Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

SIMPLE MOVEMENT AND DETECTION IN DISCRETE EVENT SIMULATION

Arnold H. Buss

MOVES Institute
Naval Postgraduate School
700 Dyer Rd.
Monterey, CA 93943 U.S.A

Paul J. Sánchez

Operations Research Department
Naval Postgraduate School
1411 Cunningham Rd.
Monterey, CA 93943 U.S.A

ABSTRACT

Many scenarios involving simulation require modeling movement and sensing. Traditionally, this has been done in a time-stepped manner, often because of a mistaken belief that using a pure discrete event approach is infeasible. This paper discusses how simple motion (linear, uniform, two-dimensional) and simple sensing can be modeled with a pure Discrete Event approach. We demonstrate that this approach is not only feasible, it is often more desirable from several standpoints.

1 INTRODUCTION

Entity locations often play a key role in many simulation models, such as combat simulations. In many cases a Discrete Event Simulation (DES) method of time advance is rejected in favor of a time-stepped approach which, although superficially more intuitive than DES, gives rise to many modeling difficulties, artifacts, and limitations. There are many advantages to adopting a discrete event world view. The fact that movement and sensing can be successfully modeled using DES is not widely known. This paper presents one way that this can be done in a simple manner.

At first glance, modeling movement seems to present a challenge to the discrete event approach, since the state of an entity in motion (its location, for instance) is in constant change when an entity is in motion. This difficulty is overcome by the notion of implicit state. An implicit state is one that is not explicitly stored in state variables (instance variables in an object-oriented framework) but rather can be implicitly determined from other state variables. An entity that moves in uniform, linear motion can have its position modeled by implicit state in that its position is not stored as an instance variable but is computed "on demand," as described below. The implicit state of position is determined from three explicit state variables: the entity's position when it started the move, the time it started the move, and its velocity vector.

Recall that in DES, time does not advance in regular intervals or steps. Rather, the simulated time is moved to the time of the next occurring event which is then processed; first the model state is changed, then event cancellations, if any, performed, and finally further events, if any, are scheduled. In between the occurrence of events, there is no change in the value of any state variable.

We begin our discussion with a description of uniform linear motion in the following section. We will then consider the simplest kind of sensing, the "cookie-cutter." A cookie-cutter sensor sees everything that is within its range R, and must be notified at the precise time a target enters it range. In a time-step simulation, cookie-cutter detection is very easy. Simply compute the distance between the sensor and the target at each time step. If the target is within distance R of the sensor, then a detection occurs. The precise time of detection cannot be determined, of course – we know it only up to the resolution of the time step chosen.

Subsequent sections will present more complex detection, give an Event Graph implementation using the LEGO (Listener Event Graph Objects) framework, and briefly discuss some specific scenarios in which this approach has proved fruitful.

2 SIMPLE MOVEMENT

The simplest possible movement is uniform, linear motion. A moving entity starts its move at some initial position x at time t_0 and begins moving with velocity v. Thus, the location of the entity at time t is $x+(t-t_0)v$. Equivalently, the location of the entity s time units after it began its movement is $x+sv$.

In a DES model the location of moving entities is modeled using implicit state, rather than explicit state, as mentioned above. Rather than storing the current location of the entity at all times, enough information is stored so

that the current position can be computed easily whenever desired using "dead reckoning." For uniform linear motion, it is enough to store: (1) the initial position x (i.e. the location of the entity just prior to when it started moving); (2) the velocity vector v; and (3) the time it started moving t_0. The equations of motion of the previous paragraph are then applied whenever the position is needed within the model. Note that since there is no explicit location state, state updates are only required when the velocity vector changes.

The coordinates and velocities of the entities are all in some common base coordinate system, so the motion represented above can be considered absolute motion in the base coordinates. Often it is desirable to consider location and motion relative to some particular entity's coordinates. In that case, the locations and velocities can be represented relative to that entity's coordinates. For most purposes the entities' coordinate systems may be considered to be simply a translation of the base coordinate system. Thus, an entity at position y in base coordinates is at position $y - x$ in the coordinates of an entity located at position x in the base coordinate system. Relative velocity is equally simple for uniform linear motion. Suppose the equations of motion for two entities are given by $x_i + tv_i, (i = 1,2)$. Then in the coordinate system of entity 1, the motion of entity 2 is given by $(x_2 - x_1) + t(v_2 - v_1)$. Thus, relative to the first entity, the motion of the second is uniform and linear with starting position $x_2 - x_1$ and velocity $v_2 - v_1$.

2.1 Modeling Acceleration and Turning

Using constant linear motion may seem overly simplistic and restrictive at first glance. If a more detailed movement model is desired, then more complex equations of motion may be used. For example, equations using constant acceleration (rather than constant velocity) are easily developed and can be used in place of the constant velocity equation of the previous section.

However, it has been the authors' experience that linear motion can provide a wide range of possibilities. For example, acceleration and turning can be modeled using a piecewise linear approximation to smooth curved trajectories. For all but the most detailed engineering models, this turns out to be a sufficient level of detail.

2.2 Managing Movement

The model of simple movement described in the previous section is useful for capturing the effects of very simple maneuvers, that is, from a single point to a single destination. In practice models typically require slightly more complex rules of movement.

Among the simplest movement rules is one in which there is a list of waypoints, the rule being to travel to each waypoint in turn, stopping at the last one. A similar movement rule follows the waypoints, except after the last one is reached, the entity moves to the first waypoint again, and continues cycling through the waypoints. This last movement behavior is one an observer on patrol might follow. Another patrolling type of behavior is to choose the next waypoint at random after each one is reached. This is related to the "random" search pattern that is extensively used in search theory.

Now that we have shown how entities can be modeled using a pure DES approach, we turn to a simple way of modeling detection using DES.

3 SIMPLE DETECTION

The simplest kind of detection is the cookie-cutter sensor. Consider the most basic scenario, consisting of one stationary sensor and one moving target. We will consider the sensor to be located at the origin in a two-dimensional coordinate system. At time 0 the target starts at point x (relative to the sensor) and proceeds with constant velocity v (again, relative to the sensor). This situation is illustrated in Figure 1. The problem is to determine the time td at which the target enters the sensor's range. Note that the location of the target at the time of detection is given by $x + tv$.

3.1 Computing Detections

Since detection occurs whenever the distance between the target and the sensor is exactly R, the time at which this occurs is the solution to the equation

$$\|x + tv\| = R. \tag{1}$$

Equivalently, t_D is the solution to

$$\|v\|^2 t^2 + 2(x \cdot v)t + \|x\|^2 = R^2 \tag{2}$$

where "·" represents the vector inner product and || || is the length of a vector. Equation (2) is a quadratic in t, so the solutions are given by Equation 3 (Buss 2000):

$$t = -\frac{x \cdot v}{\|v\|^2} \pm \frac{\sqrt{\|v\|^2 (R^2 - \|x\|^2) + (x \cdot v)^2}}{\|v\|^2}, \tag{3}$$

provided the expression under the radical is non-negative. If the target starts out of range but is eventually detected, then the solutions in Equation 3 are both real and positive. This is the situation depicted in Figure 1. In this case, the smaller of the two solutions is t_D, the time of detection, and the larger solution t_E is the exit time, the time when the target leaves the sensor's range. At time t_D, therefore, the target will be at location EnterRange and at time t_E it will be at location ExitRange.

The expressions in Equation 3 can be used to schedule the time that the target enters the range of the sensor (the "EnterRange" event in Figure 1) as well as the target's exit time (the "ExitRange" event in Figure 1). The calculations above assumed that the starting time was 0.0. In general, the interpretation of t_D and t_E would be the amount of time elapsed after the target started moving. This fits nicely with DES, since events are scheduled after a time delay and, in general, relative times are easier to model than absolute times.

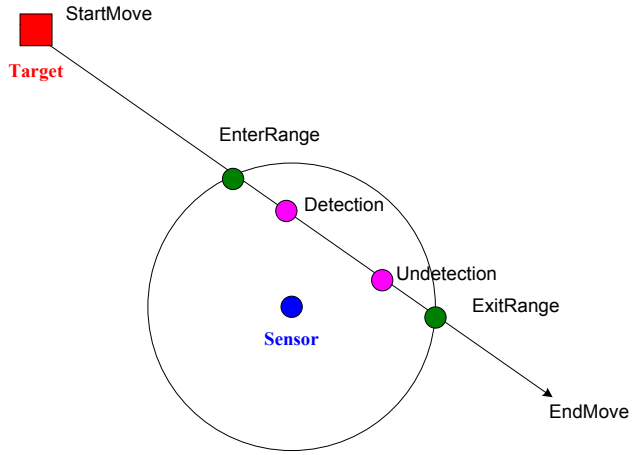

Figure 1: Cookie-Cutter Detection: The Basic Scenario

Of course, Figure 1 is a canonical depiction of an interaction in which the target starts outside the sensor's range and subsequently enters it. In general, the target may also miss the sensor's range altogether or be starting inside the sensor's range when it changes its movement state. Figure 2 shows these possibilities with various starting points of the target, labeled A-D. The possible outcomes may be summarized in terms of the roots of Equation 3 as follows:

- *Both roots positive (A)*. The sensor's range will be entered after a delay of the smaller root and exited after a delay of the larger root. In Figure 2, this corresponds to a target starting at point A heading through C.
- *One positive and one negative root*.(B) The target is already within the sensor's range and will exit after a delay of the positive root. In Figure 2, the target starts at B and proceeds through C. In case of equality of the roots, the target will be on a course tangent to the range ring.
- *Both roots negative (C)*. The target is outside the sensor's range and is moving away from the sensor. The target will never enter the sensor's range. In Figure 2, the target starts at point C and heads away from the sensor.
- *No real roots (D)*. The target will never enter the sensor range. In Figure 2, the target starts at point D and proceeds in a direction which completely misses the sensor's ring.

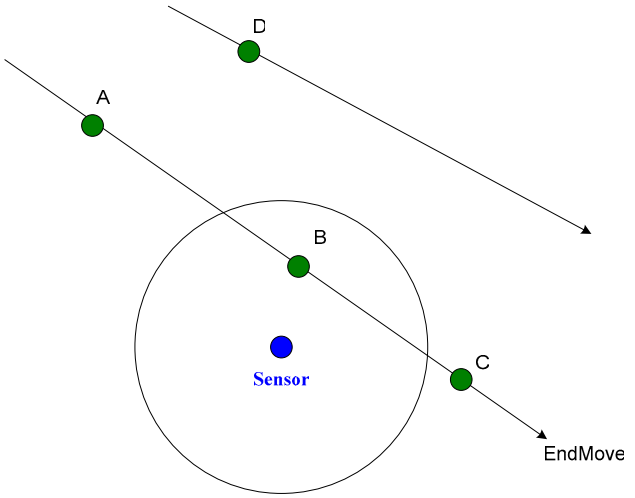

Figure 2: Cookie-Cutter Detection: All Possibilities

All the above cases assume, of course, that the target will not stop, change direction, or change speed. If any of these events occur, the results must be recomputed. Any events that had been scheduled based on the original computations are of course invalid and must be canceled and new events scheduled, if necessary.

The simple scenario described above was one in which the sensor was stationary at the origin and the target was moving. If this approach was only capable of modeling such situations its utility would be extremely limited. Although stationary sensors, such as a ground radar facility, are well-modeled as described above, typically in a simulation both sensors and targets are in motion. Alternatively, the trigger for the interaction could be a sensor that starts moving towards a previously undetected stationary target.

3.2 Scheduling Detections

The methodology outlined in the previous section covers the two key events that trigger and end a possible detection of the target by the sensor. The problem now becomes one of scheduling the Detection and Undetection events in Figure 1.

3.2.1 Cookie-Cutter Detection

The simplest possible sensor is the so-called "cookie-cutter" that detects all targets within its range and cannot detect any target outside its range. The cookie-cutter sensor is easily modeled by scheduling the Detection event with a zero delay from the EnterRange event and the Undetection event with a zero delay from the ExitRange event.

Although exceedingly simple, the cookie-cutter sensor is useful as a starting point for simulating more sophisticated sensors. In general, the sensor's maximum range can be seen as the "cookie" so that the detection algorithm is simply triggered when a target enters the range. The timing of the entry and exit would be determined by the cookie-cutter algorithm described in this note. These entry and exit times bound the range of possible detection times, with the actual time determined by the particular model used for detection. For example, suppose that the time to detect a target after it enters a sensor's range is exponentially distributed with mean μ. This sensor could be simulated by scheduling the actual detection with an exponential delay following the event that the target enters the range. If the target exits the sensor's range before that time has elapsed, then that detection must be canceled, of course. Alternatively, we need not schedule the detection at all if the time to detect is greater than the time the target exits the sensor's range. These possibilities will be explored further below.

3.2.2 Constant Rate Detection

Consider a time-stepped model of detection with the following rule: as long as the target is within a sensor's range, every Δt times units there is a constant probability p that it will be detected. Although this rule is exceedingly simple, it is nevertheless more complex and slightly more realistic than the cookie-cutter sensor rule. For example, with a cookie-cutter, every detection will occur at the sensor's maximum range, whereas with this detection rule they will occur strictly inside the maximum range.

Converting this simple time-stepped rule to a DES approach consists of determining the probability distribution of the time between when the range is entered and the detection occurs (see Figure 1). In this case, the detection attempts are a sequence of Bernoulli trials with identical probabilities, so the number N of detection attempts until first detection is a geometric random variable with parameter p. Thus, the time to detection is exactly $N \cdot \Delta t$, where N is a geometric(p) random variable. In this case, the DES formulation is exact with respect to the original time-step rule.

This DES formulation requires two parameters, Δt and p. The parameterization can be simplified slightly by approximating $N \cdot \Delta t$ with an exponential random variable with mean $\cdot \mu = \Delta t / p$.

3.2.3 General Approach using Instantaneous Detection Probability

The exponential approximation in the previous section can be viewed as the limit of the geometric random variable as Δt approaches zero, with the mean time to detection held constant. In general, if the instantaneous detection probability at time t is given by γ(t), then the probability distribution of the time to detect, T, can be obtained using the complementary cdf (Wagner, et al. 1999)

$$\Pr\{T > t) = \exp\left(-\int_0^t \gamma(u)\,du\right). \quad (4)$$

Note that if the instantaneous detection rate γ(t) is constant, that Equation 4 reduces to the complementary cdf of the exponential distribution, hence the term "constant rate" when applied to that particular rule.

3.2.4 Meta-Modeling Approach To Detailed Detection Algorithms

In general, an exact probability distribution cannot be obtained for a particular detailed detection algorithm. In these situations an approximate implementation can be estimated empirically. The idea is to treat the detection algorithm, however complicated, as a "black box" and apply a meta-modeling approach. Using an implementation of the detailed detection algorithm, a series of experiments are conducted with the sensor trying to detect a target under various conditions. The times to detection are recorded for each parameter setting and interaction. Then a statistical model is fit to these data, with the independent variables being the different parameters such as geometry, target and sensor state, environment, etc. Assuming a reasonable fit of this meta-model, the DES implementation uses the fitted model to generate the time to detection whenever the range of the sensor is entered.

This approach is being applied using the Acquire algorithm (ACQUIRE 1995), as implemented in CASTFOREM (CASTFOREM 2001) and COMBAT^{XXI} (COMBAT^{XXI} 2004) as the detailed detection algorithm to produce a DES. Although Acquire itself is a reasonably fast algorithm, its implementations involve each sensor invoking it many times throughout the simulation. In a lower resolution DES model, the additional computational effort is not appropriate.

The result will be a method that can be utilized in DES models, resulting in faster runtime and at least a first-order similarity to their Acquire implementation.

4 IMPLEMENTATION

We now give a brief overview of an implementation of the ideas presented. This implementation is based on the Simkit package (Buss 2002), a Java-based library that supports creating component-based DES models. Simkit is based on Schruben's Event Graph methodology (Schruben 1983) for the design of its components. Event Graphs describe a DES by specifying a directed graph in which each node specifies a state transition and each directed arc specifies a scheduling or canceling relationship between events (Schruben 1983). Component interactions are specified us-

ing "LEGO" connections (Buss and Sánchez 2002), which are based on Listener patterns (Buss 2002).

4.1 Mover Component

A Mover component is based on an equation of motion *x(t)* that describes the location of an entity at any point in time. In the previous sections, we have used the uniform linear motion equation to describe the entity's location, but in general any such equation can be used to model movement. The crucial distinction between this DES approach and the traditional time-step approach is that an event is only scheduled when the entity changes its movement state. In a time-stepped model, the entity's state is updated every time step regardless of whether the entity's equation of motion has indeed changed or not.

Since a DES state can only change when an event occurs, the entity's location cannot be part of its state. Instead, the initial conditions of the equation of motion are the quantities that remain fixed throughout a given maneuver, so those quantities are what defines the DES state of the moving entity. Thus (x_0, v, t_0) are the states for a uniform linear mover. For a more complex equation of motion, other state variables may be required, since the initial conditions will differ. Given these three values, the location of the entity at any point in time can be exactly computed

The Event Graph for the Mover component is shown in Figure 3.

Figure 3: Mover Event Graph

The Mover component has parameters that include the maximum speed. The basic command is to tell the Mover to move to a given destination at its maximum possible speed, *s*, which triggers the StartMove event. The required velocity is computed along with the time required to perform the move, giving the delay t_M in Figure 3. The current location is saved in x_0, the current simulation time is saved in t_0, and the velocity v is computed by normalizing the vector difference between the destination and the current location to have length s. Finally, the EndMove event is scheduled to occur with a delay of t_M. At any time between the occurrence of the StartMove event and the EndMove event, the Mover's actual location is determined by the dead reckoning calculation described earlier. The EndMove event sets the current location to the destination and the velocity vector to 0.

This simple component is sufficient to implement the DES approach to movement outlined in the previous sections. We now turn to issues involved with implementing the interactions between sensors and targets, which are considerably more complex.

4.2 Sensor Component

The Sensor component has two functions: to maintain a list of contacts, targets that have already been detected, and to be a holder of parameters needed for the detection algorithm used. The Sensor therefore only has two events: Detection and Undetection, shown in Figure 4.

Note from Figure 4 that there are no scheduling arcs in the Sensor component. That is because the Sensor's events are not scheduled by the Sensor itself, but are "heard" from another object called the Mediator, described below. A given detection algorithm is not implemented in the Sensor, but in an instance of the Mediator.

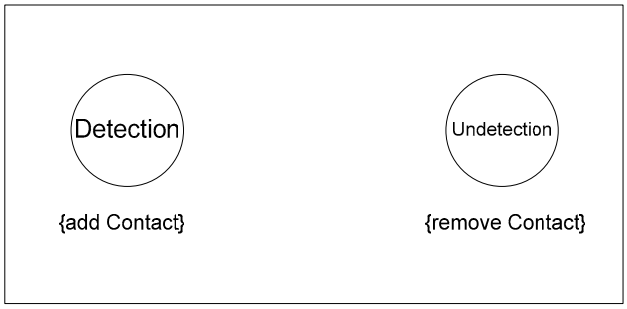

Figure 4: Sensor Event Graph

4.3 Listener Patterns

Simkit implements a listener pattern called the "SimEventListener Pattern" (Buss 2002, Buss and Sánchez 2002). A simulation component that is interested in responding to simulation events that occur in other components is registered as a SimEventListener to those source components. Whenever a simulation event occurs in a source component (i.e. an event scheduled by that component is processed by the Event List), after the scheduling component executes its state transitions and schedules its events, all listeners are notified of the event. If the listener has an event of the same name and signature, it is executed as if it had been explicitly scheduled. If no matching event is found, then nothing happens. The only difference between a scheduled and a "heard" event is that a heard event is not dispatched to its listeners.

Figure 5 shows a simulation component (Referee, explained in the next section) listening to a Mover component. When the StartMove event in the Mover occurs, the StartMove event in the Referee is also executed. Similarly, when the EndMove event occurs in the Mover, the End-

Move event in the Referee is executed. We will now explain the Referee depicted in Figure 5 in more detail.

4.4 Referee

As long as a target remains outside the maximum range of a given sensor, there is no need to be concerned about any interaction between the two. Only when the range is entered is a detection possible. Similarly, when the maximum range is exited by the target, there is no further need for any sensing interactions to occur. Determining when the events EnterRange and ExitRange occur is the responsibility of some simulation component. For the uniform linear motion considered here, this amounts to simply applying Equation 3, considering all the cases enumerated above. In the implementation, these events must be scheduled by some entity in the simulation. The question is which entity should do this.

It does not make sense for these events to be scheduled by either the Sensor or the Mover in question because the computation in Equation 3 involves "ground truth" data that should not be available to either object. The scheduling of EnterRange and ExitRange events should be done by a third party, which is called the "Referee." An instance of the Referee maintains a list of targets (Movers) and a list of Sensors that could potentially detect those targets. The Referee listens for the StartMove and EndMove events of both the Movers, as shown in Figure 5; it also listens for these events from the Sensors, which is not depicted in Figure 5.

Note that in Figure 5 each scheduling edge with a condition also has a canceling edge, which is not depicted to make the basic logic more clear. The condition (a) is that the target be out of the sensor's range and the new movement state will result in entering the sensor's range t_1 time units in the future. Condition (b) is that the target is inside the sensor's range and its movement state will result in it exiting the sensor's range t_2 time units in the future.

Also, not depicted in Figure 5 is the fact that the signatures of the EnterRange and ExitRange events include arguments of type Sensor and Mover, which are the sensor and target, respectively. Thus, these events have access to the state of the Sensor and the Target that are interacting at that event.

The Referee's only responsibilities are for scheduling (and possibly canceling) EnterRange and ExitRange events. Only uniform linear movement is supported by Simkit's default implementation; modeling another equation of motion would require implementing a different Referee to override the default behavior.

The Detection and Undetection events are not scheduled by the Referee but by one of another collection of third-parties called Mediators, which are discussed next.

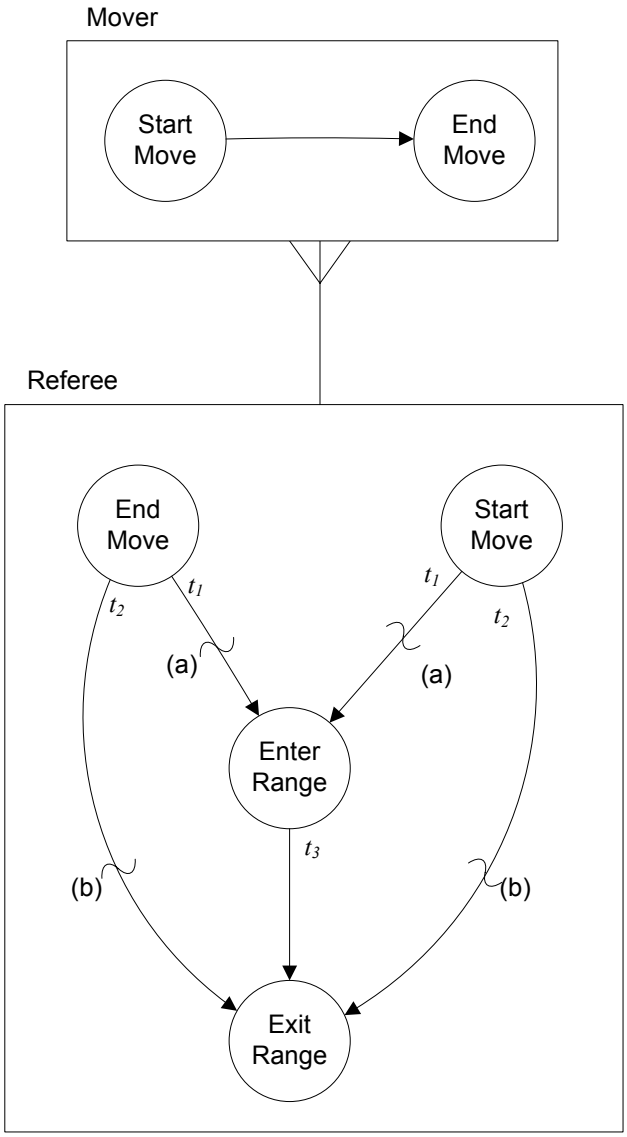

Figure 5: Referee Event Graph Listening to Mover

4.5 Sensor-Target Mediators

Just as with the Referee, implementing a detection algorithm in either a Sensor or a Mover does not make sense, because the algorithm requires ground truth information that should not be available to entities of either type. Furthermore, since many different detection algorithms can be supported, implementing them in the Referee would require re-writing the Referee any time a new detection algorithm was to be added. Therefore, another collection of third-parties is given responsibility for the Detection and Undetection events - the Sensor-Target Mediator, or just Mediator for short.

The Event Graph for the Mediator is shown in Figure 6 below. An instance of the Mediator listens to the Referee

for EnterRange and ExitRange events. As shown by Figure 6, the EnterRange event computes the time until detection, t_D, and schedules the Detection event. Similarly, the Detection event schedules the Undetection event. An ExitRange event, which will be heard from the Referee, will cancel any pending Detections or Undetections and schedule an Undetection immediately (See Figure 6). The Detection and Undetection events are heard by the appropriate Sensor (See Figure 4), which maintains responsibility for the contacts that have been detected.

Each Mediator is a very small class, since only the EnterRange and ExitRange events have non-trivial implementations. Indeed, for most algorithms the ExitRange event consists solely of the scheduling and canceling arcs shown in Figure 6. As with the Referee, the signatures for EnterRange and ExitRange include arguments of type Sensor and Target. Thus, these events will also have access to parameters and state variables for both entities.

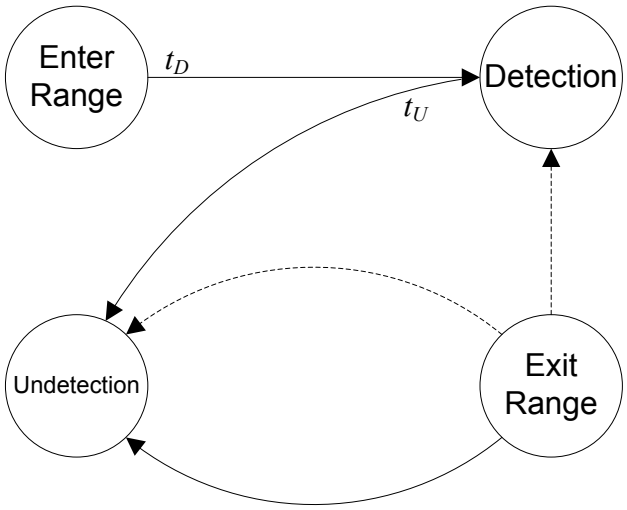

Figure 6: Mediator Event Graph

Each detection algorithm, therefore, has a Mediator class that implements that algorithm, and each simulation run will have exactly one instance of each type of Mediator. Typically, it will be matched to a Sensor class that defines parameters needed for the detection algorithm.

The EnterRange event is responsible for scheduling the Detection event using whichever detection algorithm it is implementing. Typically it uses parameters and state variables from both the Sensor and the Mover to compute the time until detection, t_D.

The CookieCutter detection algorithm is easily implemented because the time until detection is always 0.0. No additional parameters are required on the Sensor, and the CookieCutterSensor simply has its maximum range as the parameter.

The constant rate detection algorithm described in Section 3.2.2 is implemented by the Mediator having a parameter that generates Exponential(1.0) random variables. The time to detection is thus computed by first generating an Exponential(1.0) random variate and multiplying it by the mean time to detection, a parameter of the ConstantRateSensor class. Since there would be only one instance of the ConstantRateMediator in a given simulation, the generated random variates will be independent.

A more complicated algorithm, such as the Acquire meta-model described in Section 3.2.4, may require parameters on both the Mover and the Sensor, as well as some additional parameters corresponding to the environmental conditions, for example.

5 ANIMATION

The fact that movement and sensing is modeled in a pure DES way does not preclude the display and animation of entities that implement these functions. In fact, animation is very straightforward.

Animation is fundamentally a time-stepped operation, consisting of a sequence of frames displayed in rapid succession. Each mover needs an icon associated with it to draw on the canvas. The animation is performed by periodically scheduling a single recurring event. A listener to that recurring event has the simple logic of redrawing the screen of the canvas. Since simulated time has advanced with each event, and hence each redraw of the screen, each moving entity is drawn in a slightly different location.

One implementation of this approach involves a component responsible to synchronizing simulated time and clock time and another component responsible for displaying the entities that are to be animated. The former component simply schedules an event with a deterministic time between occurrences. The event also sleeps for a predetermined amount of clock time. This component is called a "PingThread" (See Figure 7). An instance of PingThread does not have to be aware of any other objects in the simulation. The animation itself is displayed in a listener object, one example of which is called a "Sandbox." An instance of this component has Movers and Sensors registered with it, with icons specified for each entity. It listens for the "Ping" event and simply redraws its contents when Ping occurs. The location of the icons is determined by each Mover and Sensor's location. Since Δt elapses between each redraw, the entitles appear in slightly different locations with each Ping event, thus creating the animation.

A screenshot of a simple implementation of this is shown in Figure 8. Three different types of sensors are shown, distinguished by the colors of the rings. Each mover implements the constant linear motion described earlier. For this application, the movement rule is "random" – each mover generates a random destination in a square and moves to it. When it arrives at the destination, another destination is generated and moved to.

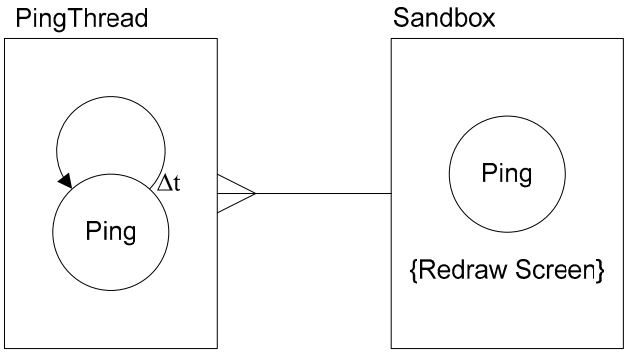

Figure 7: Animation Event Graph

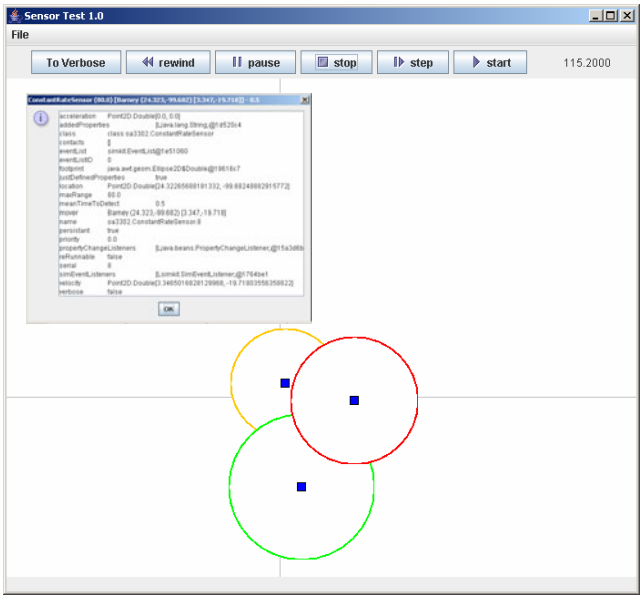

Figure 8: Animation of Movers with Sensors

6 USE IN MODELING AND ANALYSIS

Over the past several years a number of specific scenarios have been modeled and analyzed, primarily as thesis work by students at the Naval Postgraduate. These scenarios have been as diverse as analyzing mine avoidance tactics for autonomous underwater vehicles (Allen 2004), dynamic allocation of weapons and sensors to ground targets (Havens 2002), and waterfront force protection (Childs 2002), to name just a few.

Much of the generic framework for modeling movement and sensing, as presented in this paper, is implemented in Simkit. Using Simkit, students are able to create models in a relatively short time, allowing interesting analysis to be performed that would not otherwise be possible. In most cases, there are no off-the-shelf models that capture the essential features required to answer the study questions, and creating a scenario using an existing combat simulation is often impossible as well. The pure discrete event methodology and the robust LEGO component framework are key features that support these efforts.

A model in support of the United States Army's Future Combat Force has been under development, sponsored by the US Army TRADOC Analysis Center, TRAC-Monterey. This model, called DAFS (Dynamic Allocation of Fires and Sensors), also uses the concepts presented in this paper as part of its model of movement and sensing. DAFS is an example of an emerging approach to low-resolution entity-level simulation (Phillips and Jackson 2005). DAFS has a number of additional features, discussion of which are beyond the scope of this paper.

7 DISCUSSION

We have demonstrated that using a pure discrete event approach to modeling movement and sensing is in fact possible. It turns out that using DES is also more desirable than the traditional time-step approach from several standpoints.

One significant difference is that there is substantially less polling that takes place in the DES approach. Typically a time-step model must poll every moving entity at every time step to determine the necessity of updating its position. The modeler must choose the size of the time step carefully – too large a time step introduces errors in the model, while too small a time step may cause extremely large run times. If sensing interactions occur in the model, all possible interactions need to be considered at each time step. If the number of entities with potential interactions is n, the number of interactions to check for is n choose 2, and the computational complexity is $O(n^2/\Delta t)$.

In contrast, in the DES approach presented here, an entity only updates its state when its movement status actually changes. In most scenarios that is substantially less frequent than the time step. Potential interactions with other entities can be determined when a given entity changes its movement status. The amount of work involved is $O(nm)$, where n is the number of entities with potential interactions and m is the number of movement change events. Note that if T is the run length of the simulation, m is generally much smaller than $T / \Delta t$. Thus, for most models the DES approach will be substantially more efficient.

Additionally, the locations of entities in a DES model can be anywhere, whereas time-step models often use a grid in which the location of an entity can only be specified up to the size of the grid cells. The time step and grid impose limits on the speed of movement. With a time step of Δt and a grid with of Δx, the minimum speed a unit can travel is $\Delta x/\Delta t$. In a pure DES model, units can travel at whatever speed is desired by the modeler.

8 CONCLUSIONS

Despite the widespread impression that modeling of movement and sensing in simulation must be done using time-steps, it is not only possible, but desirable to use a pure discrete event approach. This paper has described the basic methodology for such an approach, developed the necessary equations and shown some of the Event Graph components for implementation, and briefly discussed some specific applications in which this approach has led to analyses that would not have otherwise been possible in the short timeframes available.

ACKNOWLEDGEMNTS

Part of the work of the first author was supported by the US Army TRADOC Analysis Center, TRAC_Monterey. This support is gratefully acknowledged.

The second author gratefully acknowledges the support of the U.S. Marine Corps' *Project Albert*.

REFERENCES

ACQUIRE Range performance model for target acquisition systems. 1995. Version 1 User's Guide, U.S. Army CECOM Night Vision and Electronic Sensors Directorate Report, Ft. Belvoir, VA.

Allen, Tim. 2004. 2004. Using discrete event simulation to assess obstacle location accuracy in the REMUS unmanned underwater vehicle. Masters Thesis, Naval Postgraduate School, Monterey, CA.

Buss, Arnold H. 2000. Simple movement and detection in discrete event simulation. Class notes.

Buss, Arnold H. 2002. Component Based Simulation Modeling With Simkit. *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

Buss, Arnold H. and Paul J. Sánchez. 2002. Building complex models with LEGOs (listener event graph objects). *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

CASTFOREM User input guide. 2001. US Army TRADOC Analysis Center, White Sands, NM.

Childs, Mathew. 2002. An exploratory analysis of water front force protection measures using simulation. Masters Thesis, Naval Postgraduate School, Monterey, CA

COMBAT^{XXI} Programmers manual. 2004. US Army TRADOC Analysis Center, White Sands, NM.

Havens, Michael. 2002. Dynamic allocation of fires and sensors. Masters Thesis, Naval Postgraduate School, Monterey, CA.

Phillips, Donovan, and J. Jackson. 2005. Using a low resolution entity level modeling approach. *Phalanx*, Military Operations Research Society, Alexandria, VA.

Schruben, Lee 1983. Simulation modeling with event graphs. *Communications of the ACM*. 26: 957-963.

Wagner, Daniel H., W. Charles Mylander, and Thomas J. Sanders. 1999. *Naval operations analysis, third edition*. Naval Institute Press, Annapolis, MD.

AUTHOR BIOGRAPHIES

ARNOLD H. BUSS is a Research Assistant Professor in the MOVES Institute at the Naval Postgraduate School. His interests include component-based simulation modeling, with emphasis on military applications. His e-mail and web addresses are abuss@nps.edu and http://diana.cs.nps.navy.mil/~abuss.

PAUL J. SÁNCHEZ is in the Operations Research Department at the Naval Postgraduate School. His research interests include component-based simulation modeling, object-oriented modeling, and simulation output analysis. He is an avid reader and collector of science fiction, and enjoys riding recumbent bikes around the Monterey Bay area. You can reach him by e-mail at PaulSanchez@nps.edu, and his web address is http://diana.cs.nps.navy.mil/~pjs.