Rule Based Event Management Systems

Ridhika Malik
Guru Gobind Singh
Indraprastha University
India
ridhikamalik@gmail.com

Nandan Parameswaran
University of New South Wales
Australia
paramesh@cse.unsw.edu.au

Udayan Ghose
Guru Gobind Singh
Indraprastha University
India
g_udayan@lycos.com

Abstract

Event Management is one of the most lucrative and growing professions today. At present event management is done by humans. With the growing demand for managing large events, there is a rising demand for building intelligent systems to manage events. The so called event management systems today are only data processing systems that are unable to carry out decision making task on their own. Event management systems today do not consider emergencies and risk assessment as part of their execution. In this paper, we present an approach for representing events and monitor their execution. In particular, discuss the exceptions that can occur during an event execution and how they can be managed using event management rules. We present strategies for writing management rules that are used to handle problematic events and to build a DAG based programming system for event management. Our simulation results show how the performance of our event management system performs with the exception management rules.

Introduction

Event representation and event management have been gaining attention increasingly from the researchers of diverse discipline. Many real world situations can be viewed as events involving resources whose states change over time causing events of complex patterns. Often agents not only participate in causing the events but also deliberately manage the events by observing and reasoning with them. In this paper, we present an approach where we represent events hierarchically at different levels of abstraction while relating events at a given level of abstraction both temporally and causally. Each node in the hierarchy is associated with a set of rules called the event management rules which monitor event at that node.

The paper is organized as follows. In the next section, we present the related work. In the section Event Representation, we discuss how events can be represented hierarchically. In the following section, we present a programming system based on the event representation language VERL and discuss the exceptions that can occur during an event execution. We then describe our implemented programming system for event management (PSEM) and discuss the results of our analysis. The final section concludes the paper.

Related Work

Allen et al. [1] argue that events are methods used to classify relevant patterns of change rather than entities in the real world. Sowa [2] categorize events as changes that occur in the discrete steps of a process. In knowledge representation, an event is an activity that involves an outcome [3] or an arbitrary classification of a space-time region by a cognitive agent [4]. Event and time are intrinsically linked as discussed in [5]. The DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering)[6] is an event ontology, in which events are a subclass of perdurant occurrences that are disjoint to the entities of endurant, quality, and abstract. The SUMO (SuUpper Merged Ontology)[7], designed by the IEEE Standard Upper Ontology Working Group consists of a set of concepts, relations, and axioms where abstract and physical entities are divided, in which the physical entities are classified into objects and processes.

Various event representation languages have been proposed such as SDL [9] (Scenario Description Language), VERL [11](Video Event Representation Language) and SWRL[10]. OWL [8] (Web Ontology Language) is adequate for describing static concepts and relationships but is insufficient for modeling event ontology with dynamic features [10]. VERL stands close to our need for describing events and event plans. But VERL faces certain limitations. For example, VERL does not handle exceptions events and cannot be used directly for exception handling, and as such can not be comfortably used for large scale events. However, in this paper, we use VERL to describe events, and compile VERL programs to produce event DAGs.

Event Representation

Let S be the aggregate states of all resources in the world, and Γ be the set of actions that an agent can perform at any state.

Event as a DAG structure

We represent an event using a dag (directed acyclic graph) where each node is a sequence of states $s \in S$ occurring over an interval of time and each state is associated with an action $\alpha \in \Gamma$ that can be performed by an agent when the control reaches that state. The intervals are organized hierarchically where the higher level nodes represent abstract events and lower level nodes represent nodes with more details. Additionally, the nodes are linked to show Allen's [1] temporal relations, causality, and domain dependent relations. Figure 1 below shows the structure of a simple event.

VERL and the DAG structure

Events as a DAG structure are not a convenient formalism for representing large events. In this paper, we have used VERL for event description since VERL permits description of an event the way a programming language permits the description of a computation as a program. In VERL, the primary entities (objects) are events and operations are permitted on these events. Abstract events can be described as composite events which in turn are described using the "Process" concept in VERL. A process is defined using several sub events. Events can be related using the relations AND, OR, sequential, overlap, etc., and composed using constructs such as if, while, repeat-until, and loops. Besides this, the temporal relations could be implemented using Allen's interval algebra [9].

A VERL description can be compiled and represented as a DAG by beginning with the abstract process entity which will form the root node of the DAG. In Figure 1 below, we depict the root node to represent an abstract event. On further analysis of the VERL code, we identify the next level nodes and describe them using the AND, OR, SEQ, REP, and PAR operations as shown.

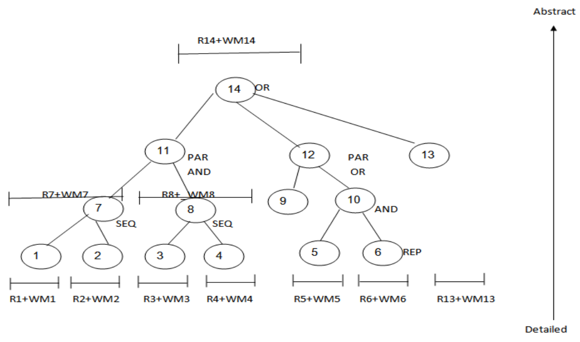


Figure 1: An event structure as a DAG.(Undirected arcs have been used for simplicity.)

Event management rules

When events are performed (or executed) by agents, often unforeseen situations arise resulting in unexpected states of resources involved in the event. An event plan needs to be robust in handling exceptions. More the exceptions handled, the more secure are our plans. To handle exceptions, we attach excepting handler rules to the event nodes. These rules are called event management rules. We briefly discuss below a few event management strategies.

An event management rule is a set of rules (attached to an event node) that monitors the states of the resources involved in the event execution. As long as the resources are in acceptable states, the event execution is said to be normal. When a resource reaches an erroneous state, the execution of the event is said to have reached an exceptional situation, and the event management rules associated with the event node are triggered to handle the exceptional situation.

Prioritize event nodes In a given DAG structure, not all nodes may be important in an event management activity. Thus, our first task is to attach a weight at each event node (at all levels) where the weight signifies the importance of the nodes for the overall event to perform successfully. The resources that are involved in the event nodes of higher weights are identified as resources of higher priority. The event management rules will have to be "hard working" rules while monitoring resources of higher priority.

Risk assessment in nodes Within a given event (which spreads an interval of time), prioritize states that based on the risk associated with the states. Suppose that Σ is a set of valid states, E is a set of error states. When α action is attempted on a given state $s_i \in \Sigma$ of a resource r, in a real world, it is not guaranteed that the resource r will always reach its targeted state $s_j \in \Sigma$. There is always a non zero probability that r will end up in one of the error states s□ ∈ E. Larger the probability of reaching a state in E, higher the risk associated with the state s.

Weighted relational links Often, the failure of an event can contribute to the failure of other events in varying degrees. One way we can account for the degree of impact of the failure of an event is to assign a weight to the relational links as well that emanate from the event node. This will help in propagating the effect of the failure of one event across the entire DAG structure. Note that in this strategy, even the relation between the parent and child node in a DAG structure will be assigned a weight. The weight on the relational link in a sense quantifies the dependency of one node upon the other. The following relations are needed to be considered.

Parent-child relation: Not all child nodes will be equally important. Thus those that are considered crucial can be assigned higher weights.

Parallel event nodes: Typically, parallel events execute independently as they are often mutually independent. They may have lower weights for their relational link.

Causal Dependency: Links of this type are more important than the parent-child relationship, and thus must be weighted more.

Nested management rules: When the management rules (attached to an event node) themselves run into exceptional situations, we call this as a nested exception. In such situations, we need another level of exception management rules. In such situations, there is a risk that cost of the event management activity itself might surpass the cost of executing the overall event, and thus it is permitted only when the overall cost is less than the threshold assigned to the current event node. When the management cost exceeds this threshold, event management of that node will be suspended. The cost is typically estimated in terms of the physical resources involved and the time that will be spent.

Executing a DAG structure

Executing a DAG structure involves executing the events in the DAG. We assume that the nodes are temporally organized from left right (time flows from left starting at the left most leaves). (Note that there can be several left most leaves temporally.) We begin at the lower most event intervals (nodes) at their left most (starting) point. We begin with the first state $s_0 \in \Sigma$ in the event which will be the initial state. As we execute the actions at s0, the involved resources change states leading to an aggregate state s_1. At this point, two updates have to be made: (a) revise the states of the parent event nodes; and (b) revise the actions at the parent event nodes. We refer to this update as bottom-up data flow. In general, the bottom-up data flow will affect all nodes from the leaf nodes up to the top most nodes. (Note that there may be more than one such top most node in the DAG.)

As we execute the actions, resources transitioning from one state to another, an interesting situation arises when a resource enters an error state instead of a normal state. This signals that a management action is now necessary to bring the resources from the error state to the normal state. The management actions are executed by the event management rule attached to the event node.

At any event node n, when the management rules are triggered, two types of actions take place: (a) the data that flows in from the children nodes are consolidated to compute and current (abstract) state of the resource appropriate to the level of the node n; and the results that need to be sent to the parent node are computed and sent to the parent node; (b) wait for the data for results from the parent node, update the current state at node n, compute new results and send it to the children node. The event execution and management involves information flowing from the bottom to top nodes and from the top nodes to the bottom nodes.

PSEM: A Programming System for Event Management

Our implementation of an event management called PSEM (Programming System for Event Management) consists of the following modules:

VERL Compiler The VERL compiler accepts a procedural description of an event and compiles it into an event DAG structure. (This module is still under development. For the results reported in this paper, manual compilation was performed.)

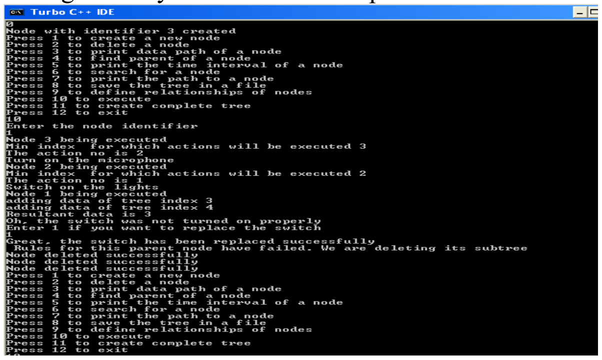
DAG editor The DAG structure build from the VERL Compiler can further be edited manually when necessary using this editor. This will typically happen when event management rules begin to fire, changes in external situations may demand editing changes to the DAG structure. For example, the user can modify an event's attributes like start time, end time, number of children, actions, and the management rules associated with the event.

Causality based Heuristics

Causal relationship among the events is perhaps the most important relationship amongst all relationships. In our system, we use heuristics to exploit causal relationship specifically. Assuming causal relationship to be transitive, we use the degree of an event node to ascertain its significance in the DAG. We use the following heuristics while reasoning with causal links:

a) The impact of causality weakens as it propagates over time across several future events; thus immediately causally depending event nodes will be affected more strongly than the future event nodes that will be affected eventually in future; b) a node with more causal links emanating from it is more important (higher priority) than the one that has less; c) if two nodes have comparable number of out going causal links, then the one with higher number of incoming causal links is more important.

Figure 2 below shows the sample session of the event management system PSEM we implemented.



```
0
Node with identifier 3 created
Press 1 to create a new node
Press 2 to delete a node
Press 3 to print data path of a node
Press 4 to find parent of a node
Press 5 to print the time interval of a node
Press 6 to search for a node
Press 7 to print the path to a node
Press 8 to save the tree in a file
Press 9 to define relationships of nodes
Press 10 to execute
Press 11 to create complete tree
Press 12 to exit
10
Enter the node identifier
1
Node 3 being executed
Min index  for which actions will be executed 3
The action no is 2
Turn on the microphone
Node 2 being executed
Min index  for which actions will be executed 2
The action no is 1
Switch on the lights
Node 1 being executed
adding data of tree index 3
adding data of tree index 4
Resultant data is 3
Oh, the switch was not turned on properly
Enter 1 if you want to replace the switch
1
Great, the switch has been replaced successfully
 Rules for this parent node have failed. We are deleting its subtree
Node deleted successfully
Node deleted successfully
Node deleted successfully
Press 1 to create a new node
Press 2 to delete a node
Press 3 to print data path of a node
Press 4 to find parent of a node
Press 5 to print the time interval of a node
Press 6 to search for a node
Press 7 to print the path to a node
Press 8 to save the tree in a file
Press 9 to define relationships of nodes
Press 10 to execute
Press 11 to create complete tree
Press 12 to exit
```

Figure 3 below shows a DAG structure for a video clip (Mickey Mouse Clubhouse HOT DOG Song, 65secs on youtube). Figure 4 show the node distribution at different levels of abstractions.

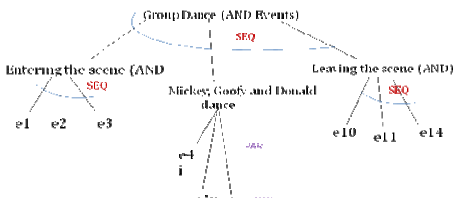


Figure 3: DAG structure for Hot Dog example

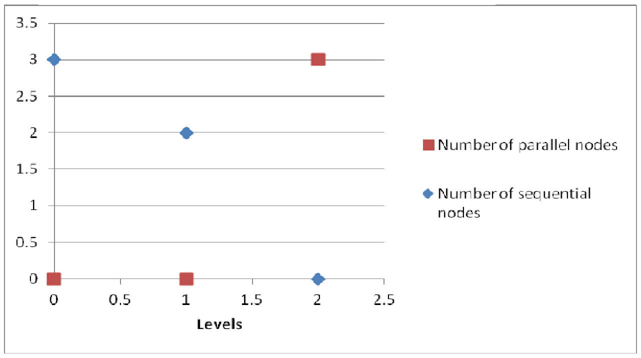


Figure 4: Event distribution in levels

The trend shows that as we go from root to leaf node parallelism increases but sequentialism decreases i.e. at lower levels the events generally occur concurrently.

Scores

Let us define indegree and outdegree values (based on causal link) for each node as follows:

e1: (0, 1) e2: (0, 1) e3: (0, 1) e123: (3, 1) e4i: (0, 1) e4ii: (0, 1) e4iii: (0, 1) e4: (3, 1) e10: (0, 1) e11: (0, 1) e14: (0, 1) e910: (3, 1) e111: (3, 0)

We divide the nodes in to two sets S_1 and S_2 based on their indegree and outdegree values.

S_1 = { e1,e2, e3, e4i,e4ii,e4iii,e10,e11,e14 }. All these nodes have indegree 0 (outdegree 1), and thus the events in them do not depend on any other nodes.

S_2 = {e123, e4 , e910} and these nodes all have indegree 3 and outdegree 1. Since the outdegrees are the same for the nodes in both the sets, we distinguish between the nodes in these two on the basis of their indegree. Thus, we consider the nodes in S_2 to be contributing more to the success of the overall event than S_1, and we signify this fact by choosing a higher weight for these nodes. In a real world scenario, events often do not get executed completely, and thus when they occur, they only occur partially. It is then useful to quantify the overall success of an event.

We quantify the (partially) successful completion of an event e at a node i with weight w_i by a score c(i) as follows. Let n be a leaf node. Then, $c(i) = w_i$, if the node was executed successfully; else $c(i) = -w_i$. If the node is not a leaf node, then c(i) = weighted average of the scores of all the children nodes. A positive score signifies the fact that the event has been partially successful, while the higher negative score signifies a total disaster.

Assigning a weight of 0.3 for all nodes in S_1, and 0.9 for all nodes in S_2, the scores have been plotted in Figure 5 below.

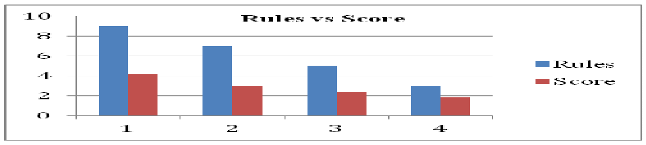


Figure 5: Scores and management rules

We observe from the figure above that the score improves as we employ more management rules.

Conclusion

The goal of our research is to provide a framework for a flexible event management system. The DAG structure supports a scalable event representation. We have also argued that with every event, we need a set of management rules to make event execution robust against unpredictable changes in the environment. Future work involves developing management strategies for complex events such as the ones occurring in long action filled videos.

References

[1] Allen James, F. and Ferguson George. “Actions and events in interval temporal logic” Journal of Logic and Computation, 4(5):531–579, October 1994.
[2] Sowa J. F.. Knowledge Representation. Brooks/Cole, 2000.
[3] Allen, J.F., “Towards a general theory of action and time”, Artificial Intelligence, 23:123-154 (1984).
[4] Hobbs, J.R., Pan, F., “An Ontology of Time for the Semantic Web”, TALIP, 3(1):66-85 (2004).
[5] Nguyen, P. and Corbett, D., “A Formalization of Subjective and Objective Time Ontologies”, 3rd Australasian Ontology Workshop (AOW-07), Gold Coast, Australia, CRPIT series, 85:45-54 (2007).
[6] Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A., and Schneider L..Wonderweb deliverable d17. the wonderweb library of foundational ontologies and the dolce ontology
[7] Niles I. and Pease A.. Towards a standard upper ontology. In Proc. of the 2ndInter. Conf. on Formal Ontology in Information Systems (FOIS-2001), 2001.
[8] Bechhofer, S., et al., OWL Web Ontology Language Reference,Feb. 2004, http://www.w3.org/TR/owl-ref.
[9] Van-Thinh Vu ,”Temporal Scenario for Automatic Video Interpretation”, Ph D Dissertation , Université de nice sophia antipolis, October 2004.
[10] Niles I. and Pease A.. Towards a standard upper ontology. In Proc. of the 2nd Inter. Conf. on Formal Ontology in Information Systems (FOIS-2001), 2001.
[11] Bolles B.; Nevatia, R., “A hierarchical video event ontology in owl”, Proc. ARDA Challenge Workshop, 2004