CoWDMovRods and CoNDMovRods is obtained from the PEA notation formula that defines the sequence of actions that have to be performed by the safety controller in order to meet the requirements imposed on the movement of the rods depending on the temperature of the reactor; the places ActDMovRods and ActNDMovRods represent the outputs of the (logical) actuator that moves the rods. The predicates associated with the transitions T3, T4 and T5 are derived from the start and finish conditions defined for the actions to be executed by the safety controller.

6. Conclusions

This paper presents the E/A model, as a basic model for describing the behaviour of real-time safety-critical systems, which consists of primitive concepts such as events, actions and states. The concepts of the E/A model are related to the flow of time by introducing two primitive functions: point function and interval function. A key feature of the E/A model is its flexibility: it can be incorporated into a wide range of formalisms which are based on either dense or discrete time structures, and distinguish between analysis in the time and value domains. However writing specifications directly in terms of the E/A model would be cumbersome, hence the PEA notation is introduced as a more compact notation to facilitate the formal analysis of system behaviour in terms of the E/A model primitives.

The flexibility of the E/A model is demonstrated by introducing the primitive functions, into a property-oriented formalism (THL) and an operational- formalism (ER nets). The support, for modelling system behaviour, provided by the primitive concepts of the E/A
Legend of the relevant transitions:

T3 = {((CiTemp, CoNDMovRods), (CoDMovRods, ActDMovRods)) | CiTemp.value > 5000}
T4 = {((CoDMovRods, ActNDMovRods), CoWDMovRods) |
  CoWDMovRods.time = CoDMovRods.time + 20}
T5 = {(CoWDMovRods, CoNDMovRods) | CoNDMovRods.time = CoWDMovRods.time + 10}
T7 = {{P1, P2} | P2.time = P1.time + 15}

Figure 4. The ER net model of the nuclear reactor.

model was illustrated by performing the analysis of a simple example from the perspectives of the PEA notation, THL and ER nets.

Acknowledgements

The authors would like to acknowledge the financial support of British Aerospace (DCSC), and the SERC (UK) SCHEMA project.

References


