ARL-TN-1106 • FEB 2022

DEVCOM ARMY RESEARCH LABORATORY

The Tentative Event Strategy (TES): A Work Avoidance Strategy

by Cem Karan

ARL-TN-1106 • FEB 2022

The Tentative Event Strategy (TES): A Work Avoidance Strategy

by Cem Karan
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE		***Form Approved OMB No. 0704-0188***

1. REPORT DATE *(DD-MM-YYYY)* Feb 2022	**2. REPORT TYPE** Technical Note	**3. DATES COVERED (From - To)** October 2012–November 2020
4. TITLE AND SUBTITLE The Tentative Event Strategy (TES): A Work Avoidance Strategy		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Cem Karan		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLC-IA Adelphi Laboratory Center, MD 20783		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-1106
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Author's email: <cem.f.karan.civ@army.mil>.

14. ABSTRACT
This technical note presents the Tentative Event Strategy (TES), a strategy for reducing the work performed by a simulation engine when the deviation in the duration of different events can be very large and highly unpredictable. The strategy is compared to and contrasted with other methods, including discrete time step simulation techniques and continuous time methods. Quantitative analysis is not performed in this work as the strategy described is not appropriate for all types of simulations. Instead, the author describes his own experiences and provides suggestions on when and how to predict if the strategy could be useful in a given situation.

15. SUBJECT TERMS
simulation, software engineering, discrete event simulator, concurrent simulator, TimeWarp

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Cem Karan
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	19	**19b. TELEPHONE NUMBER (Include area code)** 301-394-0667

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

1. Introduction **1**

1.1 The Tentative Event Strategy 2

1.2 Why the Tentative Event Strategy (TES) Is Useful 4

2. Related Work, and the Author's Experiences **5**

2.1 Discrete Time 5

2.1.1 Fixed Time Step 6

2.1.2 Dynamically Adjustable Time Step 6

2.2 Continuous Time and Events 7

3. Experiences **8**

4. Conclusions and Future Work **9**

5. References **11**

Glossary **13**

Distribution List **14**

1. Introduction

Writing simulators is an art where the practitioner must balance the accuracy of the results obtained against the resources required to gain useful information. A significant resource that is often in short supply is time, which is why strategies that reduce work without sacrificing accuracy are generally useful. This work describes a simple method termed the Tentative Event Strategy (TES) that is useful when events have starting and ending dates that can be predicted in the future, but which may be obsoleted by other events before they can conclude. The method is an extension of the well-known Global Event List (GEL) protocol traditionally used by discrete event simulators (see[1] and Section 2), but which simplifies event cancellation. TES can be viewed as the dual of the well-known Time Warp Strategy,[2,3] where instead of rolling back time when a conflict is detected, obsolete events are discarded before their state is committed to the global state.

As a motivating example of why tentative events might be useful, consider a simulation of a moving robot confined within a box and a simulation engine that takes as input the current position and velocity of the robot to predict the moment that the robot collides with one of the walls. These events are tentative in that they are predictions of the future; if the robot changes its velocity before the moment when the collision is expected to occur, then the tentative event is no longer valid to execute or commit to the simulator's global state. However, continuously verifying that other events have not affected the collision event may be prohibitively expensive to calculate. What is needed is a way of quickly determining if the tentative event is still relevant when it is time to commit its changes to the global state. The TES is one method that can be used to make this determination quickly.

The TES was successfully used by the author in his doctoral work in the implementation of a simulator that simulated events whose duration could be as short as a nanosecond, but might be as long as a full day. In the author's opinion, the TES as a strategy is most useful when all of the following statements hold true:

1. A simulation is a collection of datum, each of which is atomic. The collection of datum contains all information about the state of the simulation (no external state), and every version of every datum can be associated with its own unique canary (described in Section 1).

2. Global simulation state is only modified when an event is committed.
3. There are large spans of "boring" time that have no effect on simulation state and that are not of interest to the researcher.
4. It is possible to predict at least some future events using the current state alone.
5. Any particular event is only dependent on a small subset of the global data[*].
6. Some future predictions will be cancelled, and it is not easy to decide which ones will be cancelled until the simulation has evolved further.

While the strategy may be successfully used under other circumstances, in the author's opinion, these are the conditions where the TES will be most useful.

1.1 The Tentative Event Strategy

The strategy depends on canaries and tentative events. Canaries are mutation-unique 128-bit unsigned integers that are incremented each time the datum is mutated. They are similar to version numbers, but unlike version numbers, the amount that they are incremented by need not be uniform with each mutation, nor are they guaranteed to be strictly increasing in value[†]. The only strict requirement is that every mutation of a given datum be given its own mutation-unique canary[‡]. An alternative to using an incrementing counter is to use Universally Unique IDentifiers (UUIDs).[4] The advantage of using increment operators over that of using UUIDs is that incrementing a single value may be faster on a given platform than creating a new UUID[§].

A canary is cheap to create, cheap to copy, and cheap to compare with any other canary. Most importantly, a canary is conceptually immutable; if two canaries are

[*]For example, the motion of a robot only depends on its internal decision-making processes and collision interactions; radio communication events have no effect on motion events, even though there may be an indirect connection where a robot receives a radio communication, which affects its internal decision-making processes, which then affect its motion, but chains of events can be reduced to the behavior of the individual links between events. That is what we discuss here.

[†]This allows the use of some bits as flag bits, which can be helpful in some circumstances.

[‡]In particular, mutations must not reuse canaries while any uncommitted event holds a copy of the canary. While this can be guaranteed by having a set of in-use canaries, the most pragmatic choice is to use a counter that can be guaranteed to never roll over, which ensures that the canaries are never reused.

[§]The author strongly recommends testing all techniques before adopting them for use within the reader's own simulation engine to see if they are the most appropriate solution for the reader's own situation.

not equal, then it implies that something has changed about the item that they are attached to[*]. Given these properties, we can now discuss how canaries are used.

Not only does every datum in the simulation have its own canary associated with it, each and every time a given datum is mutated, it is given a new canary. Since the canaries are guaranteed to be unique over time, we can immediately determine if a datum has changed since we last checked it by storing a copy of the canary with a tentative event when the event is created, and then comparing it with the current value of the datum's canary when the event is ready to be executed. In short, a simulation engine reduces to the following:

1. When a tentative event is created, it captures copies of the canaries of each datum that could affect execution of the event and stores those copies with the event.
2. The event is pushed into a minimum-priority queue that is ordered by when events are supposed to occur.
3. The next event is popped and examined. If and only if the canary values in the tentative event match the current canary values, execute and commit the event. Otherwise, discard it.

As an example of how the TES is applied, consider a simple simulation consisting of two robots driving toward one another. It is possible to predict when the robots will collide with one another based on their current velocities and positions, and to schedule an event for that moment in the future. However, as soon as one (or both) robots alter their velocities, the collision event will be obsolete and must not be committed to the global state. In this particular case, as soon as a robot alters its velocity, the canaries associated with each datum (position and velocity) will be altered at that time. A new collision event can be created with the new canary values of the relevant data and be pushed into the priority queue. As events are popped and executed, the old and new events will eventually be encountered and examined for execution. Since the old event's canaries no longer match the canaries of the data used to generate it, the old event will be silently discarded. If the new event has not been superseded by some other event, then it will be executed.

[*]It is not always obvious that a particular datum has changed; this is the well-known ABA problem. See[5] for further details.

1.2 Why the TES Is Useful

There are at least two reasons why the TES is useful. First, events are not necessarily committed in the same order as they are executed. Second, it makes cancellation much simpler.

The first point is best illustrated by reconsidering the two robot examples discussed earlier. Imagine that when the first collision event was calculated, both robots were traveling toward each other at $1\,\mathrm{m\,s^{-1}}$. Before the event is executed, the robots suddenly increase their speeds toward one another to $2\,\mathrm{m\,s^{-1}}$. In this case, the second event will clearly occur sooner than the first event. Finding the first event in the priority queue to cancel it may require $\mathcal{O}\left(n\right)$ search time over the list of future events[*]. If the event is not cancelled or otherwise tracked so that it is not executed, then the simulation can end up in an incorrect state. TES makes it easy to decide if an event is still valid.

The second reason was implied previously; it makes cancellation easier. With just two robots, it is difficult to justify the use of this strategy, but imagine that instead we had a larger number of robots, say, one billion, all of which are spread out on a finite two-dimensional plane. In addition, instead of collisions, let us consider radio communications, which can travel long distances. Determining which robots are in communications range of other robots can require pairwise analysis in the worst case. This is expensive, and we would like to avoid repeating this work as far as possible. The problem is that changing the velocity of any robot means that we need to cancel the radio events which involve that robot. Canceling events is expensive, even if we store pointers to events of interest within each robot (in the worst case, we would have to store a billion events with a given robot). However, since we are using canaries, cancellation becomes a $\mathcal{O}\left(1\right)$ operation as we need only change the canaries associated with the position and velocity of the robot[†]; we do not have to search for and cancel events individually, we just need to update the current canary value of the velocity of the robot in question, which immediately invalidates all events that depend on the old value. Moreover, we do not need to track the events we are going to cancel; each robot might not know what events

[*]This assumes that there does not exist an auxiliary data structure which tracks events that may need to be cancelled.

[†]There is the obvious issue that we must still examine every event to see if it has been cancelled yet or not, but if an event's validity depends on a small number of datum (where the number is upper bounded by some constant), then examination itself is a $\mathcal{O}\left(1\right)$ operation.

are interested in it at all. The only cost incurred happens when events are examined for execution, which requires comparing the stored values of the canaries with the current values.

The rest of this technical note is organized as follows. Section 2 discusses some alternative methods to TES. Section 3 discusses the author's experiences using the TES, including some heuristics that may be of use in limiting the memory usage of the priority queue when using it. Section 4 finishes with remarks on how the technique could be further improved in the future.

2. Related Work, and the Author's Experiences

Prior to discovering the TES, the author experimented with a number of other well-known methods for developing a simulation engine. Some of the approaches that were tried are discussed in this section.

2.1 Discrete Time

Discrete time step simulators generally have two global states, *now* and *later*. The simulator performs a simple update operation, calculating the *later* state from the *now* state by determining what changes occur over some time delta. The primary choice in these simulators is whether the time delta is fixed in duration, or if it is dynamically adjusted.

These types of simulators have a number attractive advantages:

1. They are simple to understand, implement, and debug.
2. They have a strong separation between the *now* and *later* states, which makes parallelization simpler.
3. They are easy to defend when publishing results.

Parallelization is simple because the *later* state values can only be affected by the *now* state values, which means that every state element that must be calculated for the update can be calculated in isolation from any other element. This is in contrast to techniques such as the well-known Time Warp method (see[3]) that optimistically executes events, and then rollback the results using anti-messages to rollback state after the fact.

The disadvantage is the tension between fidelity and speed of execution caused by the time steps themselves. As noted in Van Verth and Bishop[6](p. 510), if our time steps are too large relative to the speed of the objects within our simulation, there will be important interactions that occur between *now* and *later* that are not correctly simulated. A method for overcoming this shortcoming is to calculate bounding volumes over the objects, and then to sweep or extend the bounding volume along the trajectory that the object will take between *now* and *later*. Bounding volumes that intersect indicate objects that may intersect, and which require special handling. The difficulty with this technique is that as the duration between *now* and *later* increases, so does the volume of the bounding volumes, and therefore the probability that the number of interactions that need to be checked increases.

Nevertheless, due to their simplicity and utility in many areas of interest, discrete time step simulators are widely used, including by the author in his first two simulator designs for his doctoral work, which are described in Section 2.1.1 and Section 2.1.2.

2.1.1 Fixed Time Step

The first version of the author's simulator used a fixed time step system. Testing and performance tuning revealed that there could be long periods of boring time, when nothing of interest in the evolution of the simulation occurred[*]. Given that a single simulation would take place over the period of a simulated day, doing no work while simulating gigahertz radios would still require weeks of wall-clock time to execute. Actually updating all state in the simulator between time steps when there were numerous robots and peers required significantly more time, which meant that individual experiments could take months to execute, all because of the boring time steps.

2.1.2 Dynamically Adjustable Time Step

The second version of the simulator was a modest evolution of the first, where a set of rules were enumerated that described when the simulation was statically known to be boring. The simplest rule was that if all robots were beyond communications range of one another, then the radio communications were not calculated. During these ranges of boring time, the simulator adjusted to take much larger time steps.

[*]Since this was a fixed time step simulator, the sizes of the time steps had to be chosen to capture the shortest duration event of interest. For the research in question, this meant that the duration of a time step was $1\,\mathrm{ns}$.

While these rules were easy to implement on top of the first version of the simulator, they did not produce the hoped-for performance gains. This was because the author's research was focused on improving communication using robots as routers and couriers, and the behaviors that were studied naturally discovered that forming mesh networks that covered as many of the stationary peers as possible was the most efficient at improving communications. So while those simulations that involved very few mobile robots and widely dispersed immobile peers did improve their runtime performance, this method did little to improve those simulations where a significant fraction of the simulation time included at least one pair of actors that were in radio communications with one another, which forced the simulator into its low-speed, high-fidelity mode.

Worse, because the rules governing when the simulator switched from low-speed to high-speed mode were evaluated at every time step, this version of the simulator was always slightly slower than the first version when operating in the slow mode. This necessitated the move to a GEL-based continuous time simulator.

2.2 Continuous Time and Events

Continuous time simulators do not have fixed time steps. Instead, they calculate when the next event of interest will occur using basic physical laws. For example, using the current velocity and position of an object, it is possible to determine with perfect accuracy where the object will be at any moment in the future (assuming that the object's velocity is constant over the intervening duration). This forms the basis of continuous collision detection algorithms (see[7] for one such type of algorithm), and became the core of the third version of the simulator, which was a complete rewrite of the simulator from scratch[*]. The basics of this type of simulator can be found in various publications.[1,8,9]

Switching to the event-based model of continuous time simulators proved to be an extraordinarily good choice. The boring stretches of time were eliminated completely, leaving only the "important" events to consider. This greatly improved the performance of the simulator, but there was still a cost: all events that were calculated that were not the immediate next event to execute were discarded. Events,

[*]The architectural differences between discrete time simulators and continuous time simulators are so great that virtually no code can be profitably shared between the two types. The time spent learning this fact is part of why the author was motivated to write this technical note: it is the author's hope that it will save someone from making the same mistakes the author did.

such as when a robot would reach its destination, would be recalculated a large number of times even though the robot had not changed its velocity.

This realization lead to the search for methods of avoiding having to redo work, which led to TES as described in Section 1.

3. Experiences

The TES was successfully used in the author's Ph.D. dissertation[10] research efforts when he needed to write a simulator for his work. In brief, the author's Ph.D. research centered around swarms of robots that were tasked with minimizing the maximum latency of messages that were transmitted between specialized, immobile peers within the network. Normally, one would form a mesh network, possibly using some form of distributed minimum spanning tree algorithm,[11] to connect the peers together. The author was interested in how to solve the problem when network partition was sometimes unavoidable, such as when the peers were separated to such a degree that forming a network graph which contained a single component was not possible. In this case, some robots could be tasked to physically carry messages between different network partitions.

The demands of the research meant that the simulator had to cope with very short duration events (such as one might expect when simulating gigahertz frequency radios) to long duration events (such as movement). That said, the types of events did not neatly partition into 'fast and 'slow events; as part of the research, the robots were given multiple radios that they could use concurrently, from kilohertz radios that had long range to gigahertz radios that had very short range. As a result, it was not possible to treat transmission of packets as instantaneous events as the robots could move an appreciable distance between the start of a packet's transmission until the end of the transmission. Moreover, while a packet might be corrupted due to packet collisions, it was still possible for a robot to detect that a channel had been in use and for how long. This information itself was useful for some of the strategies that were analyzed and could cause a robot to change its behavior, even if the packet was corrupted in transmission. Different simulation techniques were implemented and discarded as their performance characteristics were too slow to be of use. Once the author discovered the TES, he was able to write a simulator that had sufficient performance to complete his research.

4. Conclusions and Future Work

In this work, the author described the TES, a novel and simple strategy for determining if a given tentative future event is still valid and should be committed to the global state on its execution date. It was compared and contrasted with other well-known techniques that were used by the author during his Ph.D. research work, providing qualitative assessments and recommendations that other researchers may choose to follow when developing their own simulation engines. The author knows of three possible improvements that could be useful when designing simulation engines.

First, the TES as currently defined is a conservative strategy. By using confluently persistent data structures (see[12] and[13]), it may be possible to make the TES an optimistic strategy similar to Time Warp, but without rolling back state. Instead, events that are still valid at commit time will have their copy of the data structure merged back into the global data structure. All other events will be silently discarded.

Second, employ a concurrent garbage collection thread that proactively searches the GEL for events that are no longer valid, either removing them from the GEL immediately or marking them as invalid so that when they are popped from the GEL they can be discarded immediately.

Third, limit how far into the future the simulation is able to advance in a single step. While this appears to directly contradict the gains that event-based simulators can provide, the advantage gained is in concurrency. With the limit in place, the speed of light ('light-cone) limits the distance an event's cancellation or mutation can affect the state space. Thus, events that are widely separated cannot affect each other and can be safely merged into the global state space without regard to one another. While this may seem to be a useless limit given how fast the speed of light is, if actions are limited to within a certain range of an actor (e.g., with the range of what an actor can touch), the light-cone of the actor is defined by the actor's speed. This can significantly increase concurrency.

Lies, Damned Lies, and Statistics

— popularized by Mark Twain

Finally, a comment about the lack of plots, tables, and other quantitative analysis in this work. While it is normal to make claims about performance and general utility in a work such as this one, the author wishes to be extremely cautious in making broad statements about the applicability of this technique. It is too easy to provide misleading plots and claims of speedups that may not transfer well to other problem domains or simulation platforms. In the author's opinion, the TES should be kept as a potential tool in a researcher's toolbox, but before it is applied, a researcher must evaluate it to estimate its utility in a given problem domain. It is for this reason that the author has deliberately chosen to avoid giving what could be misleading metrics regarding the use of the TES.

5. References

1. Jha V, Bagrodia R. Simultaneous Events and Lookahead in Simulation Protocols. ACM Transactions on Modeling and Computer Simulation. 2000;10(3):241–267.

2. Lin YB, Lazowska ED. A study of Time Warp rollback mechanisms. ACM Transactions on Modeling and Computer Simulation. 1991;1(1):51–72.

3. Jefferson DR. Virtual time. ACM Transactions on Programming Languages and Systems. 1985;7(3):404-425.

4. Leach P, Mealling M, Salz R. A universally unique identifier (UUID) URN Namespace. RFC Editor; 2005 July. RFC No.: 4122.

5. Dechev D, Pirkelbauer P, Stroustrup B. Understanding and effectively preventing the ABA problem in descriptor-based lock-free designs. In: 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing; p. 185–192.

6. Van Verth J, Bishop L. Essential mathematics for games and interactive applications. Third ed. Taylor & Francis; 2015.

7. Cai W, Turner SJ, Lee BS, Zhou J. An alternative time management mechanism for distributed simulations. ACM Transactions on Modeling and Computer Simulation. 2005;15(2):109–137.

8. Das SR, Fujimoto RM. Adaptive memory management and optimism control in Time Warp. ACM Transactions on Modeling and Computer Simulation. 1997;7(2):239–271.

9. Fujimoto RM. Research challenges in parallel and distributed simulation. ACM Transactions on Modeling and Computer Simulation. 2016;26(4).

10. Karan C. Comm-Bots - distributed coordination of mobile robot swarms to support message communication. [Ph.D. thesis]. University of Pennsylvania; 2021.

11. Lynch N. Distributed algorithms. Elsevier Science; 1996. (The Morgan Kaufmann Series in Data Management Systems).

12. Brodal GS, Makris C, Tsichlas K. Purely Functional Worst Case Constant Time Catenable Sorted Lists. In: European Symposium on Algorithms; p. 172–183.

13. Driscoll JR, Sarnak N, Sleator DD, Tarjan RE. Making Data Structures Persistent. Journal of Computer and System Sciences. 1989;38(1):86–124.

Glossary

ABA A type of race condition where some datum that is originally in state *A*, is mutated to state *B*, and then back to state *A* before an observer notices that a change took place. (3)

Global Event List (GEL) A sorted list of events that are to be executed in the future. (1, 7, 9)

Tentative Event Strategy (TES) The strategy described within this work. (iii, 1–5, 8–10)

Universally Unique IDentifier (UUID) Strings of 16 bytes that are generated using one of several algorithms defined in[4] and are probabilistically guaranteed to be unique through both time and space across the universe. The algorithms defined within[4] do not require coordination between parties to ensure that each is able to generate their own unique UUIDs. (2)

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
1	DEVCOM ARL
(PDF)	FCDD RLD DCI
	TECH LIB
3	DEVCOM ARL
(PDF)	FCDD RLC IA
	D BARAN
	FCDD RLC I
	B MACCALL
	R HARDY