A New Approach to Specify Real-Time Behavior of Distributed Systems

R. Mall
Dept. of Computer Science and Automation
Indian Institute of Science
Bangalore, INDIA.

e-mail: mal@vigyan.ernet.in

L.M. Patnaik
Microprocessor Applications Laboratory,
Supercomputer Education and Res. Centre, and
Dept. of Computer Science and Automation
Indian Institute of Science
Bangalore, INDIA.
lalit@vigyan.ernet.in

Abstract

Specification of real-time behavior of a system requires a significantly different approach compared to specification of the system's non-real-time behavior. Also, a specification formalism should be easy to use, conceptually simple, and should lead to an intuitive understanding about the specified behavior of the system. With these objectives, in this paper, we develop an event-based approach to specify the real-time behavior and relevant properties of distributed systems. In our approach, events are considered as the basic entities and a system is specified by constructing various relations among the events. Based on this formalism, a specification language is also presented. A specification in this language can be automatically transformed into equivalent specifications in many existing formalisms (e.g. temporal logic, first-order logic, etc.) for further analysis. Thus, an event-based specification can be used as a front-end tool for automatically generating specifications in many existing formalisms, which can save the trouble of writing complicated formulas in those formalisms.

1 Introduction

Distributed computer systems are finding increasing use in real-time process control applications. Most of these applications are safety-critical in nature, requiring use of highly reliable computer systems. Building such systems is a nontrivial problem, since the ***correct behavior*** of these systems entails not only functional correctness but also timeliness of the results. Thus, any inaccuracy in the specification of the real-time behavior of such systems can lead to costly system failures. Consequently, a suitable tool to formally specify the real-time behavior and the relevant properties of distributed real-time systems is vital to realizing the required performance and reliability levels.

Specification of real-time behavior of systems requires a significantly different approach compared to specification of non-real-time behavior – as not only the eventual occurrence but also the exact timing of various events becomes important. Further, in case of distributed systems, in addition to precisely specifying the relative timing of various events, ***distributed concurrency*** aspects should also be represented. On the other hand, a specification formalism should be easy to use, conceptually simple, and should also lead to an intuitive understanding about the system's specified behavior. In this regard, we feel that ***post hoc*** introduction of "time" into an existing specification formalism does not lead to natural expression of the real-time behavior of systems. Another motivation for our work is the fact that most of the timing specification tools reported in the literature (e.g., [2,8]) require writing complicated mathematical expressios and/or knowledge of specialized mathematical theories. Complex specifications often obscure intuitive understanding of a system's specified behavior. In this context, we develop a formal specification method that considers ***events*** as the basic entities. We feel that it is intuitively appealing to express the real-time behavior of systems in terms of event occurrences, as events are essentially markers on a space-time diagram. Real-time behavior of systems is expressed by defining various relations on the events. This method, while expressive enough to specify the rich behavior of distributed real-time systems, is also intuitively appealing and simple to use.

An important characteristic feature of distributed systems is that the events occur at the spatially separated nodes. Also, a node typically remains unaware of various events occurring at other nodes of the system, unless it specifically receives a message to that end. Our specification method is designed to represent this important behavioral aspect of distributed systems. For this purpose, we consider events as markers in ***time and space;*** and use nodes as the basic unit of specification of distributed systems. Based on the developed language, a specification language is also presented. Also, specifications in this language can be ***automatically*** transformed into equivalent specifications in many existing formalisms (e.g., temporal logic [2,8], first-order logic [3], etc.) for further analysis. Such automatic transformation can save the tedium of writing complicated formulas while specifying a system in those formalisms.

TH0384-8/91/0000/0194$01.00 © 1991 IEEE

The rest of this paper is organized as follows. Section 2 develops the event-based specification formalism. Based on the developed formalism, section 3 formulates a specification language called TSDS (Timing Specifier of Distributed Systems). Section **4** illustrates the use of the specification language with an example. Section ***5*** presents a comparison of our approach with the related work. Section 6 concludes this paper.

2 Event-Based Specification

2.1 Preliminaries

We consider ***events*** as the basic entities in terms of which the real-time behavior and the relevant properties of distributed systems are specified. Although it is possible to develop the specification formalism solely in terms of occurrence of events, it is more convenient to define and use *actions* as derived entities. Predicates will be used to ascribe meaning to events by relating their occurrences to the aspects of behavior of the physical system. However, we will mainly be interested in the events which change the truth value of predicates. Real-time behavior of distributed systems is expressed by first enumerating the significant events associated with each node of the system, and then by constructing certain relationships among the events. This ordering among events is in the tradition of Winskel's ***event structures*** **[4].**

Definition **2.1:** Each occurrence of an event marks a point in time and space that is of importance in describing a system's real-time behavior. The set of events (E) associated with a distributed system can be partitioned into the following six classes:

i.) ***External events*** **(EE): An** external event occurs due to the action(s) of the environment on an embedded system.

ii.) ***Start events*** (EB): **A** start event marks the initiation of an action (see def. 2.3).

iii.) Stop ***events*** **(ES):** A stop event marks the completion of an action (see def. 2.3).

iv.) ***Transition events*** (ET): A transition event marks the instant at which a change to the truth values of certain formula(s) describing the system behavior occurs. An event marking the transition of a formula X to ***true*** is represented by ($\bullet$ X) and the event marking the time point at which the formula becomes *false* is represented **by ($^{\circ}$X).**

v.) ***Notifier events*** **(EN): A** notifier event at a node marks a point on time at which the node transmits a message to another node.

vi.) ***Notification events*** **(EA):** A notification event at a node marks a point in time, at which it receives a message from another node.

Thus, $E = EE \cup EB \cup ES \cup ET \cup EN \cup EA$, and

$E_i \cap E_j = \Phi$, $E_i \neq E_j$, $E_i, E_j \in \{EE, EB, ES, ET, EN, EA\}$.

An event *e* *can* have more than one instance (occurrence). Each time an event occurs (i.e., an instance), it acquires a label from the set of natural numbers **N,** so that the ith instance of an event ***e*** can be uniquely represented by (e,i), $i \in N$. We will use ***e*** itself to represent ***all*** occurrences of that event. To emphasize the fact that events occur at the spatially separated nodes, event names will be prefmed by the names of the nodes at which they occur. It should be noted that the definition of events in our model is significantly different from those **in [7,4].**

Definition ***2 2*** **:** A node is the basic unit of specification of distributed systems. A node consists of a collection of events (and actions) associated with a common time base, a definition of various relationships among these events (and actions) as well as the relationships of these events (and actions) with those of the other nodes of the system.

Definition **23:** Actions are the schedulable entities in distributed computations. Occurrence of each actions represents finite progress made in a computation. Instances of actions can be defined in terms of the corresponding start and stop events.

The names of actions are unique and are usually prefmed by the names of the nodes at which they occur. Like events, an action can occur a number of times. The functions ***begin*** and end will be used to project out the start and stop events of an instance of an action. Thus for an action instance (a_1,i) the start event is given by $begin(a_1,i)$ and the stop event can be represented by $end(a_1,i)$.

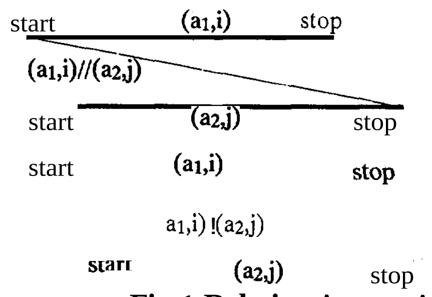
Definition ***2.4*** Predicates relate event occurrences to the behavior of the physical system. The value of a predicate can change with time due to the occurrence of ***transition events.***

In the specification of a system, we are particularly interested in the times at which a predicate (describing aspects of system behavior) becomes ***true*** and at which it becomes ***false*** (i.e. the transition events).

2.2 Relation Among Events

Definition **2.5: A** temporal precedence relation (L_t) among the events in a distributed system is defined **as** follows. The relation $(e_1,i) \angle_t ((e_2,j),t_1,t_2)$, implies that the jth instance of the event e_2 occurs between t_1 and t_2 time units after the ith instance of e_1 occurs.

Definition ***2.6*** A causal relation ($\angle_c$) on events is defined as follow: $(e_1,i) \angle_c (e_2,j)$, if the occurrence of the event (e_1,i) somehow causes the event (e_2,j) to occur eventually. Thus, the causal relation $(e_1,i) \angle_c (e_2,j)$ is, in effect, a short form for the relation (e_1,i) $\angle_t((e_2,j),0,\infty)$. Two events (e_1,i) and (e_2,j) are called ***unrelated(concurrent)***, if $(e_1,i) \not\angle_c (e_2,j)$ and $(e_2,j) \not\angle_c (e_1,i)$.



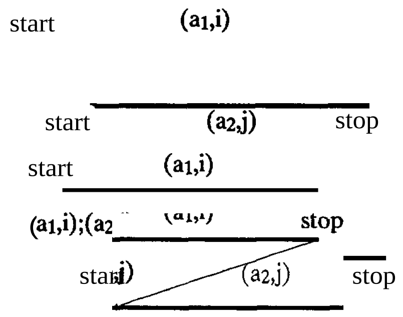


Fig. 1 Relation Among Actions

Definition **2.7**: Two events (e_1,i) and (e_2,j) are said to be causally equal (denoted by $(e_1,i) =_c (e_2,j)$, iff there exists an event (e_3,k) such that $(e_3,k) \angle_c (e_1,i)$, $(e_3,k) \angle_c (e_2,j)$, $(e_1,i) \not\angle_c (e_2,j)$, and $(e_2,j) \not\angle_c (e_1,i)$. Causal equality among actions can be defined in a similar way: two actions (a_1,i) and (a_2,j) are called causally equal $(a_1,i) =_c (a_2,j)$, iff begin$(a_1,i) =_c$ begin(a_2,j).

Definition **2.8** Two events (e_1,i) and (e_2,j) are said to be temporally equal $(e_1,i) =_t (e_2,j)$, if they are causally equal, and occur at precisely the same time instant. Temporal equality is especially useful in describing synchronous events.

Definition *2.9:* Two events e_1 and e_2 are said to be in conflict with each other (represented by e_1 # e_2), if the occurrence of one event forbids the occurrence of the other. The conflict relation provides a way to describe choices available to a system. Obviously, if $e_1 \angle_c e_2$, $e_2 \angle_c e_3$, and e_1#e_4, then e_3#e_4.

2.3 Relation among Actions

The causal dependency among actions can be represented by using the following constructs (also shown in fig. 1): **$(a_1,i);(a_2,j)$** is used to denote that the action (a_2,j) can start only after (a_1,i) completes; $(a_1,i)//(a_2,j)$ denotes that the action (a_2,j) must finish only after (a_1,i) starts; $(a_1,i)\backslash\backslash(a_2,j)$ denotes that the action (a_2,j) can start only after (a_1,i) starts; and $(a_1,i)!(a_2,j)$ means that action (a_1,i) must complete before (a_2,j) completes.

Two actions are called ***sequential*** if any pair of time points in the execution of the two actions can be compared. Two actions are called ***primitively concurrent,*** if there is at least a pair of time points between the intervals representing the two actions not comparable. Thus, the actions (a_1,i) and (a_2,j) in (a_1,i); (a_2,j) are sequentially dependent, whereas those in $(a_1,i)\backslash\backslash(a_2,j)$, $(a_1,i)//(a_2,j)$ and $(a_1,i)!(a_2,j)$ are primitively concurrent. Two actions (a_1,i) and (a_2,j) are called ***totally concurrent*** (denoted by $(a_1,i) || (a_2,j)$), if no time point in their execution can be compared.

Definition **2.10** : A computation (C) represents a single run of the specified system. A computation can be represented by a single action C, which can be decomposed into more primitive actions. The start event of the action C, e_s = begin(C) signals the start of a computation and the stop event of the action C, e_e = end(C) signals the end of a computation. Of course, for cyclic and nonterminating executions, the stop event may never occur.

3 Specification Language TSDS

In order to facilitate specification of real-time behavior of distributed systems in the event-based model discussed in the previous section, we have developed a language called TSDS (Timing Specifier for Distributed Systems) providing a set of language constructs for this purpose. After enumerating the nodes of the system along with their associated events and actions, the language constructs of TSDS can be used to express the relations among the events. Each construct of the language actually represents a set of relations among the events of the system.

The following are the language constructs of TSDS and the relations they represent:

Events

If (e_1,i) and (e_2,j) are event instances, then

Construct	Interpretation
(e_1,i) precedes (e_2,j) by (t_1,t_2)	$(e_1,i) \angle_t ((e_2,j),t_1,t_2)$
(e_1,i) succeeds (e_2,j) by (t_1,t_2)	$(e_2,j) \angle_t ((e_1,i),t_1,t_2)$
(e_1,i) causes (e_2,j)	$(e_1,i) \angle_c (e_2,j)$
e_1 conflicts with e_2	e_1 # e_2

Actions

If (a_1,i) and (a_2,j) are action instances, then

Construct	Interpretation
$(a_1,i);(a_2,j)$	end$(a_1,i) \angle_c$ begin(a_2,j)
$(a_1,i)\backslash\backslash(a_2,j)$	begin$(a_1,i) \angle_c$ begin(a_2,j)
$(a_1,i)//(a_2,j)$	begin$(a_1,i) \angle_c$ end(a_2,j)
$(a_1,i)!(a_2,j)$	end$(a_1,i) \angle_c$ end(a_2,j)

when (e_1,i), a_2 with period t_1

$(e_1,i) \angle_t$ (begin$(a_2,1),t_1,t_1)$

begin$(a_2,j) \angle_t$ (begin$(a_2,j+1),t_1,t_1)$

(a_1,i) fork $((a_2,j),(a_3,k))$	$end(a_1,i) \angle_c begin(a_2,j)$
	$end(a_1,i) \angle_c begin(a_3,k)$
	$begin(a_2,j) =_t begin(a_3,k)$
(a_1,i) join $((a_2,j),(a_3,k))$	$(a_1,i)!(a_2,j), (a_1,i)!(a_3,k)$
(a_1,i) lasts ti	$begin(a_1,i) \angle_t (end(a_1,i),t_1,t_1)$

Predicates

If P_1 and P_2 are predicates describing aspects of behavior of the physical system, then

Construct	Interpretation
Always Pi	$\bullet P_1 =_t e_s, {}^\circ P_1 =_t e_e$
P_1 since P_2	$\bullet P_1 =_t \bullet P_2$, "Pi $=_t e_e$
P_1 until P_2	$\bullet P_1 =_t e_s, \bullet P_2 \angle_c {}^\circ P_1$
when (e_1,i), henceforth P	$(e_1,i) =_t \bullet P, {}^\circ P =_t e_e$

4 Example

We now give an example of the event-based specification of an autonomous robot system abstracted from an automatic manufacturing plant problem.

4.1 Informal Problem Specification

An object (partially complete product) enters the work-space of an autonomous robot (Fig. 2). The sensors of the robot on sensing an incoming object, signals the perception node. The perception node analyzes the signals to identify the incoming object, and passes on this information to the planner node. The planner node decides about the work to be performed, records arrival of the object and the work performed, and also informs the actuator node for initiating the work. The timing requirement is that the system must initiate and complete the necessary actions within 500 time units (tus) and **1000** tus respectively from the arrival of any object.

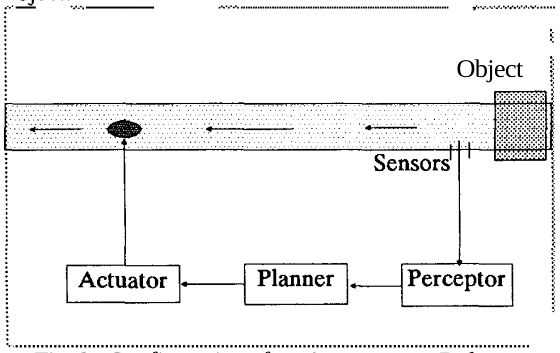


Fig. 2 Configuration of an Autonomous Robot

4.2 Event-Based Specification

Perceptor node (np)

external event: np.arrival	Arrival of an object.
action: np.ident	Object identification.
notifier event: np.nctrlr	Notify Planner node.

(np.arrival,i) causes begin(np.ident,i)
end(np.ident,i) causes (np.nctrlr,i)

Planner node (nl)

notification event: nl.objarr	Arrival notification.
action: nl.dwork	Decide on work to be performed.
nl.update	Record object and work performed.
notifier event: nl.notact	Notify actuator node.

(np.nctrlr,i) causes (nl.objarr,i)
(nl.objarr,i) causes begin(nl.dwork,i)
end(nl.dwork,i) causes begin(nl.update,i)
end(nl.dwork,i) causes (nl.notact,i)

Actuator node (na)

notification event: na.ntact	Notification of work
action: na.actuate	Drive actuators.

(nl.notact,i) causes (na.ntact,i)
((np.ident,i);(nl.dwork,i)) fork
((nl.update,i), (na.actuate,i))
(np.arrival,i) precedes begin(na.actuate,i)
by (500,1000).

5 Related Work

The presented specification formalism has evolved from a study of the formalisms **[1,2,3,4,5,7]**. The major difference between our work and those reported in [2,3] is that we specifically consider distributed real-time systems, and consider events as the basic entities of specifications. We consider nodes as the basic **units** of specification and explicitly consider message-passing aspects. We express the real-time behavior of distributed systems by constructing various ordering relations among the events. Our event-based specification model is in some respects similar to the event-action model of [1,3,5]. However, the major difference is that unlike those in [1,3,5], we use events as the basic entities of specification, and the real-time behavior is represented as relations among the events. Also, our model tries to represent distributed concurrency aspects **in** the tradition of Winskel's event *structures*. Further, our approach allows easier and more intuitive expression of the rich behavior of distributed real-time systems. For example, expressing the *join construct* in TRIO [2] requires writing a number of formulas which obscure the intuitive understanding of the involved behavior.

6 Conclusions and Discussions

In this paper, we have presented a method for specification of real-time behavior of distributed systems using an event-based approach. Our specification method is an attempt towards the development of a formalism for specification of the real-time behavior of distributed systems ***ab initio***, rather than ***post hoc*** imposition of the timing aspects on an existing specification method. Also, we feel that an event-based approach leads to a natural specification of real-time behavior of distributed systems since events inherently represent points on a space-time diagram. Further, automatic translation of an event-based specification into equivalent specifications in a number of existing specification formalisms *can* save the tedium of writing complicated expressions in those formalisms. Our current work is directed towards automatically generating specificationsin a modal logic called ***distributed logic*** [8] (which we have developed specifically for specification and verification of real-time behavior of distributed systems), from event-based specifications. We are also working towards realizing an executable specification tool based on the distributed logic specifications.

References

[1] G.H. MacEwen and T.A. Montgomery, "Expressing Requirements for Distributed Real-Time Systems," IEEE Computer Society Workshop on Real-Time Operating Systems, Cambridge, Massachussetts, 1987, pp. 125-128.

[2] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO: A Logic for Executable Specification of Real-Time Systems," Journal of Systems and Software, Vol. 12, 1990, pp. 107-123.

[3] F. Jahanian and A. Mok, "Safety Analysis of Timing Properties of Real-Time Systems," IEEE Transactions on Software Engineering, Vol. SE-12, September 1986, pp. 890-904.

[4] G. Winskel, "*An* Introduction to Event Structures," in J. de Bakker, E. de Roever, and G. Rozenberg eds., ***Linear Time, Branching Time and Partial Order in Logics and Models*** of ***Concurrency,*** Springer-Verlag, Lecture Notes in Computer Science, Vol. 354, 1989.

[5] R. Mall and L.M. Patnaik, "Specification and Verification of **Timing** Properties of Distributed Real-time Systems," in Proc. of IEEE TENCON, Hong Kong, September 1990.

[6] L.M. Patnaik and R. Mall, "Critical Issues in Real-Time Software Development," in Proc. of National Workshop on Computers in Real-Time Applications, Indore, India, February, 1991.

[7] D. Murphy, "Approaching a Real-time Concurrency Theory," in Proc. International BCS-FACS Workshop on Computing Theory, Leicester, July 1990.

[8] R. Mall and L.M. Patnaik, "A Temporal Logic over Partial Orders for Verification of Real-time Properties of Distributed Programs," to be presented at IEEE TENCON, Sept. 1991, New Delhi, India.