**Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems**

FARNAM ,IAHANIAN farnam@eecs.umich.edu

Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, M148109

RAGUNATHAN RAJKUMAR rr@sei.cmu.edu

Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA 15213

SITARAM C. V. RAJU\*\* sitaram@cs.washington.edu

Department of Computer Science and Engineering, University of Washington, Seattle WA 98195

* The distributed real-time system is viewed as an event model.
* The system properties are expresses as the invariant relationship between the events, which is monitored at the runtime.
* The invariants are expressed using Real-Time Logic (RTL).
* It uses distributed monitors i.e each node contains a local monitor which captures the event generated at the local node. The local node may transmit the events which is required by remote monitors.
* The paper addresses the following issues:
* **Time of detections of violations**: Early detection of violation should be captured so that system can take corrective measures bore the violation actually occurs
* **Number of Messages**: In a distributed the occurrences of events is communicated through the message passing. So no. of messages should be reduced to minimize the overhead.
* **Clocks & Synchronization**: The paper considers the time deviation between the processors for evaluating timed assertions at runtime as there may be some time differences among the nodes in a synchronized system.
* To detect timing constraints, multiple occurrences of an event is recorded and are stored in a circular queue called its *event history,* which maintains the last n occurrence times of the event. The value of n depends on the assertion.
* The paper uses two functions for accessing the event histories: the occurrence function **@(e, i),** which returns the time of the ith occurrence of event e, and **@val(v, i),** which returns the value of the ith assignment to a variable v.
* When the index i is negative, it refers to the ith most recent occurrence of the event in a computation. For example, @(e, -1) denotes the time of the most recent occurrence of e.
* Example of RTL : Example: Consider two events el and e2 which must always occur in pairs and within 5 time-units of each other. The following formula specifies such a constraint.

For all i @(el, i) <= @(e2, i) + 5 V @(e2, i) <= @(el, i) + 5

* **Constraint Graph Representation:** If a timing assertion is represented as a directed, weighted graph, called a constraint graph. Each constraint graph represents a conjunction of predicates, and each edge in the graph is a predicate of the form: @(e,i) <\_ @(f,j)±C such that i, j are integer variables/constants and C is an integer constant.
* The constraints in timing assertion can be 2 types : delay and deadline
* Following fig. shows the delay constraint between two events



* The following fig. shows the deadline constraints



* The following fig. shows the events with delay and deadline constraints



* **Calculation of implicit Timing Constraints:**

The Implicit timing constraint can be used to find out the violation that may occur. The above fig. It consists of two explicit timing constraints: a deadline edge and a delay edge. Events el, e2 and e3 occur on processors 1, 2 and 3, respectively. There is an explicit deadline from el to e2. In addition, since there is a path from el to e3 of length 6**, there is an implicit, intermediate deadline of 6 from el to e3**. If the intermediate deadline is not met, then either the explicit deadline or the delay constraint from e3 to e2 will eventually be violated. If the violation of the implicit constraint between el to e3 is detected, the system can be notified before any of the two user-specified constraints are violated.

* **Checking Constraint Graph for Violation:**

When an event occurs that may affect the satisfiability of a timing assertion, a **satisfiability checker** is invoked to check for violations.

In a constraint graph, the earliest time a constraint can be violated is as follows:

1. A delay constraint will be violated, if for a path of negative length -T (T >=O) from vertex en to vertex O, the event corresponding to vertex en happens before time T.
2. A deadline constraint will be violated if the minimum length T (T > O) of all shortest paths from vertex 0 to all other vertices is to a vertex em and the event corresponding to vertex em does not happen at or before T.

* **Minimization of Messages**
* Whenever an event ei occurs, its occurrence needs to be communicated (directly or indirectly) to the monitor of any vertex ej, if in the shortest path graph, there exists a path with positive weight from ei to ej or a path with negative weight from ej to ei.
* A shortest path algorithm is run on entire graph such that the shortest path from any node to every other node is obtained.
* Delay violations. If there is a vertex ej such that there exists a path from ej to ei of negative length, then there is a precedence constraint between ei and ej. The occurrence time of ei must be sent to ej's monitor, so that when ej happens the monitor can check if the delay constraint has been violated. Hence the occurrence time of ei must be sent to all such monitors.
* Deadline violations. If there is a vertex ej such that there exists an edge with positive weight from ei to ej, then the occurrence time of ei must be sent to ej's monitor, so that the monitor can check if ej happens within the deadline. There may be events ek that precede ej, but not ei. Such events will have earlier deadlines and represent intermediate points at which eventual violations of delays/deadlines can be detected. Hence messages must also be sent to all such ek's monitors.
* The requirement of earliest violation detection can be relaxed, to reduce the number of messages. In this case, the occurrence time needs to be sent only to all ej's monitors such that there exists an edge with positive weight from ei to ej.
* **A distributed runtime Monitor Prototype**

Authors have implemented a prototype of the distributed monitoring run-time system on a

network of IBM RS/6000 workstations connected to a token-ring. The workstations run

AIXv.3, a Unix variant that supports the assignment of static priorities to real-time processes,

which can immediately preempt other user processes. Real-Time processes in AIXv.3 have

higher priority than all other user processes, and the scheduler allows a higher priority

real-time process to preempt a lower priority real-time process immediately. Furthermore,

a fine-precision timer facility is supported, and the hardware clock registers in the RS/6000

can be read directly by a user process, which makes it possible to differentiate local events

that occur 1 micros apart.