Symbolic Discrete Event System Specification *

Bernard P. Zeigler and Sungdo Chi

AI-Simulation Group
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

Abstract
Extending discrete event modelling formalisms to facilitate greater symbol manipulation capabilities is important to further their use in intelligent control and design of high autonomy systems. This paper defines an extension to the DEVS formalism that facilitates symbolic expression of event times by extending the time base from the real numbers to the field of linear polynomials over the reals. A simulation algorithm is developed to generate the branching trajectories resulting from the underlying non-determinism. The extended formalism offers a convenient means to conduct multiple, simultaneous explorations of model behaviors. Examples of application are given with concentration on fault model analysis.

1 Introduction
Discrete event modelling is finding ever more application to analysis and design of complex manufacturing, communication, and computer systems among others [6]. Powerful languages and workstations been developed for describing such models for computer simulation [7]. Until now, the time base of discrete event models has been assumed to be numerical. To enhance the capability of discrete event models we extend the Discrete Event System Specification (DEVS) formalism[3] to be defined on a symbolic time base. In conventional DEVS, and indeed, in most system formalisms [27,28,29], the time base and its operations and relations are performed over the field of real numbers. In symbolic DEVS, this field is extended to the linear polynomials over the reals. For example, the expression “5 + 2 × travel.time + delay.time” is a legal time value and might represent the time taken to make a round trip journey. When travel.time and delay.time are assigned numerical values, this expression evaluates to a real number. Thus, precise knowledge of any event time can be readily incorporated by setting its symbol to a real value. Therefore, symbolic DEVS is truly an extension of conventional DEVS and reduces to the latter when all times are real valued. This contrasts with qualitative representations of differential equation models (e.g., [19]) that move to a coarser grained formalism in which exact timing cannot be recovered with acquisition of more information.

A network or coupled model[5] of symbolic DEVS components is non-deterministic since the minimum of a set of next event times is not unique as in a conventional DEVS of the same kind. Thus trajectories split into tree-like structures. Each minimal element in a set of next event times represents a possible state transition; in making the choice of such an element, we generate a set of constraints asserting that it is less than all other minimal elements. This constraint applies to the trajectory branch just initiated. Likewise constraints are accumulated and inferred to constrain minimal next event times in subsequent branchings. For example, asserting “t_1 < t_2” at one branch, and “t_2 < t_3” at a subordinate one, requires also that “t_1 < t_3” in the latter’s subtree. Since symbolic times are linear expressions, we adopt techniques from linear programming to efficiently manage and infer the constraint sets[25,26].

One major application of symbolic DEVS is to do the causal propagation required for fault diagnosis. Conventional DEVS models can do such propagation but require fixing the event times (to particular real numbers) to do so. By allowing the times to be ex-
pressed symbolically, we generate a family of trajectories that represent the possible sequences of events resulting in a particular breakdown. Such a family will be employed to trace a detected anomaly back to the components that might have caused it. The symbolic extension allows for both the situations where timings are unknown, or are known, but can vary. This approach to model-based diagnosis supersedes others [15,16,17] in that it intrinsically represents timing effects. Dynamic models are necessary in task execution application where effects of an anomaly occurring in some task are not apparent until they have propagated some time later, in some transformed manner, to a detectable error in another task.

Besides being applicable to fault diagnosis, symbolic DEVS may have a variety of applications where there is a need for efficient generation of family of trajectories characterizing all scenarios of a given model. For illustration, we shall show how the rules for strikes, balls, and hits in baseball can be given such an animation.

2 Background

There are three major formalisms, or short-hands for dynamic system specification. Differential and difference equations employed to describe continuous systems have a long history of development whose mathematical formalization came well before the advent of the computer. Discrete event modelling is of more recent vintage and is finding ever more application to analysis and design of complex manufacturing, robotics, communication, and computer systems among others [6,7]. In contrast to continuous systems simulation, discrete event simulations were made possible by, and evolved with, the growing computational power of computers. Formalization of the models that underly discrete event simulations is therefore a relatively new concern [2,12]. Formalisms that can capture both discrete and continuous behavior are also of increasing importance [9,10,11].

The classical AI knowledge representation schemes, such as rules, semantic nets, and frames, can be viewed as essentially symbolic formalisms for static models. They provide organizations for large bodies of facts with mechanisms for inferencing, access path development, and comparison based on symbol manipulation. New directions in AI, for example, qualitative modelling and planning, are finding it necessary to take time into account [13,21].

On the other hand, while differential/difference equation formalisms are numeric in character, this is not necessarily true of more recent dynamic systems formalisms. Discrete event and automata models for example, usually include state representations of a symbolic nature. Antsaklis et al. [1] refer to discrete event models that do not include a time base explicitly [14] as "logical", thereby emphasizing their symbolic nature. Narain [11] has developed a logic-based formalism intended to facilitate reasoning about the behavior of combined discrete/continuous models.

Qualitative physics has done much to bring qualitative formalisms for dynamic model specification to attention. AI researchers in qualitative physics envision an ideal formalism which captures commonsense knowledge of dynamic systems without the use of differential equations [18,19,17,20]. The ability of such models to deal with incomplete specification is emphasized, although the cost is heavy ambiguity and trajectory branching.

Traditional discrete event formalisms incorporate uncertainty in the form of stochastic processes. Modern views recognize kinds of uncertainty that cannot be represented in a probabilistic manner such as ambiguity, lack of relevant factual knowledge, and deliberate use of linguistic imprecision. Fuzzy set theory is the predominat framework for representing such uncertainty (although, it is not the only one). Incorporating fuzzy set mechanisms into models is drawing increasing attention [8,24].

The discrete event-based control methodology [4,5] provides an approach to process and device control which is especially tuned to the needs of high-autonomy applications. It provides a relatively simple and robust real-time control layer that can be linked to higher level symbolic reasoning layers with appropriate structure and behavior preserving morphisms [22]. A crucial requirement in discrete event-based control is the coupling of task execution with an error recovery capability. Such capability is based on diagnosis of faults based on detected anomalous operation. The development of symbolic DEVS to be discussed was motivated by the need for a modelling formalism to support such diagnostic requirements.

3 DEVS and Its Extensions

In this section, we review the concepts of conventional DEVS and proceed to define the extensions to symbolic time and non-determinism.
3.1 Review of DEVS Concepts

A DEVS (Discrete Event System Specification) is a structure:

\[ M = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau) \]

where

- \( X \) is the set of external (input) event types
- \( S \) is the sequential state set
- \( Y \) is the output set
- \( \delta_{\text{int}} : S \rightarrow S \), the internal transition function
- \( \delta_{\text{ext}} : Q \times S \rightarrow S \), the external transition function
- \( Q \) is the total state set
- \( \lambda : S \rightarrow Y \), the output function
- \( \tau(s) \) is the time advance function, where
- \( \lambda \) is the non-negative reals with \( \infty \) adjoined.

(Note: \( M \) is defined only as far as needed in the following discussion.)

3.3 Symbolic Time and NonDeterminism

We extend the basic definition in two ways. First, we extend the underlying time base to \( \mathcal{LP} \), the field of linear polynomials over the reals. This requires redefinition of the time advance function:

\[ \tau : S \rightarrow \mathcal{LP}_{0,\infty}^+ \]

The time advance values are now linear polynomial expressions such as \( 1, 2, s, t, 2s, 3t, t + s, 2t - 3s + 1 \). Such expressions must evaluate to non-negative real numbers (including \( \infty \)) when the basic symbols, such as \( s \) and \( t \) are replaced by non-negative real numbers or \( \infty \). The resulting structure is called a symbolic DEVS. We will see that the field operations of arithmetic on the linear polynomials elegantly justify the required manipulation of time values.

An example of a symbolic DEVS is the above propagation model with the \( \sigma \) component of the state being allowed to take on symbolic values such as \( \text{time-to-breakdown} \).

In the second extension, we allow the time advance function to be set-valued. This changes the definition of the latter to:

\[ \tau \] is a finite, non-empty set of linear polynomials.

We define a partial order relative to \( 0 \):

\[ t \prec 0 \iff \text{all coefficients and the constant in } t \text{ are negative} \]

\[ t \succ 0 \iff \text{all coefficients and the constant in } t \text{ are positive} \]

\[ t \equiv 0 \iff -t \equiv t \equiv 0 \]

and extend the order to \( \mathcal{LP}_{0,\infty}^+ \):

\[ t \prec t' \iff (t - t') < 0 \]

\[ t = t' \iff (t - t') = 0 \]

\[ t \succ t' \iff (t - t') > 0 \]

\[ t \equiv t' \iff t - t' \equiv 0 \]
Figure 1 illustrates the partial order that results on the set \( T = \{ t, 2t, 3t, s, 2s, 3s, s + t \} \). We require that \( t\alpha(s) \) be pairwise incomparable, i.e.,

\[
\forall t, t' \in t\alpha(s) (t \neq t') \Rightarrow (t \parallel t')
\]

As will be seen, the set \( t\alpha(s) \) will be derived by extracting the subset of minimals from the set of next event times for components in a simulation. Here \( \text{minimals}(T) = \{ t \mid t \text{ is minimal in } T \} \text{ and } t \text{ is minimal in } T \Rightarrow \forall t' \in t\alpha(s) (t \neq t') \Rightarrow \neg(t < t) \).

In Figure 1a), \( \text{minimals}(T) = \{ t, s \} \). Extracting \( \{ t, s \} \) from \( T \), the remaining set of times has minimals \( \{ 2t, 2s, s + t \} \). We shall later be including the effect of constraints on our choice of next event times. For example, the constraint \( s < t \) results in the partial order in Figure 1b). Now \( \text{minimals}(T) = \{ s \} \). Extracting \( \{ s \} \) from \( T \), the remaining set has minimals \( \{ t, 2s \} \).

Note that the minimals of a set are all mutually incomparable.

We have to extend the internal transition function to accord with the fact that a choice is now available for the time advance value:

\[
\delta_{\text{int}} : S \times \mathcal{L}T_{0,\infty}^{+} \rightarrow S
\]

where now \( \delta_{\text{int}}(s, t) \) is the next state to be entered from state \( s \) after waiting for a time \( t \) chosen from \( t\alpha(s) \). Of course, \( \delta_{\text{int}}(s, t) \) need be defined only for those \( t \in t\alpha(s) \).

The resulting structure is called a non-deterministic symbolic DEVS, an example of which is shown in Figure 2a). In state \( s_0 \) it has a time advance set consisting of two elements, \( t_1 \) and \( t_2 \). The internal transition corresponding to \( t_1 \) is \( s_0 \to s_1 \); that corresponding to \( t_2 \) is to \( s_0 \to s_2 \). The latter state has a unique time advance, \( t_1 - t_2 \), a linear expression over symbols \( t_1 \) and \( t_2 \). For the latter expression to evaluate to a positive real, we must have \( t_2 < t_1 \). We shall see later that the simulation process imposes this constraint on the trajectory continuing from \( s_2 \) after the choice of \( t_2 \) at the branch at \( s_0 \).

The autonomous behavior of a conventional DEVS is its behavior when no external events ever affect it. This behavior is the only one of interest when the DEVS is closed (input free) as is always the case for the highest level of a hierarchical simulation model including its experimental frame[3]. The autonomous behavior is readily characterized. Starting from an initial state \( s \) at time \( t_i \), the DEVS generates a piecewise constant state trajectory over the real line extending to the right of \( t_i \). Events are the transitions from a constant state to its successor. The inter-event times are determined by \( t\alpha \) and the state transitions themselves by \( \delta_{\text{int}} \).

An autonomous state trajectory for a non-deterministic symbolic DEVS is generated in the same manner as before except that now the times to next event are chosen non-deterministically from those offered by the current state. In other words, starting in state \( s \) at time \( t_i \), we chose a time advance \( t_s \) from \( t\alpha(s) \). At time \( t_i + t_s \), the model transitions to state
and the process continues with choice of a time advance from $t(s')$, and so on. The field axioms on $\mathbb{L}P$ assure that all arithmetic operations are consistently managed.

The time trajectories corresponding to the choices available for the DEVS in Figure 2a) are shown in Figure 2b).

Basic models may be coupled in the DEVS formalism to form a multicomponent model which is defined by the structure:

$$DN = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, Select \rangle.$$  

where

- $D$: is a set of component names;
- and for each $i \in D$,
- $M_i$: is a component basic model
- $I_i$: is a set, the influences of $i$
- and for each $j \in I_i$,
- $Z_{i,j}$: is a function, the i-to-j output translation
- and
- $Select$: is a function, the tie-breaking selector.

A multicomponent, or coupled model, tells how to couple (connect) several component models together to form a new model. If each component in a multicomponent model is a symbolic DEVS the structure is called a symbolic multicomponent model. We shall restrict discussion to the case of symbolic deterministic components.

The coupled model shown in Figure 3a) containing two fault propagation model components of the kind defined earlier. Its multicomponent specification is:

$$DN = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, Select \rangle.$$  

where

- $D = \{A, B\}$;
- $M_A, M_B$ are copies of the fault propagation DEVS defined earlier
- $I_A = \{B\}, I_B = \emptyset$
- $Z_{A,B}$ is an identity mapping
- $Select(D) = B$ (if A and B are simultaneously scheduled, let B occur first; A will then occur immediately afterwards)

Note that since there is an internal coupling from A to B, a breakdown at A is propagated as an external event to B.

### 3.3.1 Expressing a Symbolic Multicomponent Model as a Non-deterministic Symbolic DEVS

A conventional multicomponent model $DN$ can be expressed as an equivalent basic model in the DEVS formalism[3]. Such a basic model can itself be employed in a larger multicomponent model. This shows that the formalism is closed under coupling as required for hierarchical model construction. Expressing a multicomponent model $DN$ as an equivalent basic model captures the means by which the components interact to yield the overall behavior. In like manner a symbolic multicomponent model can be expressed as an equivalent symbolic DEVS basic model. The following shows how this is done:

Let the state of the basic DEVS model $M$, representing the overall system, be the vector of total states $s = (\ldots(s_i, e_i))\ldots$ of the components $i \in D$. This means that at any event time $t$, each component, $i$ is in a state $s_i$ and has been there for an elapsed time $e_i$. The time advance in state $s_i$ is $t(a_i(s_i))$ so that component $i$ is scheduled for an internal event at time $t + (t(a_i(s_i)) - e_i)$. In a conventional multicomponent model, the next event in the system will occur at a time which is the minimum of these scheduled times, namely, at time $t + \sigma$, where $\sigma$ is the minimum of the
residual times, \( \sigma_i = ta_i(s_i) - \epsilon_i \), over the components \( i \in D \). Since all times are linear polynomials all addition and subtraction operations referred to are valid. However, in a symbolic multicomponent model, the \( \sigma_i \), being polynomials are not necessarily comparable so a unique minimum of the \( \sigma_i \)'s may not exist. Thus we set:

\[
    ta(s) = \text{minimals}(\{\sigma_i | i \in D\})
\]

which represents the set of earliest next event times.

The definition accords with that required for a symbolic non-deterministic basic DEVS.

Now, each \( \sigma_i \in ta(s) \) determines a possible state transition. For each such \( \sigma_i \) let \( IMM = \{ j \in D | \sigma_j = \sigma_i \} \) be the set of imminent components having the same next event time as \( i \). Let \( i^* = \text{Select}(IMM) \) be the selected, or imminent, component. At time \( t + \sigma_i \), just before \( i^* \) changes state, it computes its output \( y^* = \lambda_{i^*}(y_i^*) \). This output is sent to each of the influences of \( i^* \) in the form of a translated input: for influencee \( j \), the input, \( z_{i^*,j} \), is \( Z_{i^*,j}(y_i^*) \). The elapsed time at any component \( i \) at time \( t + \sigma_i \) is just \( \epsilon_i + \sigma_i \). An influencee \( j \) responds to the external event generated by \( i^* \) by applying its external transition function to obtain the next state \( \delta_{ext}(si, ej, x_{i,j}) \) and to reset its elapsed time to 0. Other components not in the influencee set are unaffected by the activation of \( i^* \) except that their elapsed time clock is incremented by \( \sigma \) as just described. Finally, the imminent component \( i^* \) executes its internal transition by going to state \( s_{i^*} = \delta_{int}(s_i^*) \) and resetting its elapsed time to 0.

The above describes how \( M \)'s internal transition function, \( \delta_{int}(s, t) \) is defined. For each \( t = \sigma_i \in ta(s) \), it maps the vector of state pairs \( s = (\ldots(s_i, \epsilon_i)\ldots) \) to a new vector \( (\ldots(s_i, \epsilon_i, t)\ldots) \) computed according to the above recipe. We can similarly follow the effect of an external input event arriving to some of the components and thereby derive the external transition function of the basic model. But we will need only the internal transition behavior in the current discussion.

4 Symbolic Simulation

Often, the basic equivalent of a multicomponent DEVS is not derived by formal manipulation but is investigated through simulation. In the case of symbolic multicomponent models an appropriate simulation algorithm has to be developed which handles the non-determinism arising due the symbolic nature of the underlying time set. Let \( M = (S, \delta_{int}, ta) \) be a non-deterministic symbolic DEVS. The algorithm generates a branching tree structure of nodes, each representing a generated state of the \( M \). Since such an \( M \) represents the basic equivalent of a symbolic multicomponent model, the algorithm is easily extended to the latter case.

Each node is an object containing slots for:

- the time of this event, \( t \) (a linear polynomial)
- the chosen time advance, \( \tau \) (a non-negative linear polynomial)
- the accumulated constraint set, \( C \)
- the prior state of the DEVS, \( s \)
- the next state of the DEVS, \( st \)

The symbolic simulation algorithm is:

Initialize model \( M \) to desired initial state, \( s_0 \) at time, \( t_0 \). Create the root node:

- time = \( t_0 \)
- chosen time advance = unspecified
- constraint set = \( \emptyset \)
- prior state = unspecified
- next state = \( s_0 \)

(Note: the constraint set may also be initialized to a non-empty set if desired. This could occur if the initial state is a terminal state of some previous run or if some a priori constraints need to be established for the simulation.)

Starting at the root node, skip step 1 in the following cycle, and continue with step 2 with \( st \) taken as \( s_0 \).

From the current node with:

- time = \( t \)
- chosen time advance = \( \tau \)
- constraint set = \( C \)
- prior state = \( s \)
- next state = unspecified

Do the following:

1. Given \( s \) and \( \tau \) simulate to get \( st = \delta_{int}(s, \tau) \)
2. Given \( st \), obtain \( ta(st) \)
3. If \( ta(st) = \{ \infty \} \) then leave this node, else:
   - For each \( \tau_i \in ta(st) \)
   - If \( C \cap \text{reduce}(C \cup \{ \tau_i \mid \tau_i < \tau \}) \neq \emptyset \)
     - then create a node with
     - time = \( t + \tau_i \)
time advance = \tau
constraint set = \mathcal{C}
next state = s'

4. Repeat the simulation from each newly created node

Here reduce(C) tests whether a set of constraints, C is consistent; if so, it returns a subset containing no redundant constraints, otherwise it returns \emptyset [26].

The abstract simulator implementation in DEVS-Scheme [5] has been expanded to execute such a simulation for one-level coupled models with symbolic atomic-model components. In steps 1 and 2, it employs the existing simulation apparatus to set the initial state of the model to s, i.e., each component model is initialized to its given state. It then performs a one-step simulation to obtain the next state, s' and the next time advance set, ta(s'). Step 3 actually extracts a subset of the latter set -- the minimal relative the accumulated constraints \mathcal{C}. In other words, for a particular member, \tau_j \in ta(s) it checks all of the inequalities \{\tau_i < \tau_j\}, j \neq i for consistency with \mathcal{C}. This tests whether \tau_j can be made to be the smallest element in ta(s) without violating the constraints, \mathcal{C}. If constraint violation occurs then the assertion that \tau_j can be made the smallest element is untenable. In this case, no branch is created for \tau_j. Otherwise, \tau_j can safely be forced to be the minimum of ta(s) and a branch is created for it. Along this branch the constraint set must be updated to reflect the new set of assertions just made. Reduce does this while ensuring that the resulting set of constraints is irredundant.

4.1 Example: Symbolic Simulation of Simple Fault Propagation

The foregoing simulation algorithm is illustrated with its operation on the coupled model of Figure 3a). The tree structure generated from the initial state in which both components are functioning normally is shown in Figure 3b). There are four states that are reachable from the initial state:

- A, B = A and B both N
- A, B = A is normal, B is L1
- A, B = A and B are both at L1
- A, B = A is at L1, B is at L2

Starting from the node0 in which A and B are both in the normal operating state at time 0, we obtain the time advance set: \{t_A, t_B\} (where the respective sigmas has been set to these values). This means that components A and B are scheduled to be struck by a breakdown event at times t_A and t_B, respectively. Recall that in a symbolic DEVS, these times can be basic symbols such as 'time.to.breakdown.A' and 'time.to.breakdown.B', or indeed arbitrary linear expressions over basic symbols, in addition to the numerical values that can be assumed in a conventional DEVS. Since there is currently an empty constraint set, and assuming that both time advances are basic symbols, it is feasible for both symbols to be asserted to be minimum. Thus there are two children of the root node corresponding to the two possibilities in which the A breakdown occurs first or the B breakdown occurs first.

Node1 represents the case that the A event occurs first. Naturally, for Node1 the constraint, \{t_A < t_B\} is added to its constraint set. The time is set to 0 + t_A, i.e., t_A. With the prior state represented by A, B, simulation results in the new state, A, B in which A has failed to level L1, and in doing so, has sent an output to B causing it to fail to level L1 as well. Since both components are now in passive states, the time advance set = minimal({\infty, \infty}) = {\infty} and the trajectory ends here.

Node2 represents the case that the B event occurs first. Here the new state reached by simulation is A, B, in which B has failed to level L1 but has not affected A since there is no internal coupling to do such propagation. In this state, B is passive and A still has its original time to next event, t_A but its elapsed time has been increased to t_B. Thus A's remaining time to next event, \sigma_A is t_A - t_B. The time advance set for this state has only one element, minimal({\infty, t_A - t_B}) = {t_A - t_B}.

Node3 represents the one successor state of node2. The time slot for this node is t_B + (t_A - t_B) = t_A. In the new state, denoted by A, B, the breakdown event has occurred in A sending it to level L1, and this propagated as well to B, sending it from level L1 to level L2. Both components are now in passive states and the trajectory ends here.

Notice that the node graph of Figure 3b) is isomorphic to the state graph of the internal transitions in Figure 2a). Indeed, the latter is seen to be the state

---

1Actually, the choice is between the two symbols t_A and t_B; the case in which both would be set equal would be uniquely resolved by the Select function so would result in one component going first anyway--the result can be accounted for as a limiting case of the already considered trajectories.
(a) Model Structure of Baseball

Pitcher

Batter

(b) State Transition Graphs

Where a: approaching-time, w: waiting-time, c: chance-time, s: swing-time.

Note: Branch is labelled by the time chosen to be set minimum and constraints of each node are shown in an associated box.

(c) Generated Tree of Trajectories

(d) Timing Relations

Figure 4: Symbolic Simulation Example: Baseball
graph of the basic equivalent of Figure 3a) under the obvious identification of states and times. The trajectories in Figure 2b) therefore depict the paths from the root to leaves generated by the symbolic simulation. Notice that both trajectories go into passive states at time $t_A$, the time of the scheduled breakdown in $A$. In one trajectory, $A$ goes first and causes $B$ to breakdown earlier than scheduled. In the second trajectory, $B$ goes first, but does not affect the scheduling of the breakdown in $A$. This demonstrates the workings of the underlying arithmetic operations axiomatized in the field of linear polynomials. Due to the axioms, time points on different trajectories that have the same value do consistently represent the same time. Of course, other model specifications are possible that would lead to trajectories that may not have such common time points.

4.2 Example: Symbolic Simulation of Baseball

For another illustration of symbolic simulation, consider some timing relations between pitcher and batter in baseball. The ball departs from the pitcher and approaches the home plate providing an opportunity for the batter to swing. In not hit, it reaches the catcher. On the other hand, the batter waits for the ball to reach a good position and then swings to hit the ball.

Figure 4a) is a coupled model, and Figure 4b) shows the symbolic DEVS state transition graphs of its PITCHER and BATTER components. When the PITCHER receives the start command it changes the state from passive to approaching. After 'approaching time' has elapsed, it outputs approached to the UMPIRE and changes its state to chance during 'chance time'. Then it goes to passive state after sending output arrived to the UMPIRE. The BATTER has basically same state transition but different states and times. Symbolic simulation generates the tree of trajectories in Figure 4c) which are illustrated in Figure 4d). Note that the leaves of Figure 4c) represent all possible outcomes of the model together with the timing relationships necessary to generate them.

4.3 Development of Models for Fault Diagnosis

Our approach to intelligent fault diagnosis is compatible with Raymond Reiter's a general theory of diagnosis from first principles [15] based on earlier work [16]. Intuitively, a diagnosis is a conjecture that a set of components of a system are faulty and the rest are normal. Reiter invokes a principle of parsimony to restrict attention to the subfamily of minimal diagnoses. However, Zeigler [23] shows that non-minimal diagnoses may exist where information is available to restrict the class of possible faulty versions of components and that the diagnosis process is severely impaired by not taking such non-minimal diagnoses into account. Thus the full family of diagnoses, although potentially much more computationally demanding than the minimal diagnoses, should be taken as the basis for trouble shooting. The following approach to fault diagnosis generates such a full family of diagnoses.

![Figure 5: Component Fault Propagation Model](image)

Figure 5: Component Fault Propagation Model

![Figure 6: Multicomponent Model for Event-based Execution of An Elementary Laboratory Task](image)

Figure 6: Multicomponent Model for Event-based Execution of An Elementary Laboratory Task

Figure 5 shows how components of a multicomponent model are modelled for fault propagation analysis. Each component is abstracted into a unit consisting of one, or more, fault generators and a fault model. An anomalous condition is sent by a fault generator to the fault model which changes its state accordingly. For example, Figure 6 shows a multicomponents model for event-based execution of an elementary laboratory task. The MOVE component represents robot travel...
from on site to another. One associated fault generator generates the occurrence of an obstacle in the way. The fault model abstraction for MOVE should represent the effect of the obstacle on its state—perhaps to lengthen its time to reach its destination. The effect of the fault on other components is modeled by passing a world status description along as output from a component to its influences. For example, if the obstacle causes the MOVE component to arrive at its destination, but slightly askew, this departure from normal state will be passed on in the world status description to subsequent fault model components. Fault models include transitions to dead states that represent detectable anomalous conditions. An anomalous condition is detectable at a fault model component if there is a sensor affected by this condition in the corresponding operational model and a time window in which the sensor is expected to respond. For example, a slight delay in arrival at destination and a skewed orientation may not be detectable to the MOVE component itself. However, the skewed orientation may cause the robot to select a wrong sized syringe and this might be detected subsequently at the FILL model due to a mismatch in filling time.

Fault generators and fault models have symbolic times so that a symbolic simulation of a network of such components will generate all state trajectories consistent with a set of imposed constraints. Four types of constraints on faults can be considered:

1. State dependent faults: Once a model state is entered, a fault, F, should be activated; conversely, when a state has been left, a fault should be deactivated.

2. Causally dependent faults: once a fault, F0, is activated, another fault, F1, should be activated.

3. Mutually exclusive faults: once a fault, F0, is activated, another fault, F2, cannot be activated.

4. Timing related faults: a fault, F0, is to be activated before another fault, F2 is activated.

Activation and de-activation of a fault generator is done by appropriately coupling the output ports of its influencing components to its activate/de-activate input ports. Timing constraints on fault generation are supplied as initial elements of constraint sets that are built up along the trajectories.

The diagnostic engine is activated by receiving a symptom (the detected anomaly). It starts the symbolic simulation to generate all trajectories and mark those that reach states exhibiting the detected symptom. Alternatively, for small models such trajectories can be pre-compiled off-line for faster on-line use. Each such trajectory represents a hypothetical sequence of fault injections and activations that could have resulted in the observed symptom. The diagnoser investigates each trajectory by testing components in which the faults were injected for presence of such faults. Additional sensory probes may be used for this purpose, requiring a probe choice strategy. A trajectory for which all injected faults are found to exist is confirmed as an acceptable explanation of how the anomaly was caused.

The example in Figure 3, though rudimentary, can serve to illustrate these ideas. First note that in this example, no separation has been made between fault generators and the fault model itself. This has been done only to make it easier to explain the symbolic DEVS formalism and symbolic simulation. Further, let us assume that fault levels L1 and L2 can be directly detected as anomalies. Then the simulation in Figure 3a yields the following table:

<table>
<thead>
<tr>
<th>Detected Anomaly</th>
<th>Fault Trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>A in L1</td>
<td>Breakdown in A</td>
</tr>
<tr>
<td>A in L2</td>
<td>None</td>
</tr>
<tr>
<td>B in L1</td>
<td>Breakdown in B</td>
</tr>
<tr>
<td></td>
<td>or Breakdown in A</td>
</tr>
<tr>
<td>B in L2</td>
<td>Breakdown in B followed by Breakdown in A</td>
</tr>
</tbody>
</table>

For example, the anomaly corresponding to component A having entered phase L1 might be explained by the single fault trajectory in which a breakdown occurs in A (this is the trajectory from A, B to \( \bar{A}, \bar{B} \)).2 There are no trajectories setting A to L2 so that if such an anomaly is detected in reality (starting from the all normal state) than either the measurement device is to be suspected or the model is wrong. The entering of B into phase L1 can be explained by two trajectories, while its entry into L2 can only be a consequence of the trajectory from A, B to \( \bar{A}, \bar{B} \). The table is an example of the compiled form of knowledge directly usable for fault location. If a fault is detected, a diagnoser can look up its possible diagnoses in the table and verify whether faults have have actually occurred in the indicated components.

2 Notice that the trajectory in which B breaks first, followed by A (A, B to \( \bar{A}, \bar{B} \)) will also send A to L1. Such redundant trajectories must be eliminated with rules akin to, but less restrictive than, the minimality assumption mentioned above.
5 Conclusions and Future Research

An extended discrete event specification formalisms, symbolic DEVS, has been defined that extends the time base from the reals to the linear polynomials over the reals. This permits next event times to be expressed symbolically, although precise knowledge of any event time can be readily incorporated by setting its symbol to a real value. Therefore symbolic DEVS is truly an extension of conventional DEVS and reduces to the latter when all times are real valued. A network of symbolic DEVS components is non-deterministic since the minimum of a set of next event times is not unique as in a conventional DEVS of the same kind. A simulation algorithm was developed to generate the branching trajectories resulting from the underlying non-determinism. The extended formalism offers a convenient means to conduct multiple, simultaneous explorations of model behaviors. Since symbolic times are linear expressions, techniques from linear programming were adapted to efficiently manage and inference the constraint sets [26].

We have shown how symbolic DEVS can do the causal propagation required for diagnosis of fault-prone systems modelled with multicomponent DEVS. By allowing the times to be expressed symbolically, we generate a family of trajectories that represent the possible sequences of events resulting in a particular breakdown. Such a family can be employed to trace a detected anomaly back to the components that might have caused it. The approach requires that we represent all failures that could occur and how systems respond to them. In future research, we will assess how our simulation environment helps to visualize, and discover via actual execution, anomalies that need to be incorporated as faults. Guided by morphism preservation principles, we intend to develop tools to facilitate incorporating new fault types into model units directly from observations of simulation runs.

References


