Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

DYNAMIC ALLOCATION OF FIRES AND SENSORS (DAFS): A LOW-RESOLUTION SIMULATION FOR RAPID MODELING

Arnold H. Buss

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

Darryl K. Ahner

U.S. Army TRADOC Analysis Center-Monterey
Naval Postgraduate School
Monterey, CA 93943, U.S.A.

ABSTRACT

High-resolution combat models have become so complex that the time necessary to create and analyze a scenario has become unacceptably long. A lower resolution approach to entity-level simulation can complement such models. This paper presents Dynamic Allocation of Fires and Sensors (DAFS), a low-resolution, constructive entity-level simulation framework, that can be rapidly configured and executed. Through the use of a loosely-coupled component architecture, DAFS is extremely flexible and configurable. DAFS allows an analyst to very quickly create a simulation model that captures the first-order effects of a scenario. Although the modeling of entities is done at a low-resolution, DAFS contains some sophisticated capabilities: within the model, commander entities can formulate and solve optimization problems dynamically. DAFS can be used to explore large areas of the parameter space and identify interesting regions where high-resolution models can provide more detailed information.

1 INTRODUCTION

High resolution entity level simulations are becoming increasingly complex. The rate at which simulation complexity grows often outpaces increases in computing power. While this level of complexity is necessary for certain applications, a lower resolution approach to entity-level simulation may also be necessary. A lower resolution approach can complement existing high resolution simulations creating a more robust modeling, simulation and analysis toolkit. Analysis for concept exploration and studies often involves examining a very large parameter space. Time constraints frequently limit the number of high resolution simulation runs that can be completed resulting in only a limited number of parameters being investigated and limiting the settings of the investigated parameters.

The use of low resolution models for military analysis has been previously discussed by Ahner, Jackson, and Phillips (2006). Low resolution screening tools can help identify parameters and parameter settings of interest. Havens (2002) began the development on one such tool, DAFS, a low-resolution, constructive entity-level simulation framework designed for combat. Jackson and Phillips (2005) lay out this compelling argument for a need for low resolution simulation tools to fill these capability gaps in military simulations.

The DAFS framework consists of a Discrete Event Simulation Model with embedded optimization, Extensible Mark-up Language (XML) input and output modules, and an output analysis package. The simulation model receives scenario inputs from XML files. DAFS uses a model predictive control approach for making decisions by calling an optimization routine to allocate assets based upon current conditions. Data is collected during simulation execution and once the simulation is complete; the XML output is available for processing by an analysis package.

The DAFS framework is designed to provide maximum flexibility. Through the use of an interchangeable component-based architecture, the simulation provides the user extensive ability to modify entities, configurations, simulation parameters, and data output. DAFS is open source and is made widely available for user customization.

DAFS is a combat simulation that models BLUE friendly forces against RED enemy forces. Because it uses a low resolution approach, DAFS runs fast and is relatively easy to set up. In addition, DAFS' low resolution models use data derived from high-resolution models enabling analysts to trace DAFS inputs back to accepted models and data.

In the remainder of this paper, we will describe the major components of DAFS, the structure of DAFS input, the embedded optimization in DAFS, DAFS unique low resolution approach derived from a high resolution algorithm to construct representative probability distributions, and finally, describe a DAFS run through event graphs.

2 HIGH RESOLUTION VS. LOW RESOLUTION APPROACHES

For the purposes of this discussion, resolution will mean the level of detail at which the various elements in a model are modeled as well as the level of detail of algorithms used to drive the model (movement, sensing, line-of-sight, etc.). "High resolution" means that these elements are modeled at a very fine level of detail, whereas "low resolution" means that there is considerably less detail. For example, a high-resolution model that included tanks might include attributes such as its weight and its three-dimensional geometry, and might also explicitly represent the individual members of the tank crew as well as very detailed sensing algorithms to represent the tank's various sensor packages. A low-resolution model, on the other hand, might represent the same tank as a point on a two-dimensional map with attributes for its maximum speed, loaded munitions, and a rough representation of its sensor capabilities. Similarly, a high-resolution line-of-sight algorithm might frequently compute the direct line-of-sight between all pairs of entities, whereas a low-resolution algorithm might only consider the events that line-of-sight was gained or lost, with the times between modeled probabilistically.

Often it is asserted, explicitly or implicitly, that the level of resolution a simulation model must have is an absolute quantity. The "high-resolution" approach typically attempts to model every element and entity with many attributes and to model the dynamics and interactions to a very fine degree. The consequences can have significant impact on the ability to conduct analysis to produce meaningful recommendations in a timely manner.

The high level of fidelity in representing entities imposes a significant data burden on the analyst. Not only do data have to be produced to fill in each attribute, but the resulting memory footprint when running the model can be substantial. The high-resolution algorithms implemented often are very time-consuming, thereby substantially increasing the length of simulation runs, often to the point where no more than a few "production" runs can feasibly be performed for a study.

DAFS is an example of a low-resolution model, and henceforth in this paper we will only consider the low-resolution approach to modeling entities as it applies to DAFS. Before discussion that approach, it is first necessary to cover Event Graph Methodology, upon which the DAFS entities and algorithms are based.

3 EVENT GRAPH METHODOLOGY

Events Graph Methodology was introduced by Schruben (1983) as a simple, yet powerful, way of representing discrete event simulation (DES) models. A DES model consists of states, instantaneous state transitions (events), and scheduling relationships between events, that is, defining which events are scheduled when each given event occurs. In a DES model, time advances from one scheduled event to another, not in fixed predetermined increments.

Event Graphs are a way of representing the Future Event List logic for a discrete-event model. An Event Graph consists of nodes and directed arcs. Each node corresponds to an event, or state transition, and each arc corresponds to the scheduling of other events. Each arc can optionally have an associated boolean condition and/or a time delay. Figure 1 shows the fundamental construct for Event Graphs and is interpreted as follows: the occurrence of Event A causes Event B to be scheduled after a time delay of t, providing condition (i) is true (after the state transitions for Event A have been performed). By convention, the time delay t is indicated toward the tail of the scheduling edge and the edge condition is shown just above the wavy line through the middle of the edge. If there is no time delay, then t is omitted. Similarly, if Event B is always scheduled following the occurrence of Event B, then the edge condition is omitted, and the edge is called an unconditional edge. Thus, the basic Event Graph paradigm contains only two elements: the event node and the scheduling edge with two options on the edges (time delay and edge condition).

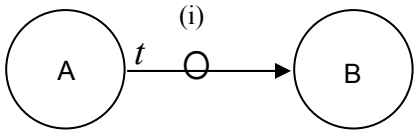

Figure 1: Basic Event Graph Construct

The simplicity of the Event Graph paradigm is evident from the fact that we can represent any discrete event model using only these constructs (Schruben 1983). An advantage of the minimalist approach of Event Graphs is that the modeler can spend more time on model formulation and less on learning the constructs of the paradigm.

4 LOW RESOLUTION MODELING

We will now discuss three of the primary elements of a low resolution, entity level combat model: movement, sensing, and weapons effects.

Intuition may suggest that these must be implemented in a time-step manner. Indeed, an entity in motion, for example, cannot have its position be modeled as a DES state, because its value is continuously changing. Since DES state must have piecewise constant trajectories, location therefore cannot be a DES state. However, it turns out there is an alternate approach that not only is more computationally efficient than time-step, but more accurate in its representation of the precise location of the moving entity.

This approach, using an equation of motion with dead reckoning, is discussed in the following section.

4.1 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 $v_2 - v_1$.

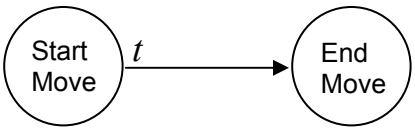

Figure 2: Mover Event Graph

Although it may not be immediately evident, representing movement in a pure DES manner such as this actually can provide a superior model to the traditional time-step approach for entities that move around in a simulation model (Buss and Sanchez 2005). A discussion about the relative merits of the two world views are beyond the scope of this paper. We will therefore confine the claim to the relatively modest one that the DES way of modeling movement is a reasonable one for low-resolution modeling described in this paper. It should also be evident that, barring pathological situations, the DES approach is generally fasters than the time-step approach.

Finally, we note that the approach itself is not limited to linear equations of motion. Indeed, *any* equation of motion in a closed-form can be used in place of the linear equations described above. It has been our experience, however, that linear motion is more than adequate for low-resolution modeling.

4.2 Sensing

A pure Discrete Event Simulation approach to the modeling of sensing starts by changing the fundamental question being asked of the sensor-target interaction. Rather than focusing on the probability of detection as the primary measure, DES sensing is concerned with when a sensor acquires a target, and also when a given sensor loses contact with a given target following acquisition.

It is easiest to start with the simplest situation in which the sensor is motionless and the target initiates a maneuver that will bring it within the sensor's range. The target's motion is initiated by the StartMove event and concludes with the EndMove event

The key events are summarized in Figure 3 (after Buss and Sanchez 2005).

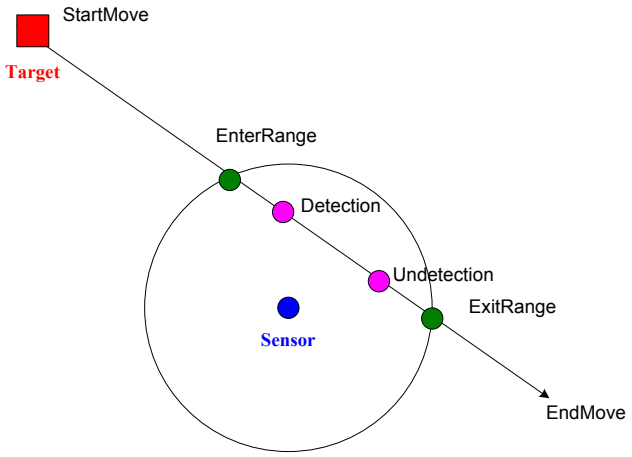

Figure 3: Canonical Event Sequence

The target entity's StartMove event is "heard" by a Referee entity using the SimEventListener pattern (Buss 2002), whereupon the time of the EnterRange event is calculated and the EnterRange event scheduled by the Referee. When the Referee's EnterRange event occurs, the time to Detection is calculated by a Mediator entity. Since different Mediators can exist even for the same Referee in-

stance, there is considerable flexibility in implementing detection algorithms. In principle the Undetection and ExitRange events are distinct, but in practice there exists little data or models on which to make that distinction. Regardless, when the ExitRange event occurs, the sensor cannot possibly detect the target. It is important to recognize that the scheduling of these events does not rely on polling or time-stepping. Rather, each scheduled event is based on a single computation and a single scheduled event for that sensor-target pair.

The simplest example of a Mediator is the CookieCuutterMediator, in which the delay between EnterRange and Detection events is 0.0. Another simple Mediator is based on an exponentially distributed time between EnterRange and Detection. This is roughly equivalent to a sensor that detects the target at a constant rate, and can be used in place of a time-step model in which the probability of detection at each time step is a constant. Finally, a methodology has been developed in which the delay time can be statistically calibrated to the Acquire algorithm (Buss and Sanchez 2005).

4.3 Weapons Effects

Representing weapons effects using a pure Discrete Event approach is similar to representing sensing. The primary focus is actually less on the weapon but rather on the munition, since a given weapon is generally capable of using different types of munitions depending on the circumstances.

A munition is represented as a fast-moving Mover whose EndMove event triggers an Impact event. Both direct and indirect fire munitions are modeled using the same approach. A MunitionTargetReferee first determines the targets that are impacted by the munition. This is determined by the shape of the impact and which entities are within that shape. For each target within the blast area, the actual effect is determined by a MunitionTargetAdjudicator. Like the Referee for sensors, for each target the MunitionTargetReferee chooses the appropriate MunitionTargetAdjudicator, thus enabling differential effects of even the same shot.

Currently, DAFS does not model damage to platforms; rather, they are either dead or alive, so the MunitionTargetAdjudicator's job is simply to determine whether the shot did or did not kill the target. As with sensor Mediators, different algorithms are possible with MunitionTargetAdjudicators. Thus, the probability of killing the target can be a function of the munition type, the target type, as well as the distance of the weapon and the distance of the target from the center of impact.

4.4 Discussion

The pure DES way of modeling these elements enables significant possibilities for improved computational efficiency over traditional time-step approaches.

It should be apparent that the DES approach to modeling movement is much more efficient than the time-step approach under most circumstances. A time-step approach typically must poll each entity regardless of whether it is moving or not. In the DEA approach, a stationary entity requires no computational effort for the movement part of its state, since there are no events on the Event List, as long as the entity remains stationary. Indeed, even for an entity in motion, there is a single EndMove event on the event list. There is no need for polling the entity's state, since it remains fixed until the EndMove event occurs. Generally, the rate at which moving entities change their movement state is orders of magnitude less than a typical time step duration. Only when entities are changing direction or speed every time step will the corresponding DES model be less efficient, and this is a highly unusual situation. Moving entities tend to keep moving according to the same equations of motion for extended periods of time relative to typical time steps.

In modeling sensing there is even more potential improvements of DES to time-step. In a scenario with s sensors and t potential targets, every time step there must be $s \times t$ determinations of detection. In the DES approach, only when a target or a sensor changes movement state does there have to be any computation of EnterRange events or Detections. Furthermore, consider as event for a potential target that changes its movement state. In that case, only the sensors need to be polled about the new detection status; the other targets are irrelevant. Similarly, if a sensor changes its movement state, then all the potential targets must be polled, but the other sensors are not relevant and can be ignored at that event. Thus, for movement state changing events, which are relatively much more rare than time steps, there is essentially an amount of computation that is linear in the number of sensors or number of targets, rather than the product of the two.

We have labeled the way of modeling these three important elements of combat "low-resolution" because of the fact that some elements are not captured in as much detail as in traditional "high-resolution" combat models. If indeed a fine-grained capturing of movement subtleties, such as increased or decreased speed along undulating terrain, is required for the performance measures of the model, then a time-step approach may be the only way to represent it. However, in many cases it turns out that the measures are relatively insensitive to the precise fluctuations in movement, and are relatively unaffected by the somewhat grosser representation of a DES model.

We now turn to some details of the implementation of these concepts in the DAFS (Dynamic Allocation of Fires and Sensors) model.

5 DAFS IMPLEMENTATION

Dynamic Allocation of Fires and Sensors (DAFS) had its origins in a Masters thesis at the Naval Postgraduate School under the sponsorship of the U.S. Army TRADOC Command, TRAC-Monterey (Havens 2002). The initial motivation was to model optimization-based decision rules for allocation weapon platforms to targets and sensors to sensor assignments and evaluate the rules in a combat scenario. The primary focus was on the optimization rules, and the simulation portion was used to adjudicate the outcomes in using a simple combat scenario. In other words, the efficacy of the optimization was determined not by its objective function value but by traditional combat measures, such as probability of achieving objective and loss-exchange rates. Some details of the optimization are presented in the following section.

DAFS is an Open Source model, copyright under the GNU Lesser Public License (Free Software Foundation 2006). The philosophy of the DAFS development team has been to make it freely available, including source codes, with the objective of creating closer ties between developers and potential users. Furthermore, allowing any user access to the source code enables the possibility of users making modifications to suit the needs of a particular study without having necessarily involve the developers. The modular design of DAFS enables rapid modifications to be made and additional features added according to the needs of the study. This is in contrast to proprietary models for which desired modifications require a lengthy and expensive process of negotiations.

The simulation elements of DAFS are implemented in Java using the Simkit DES engine (Buss 2001; Buss 2002). Simkit is itself an Open Source simulation engine designed to enable the ease of building DES models based on Event Graph Methodology. Simkit adds support for the two listener patterns that enable construction of models based on a loosely-coupled component architecture (Buss 2002, Buss and Sanchez 2002). Support for Event Graph methodology and for the Listener Patterns is crucial to implementing the essential elements of moving, sensing, and weapons effects described in the preceding sections.

We now discuss some of the salient classes in DAFS.

5.1 Movement in DAFS

Movement in DAFS is accomplished through the interaction of three kinds of objects: a Mover object, responsible for maintaining the movement state, an instance of a MoverManager, which is responsible for elementary maneuver types, and an instance of a PlatformCommander, that provides rudimentary decision logic. Together instantces of these three classes comprise a basic platform that can move and plan its motion based on simple rules of engagement.

The Mover instance in DAFS models the constant velocity movement described previously. In addition to the StartMove and EndMove events there are methods to stop and to pause the Mover instance. These commands are invoked by the MoverManager instance that is in control of the Mover.

A MoverManager is an implementation of a particular type of rule for maneuver. The overall movement is comprised of a sequence of elementary maneuvers, each executed by the Mover. Each Mover has a single MoverManager that controls its movement at any time, but MoverManager instances may be changed during a simulation run depending on the situation. Each MoverManager however is responsible for only a single Mover instance. A MoverManager listens to its Mover for an EndMove event and then chooses what action to take based on the type of MoverManager it, its parameters, and possibly its own state. DAFS uses three kinds of MoverManagers: PathMoverManager, InterceptMoverManager, and RandomLocationMoverManager.

The PathMoverManager causes its Mover to move sequentially along a predetermined list of waypoints. When each waypoint is reached by the Mover (signaled by its EndMoveEvent), the PathMoverManager sends the Mover to the next waypoint, if there is at least one remaining. If the last waypoint has been reached, the Mover stops. This is the default MoverManager for most DAFS platforms.

The InterceptMoverManager becomes the active MoverManager when there is a desire for the platform to intercept another platform. When active, the InterceptMoverManager computes the intercept point based on the velocities of its Mover and of the target, as well as the desired range of intercept. When the intercept point has been calculated, the InterceptMoverManager instructs the Mover to move to that point. When the intercept point is reached, control is returned to the default MoverManager for that Mover. One use of the InterceptMoverManager in DAFS is when a weapons platform is instructed to engage a target that is currently outside its range. The InterceptMoverManager computes the point for the platform to engage the target and moves it there. Once the point of engagement is reached, what happens next is determined by other factors, depending on what type of platform the Mover is on.

The RandomLocationMoverManager has the following logic. A destination is randomly generated and the Mover is sent to that destination. When the destination is reached, another one is generated according to the same distribution, and the process continues until the platform is instructed to stop or another MoverManager becomes active. A common use in DAFS for the RandomLocation-

MoverManager is for UAV platforms responsible for patrolling Named Areas of Interest (NAI).

5.2 Sensors

Several types of sensors are implemented in DAFS, and the flexibility of the sensor framework allows new types of sensors and sensing algorithms to be easily deployed in DAFS. The three main ones used in DAFS are the CookieCutter, the ConstantRate, and the LowResAcquire sensors. All three utilize the same event-driven framework described in Buss and Sanchez (2005).

The CookieCutter sensor is the simplest, for which the delay between EnterRange and Detection is 0.0. The ConstantRate sensor has a delay between EnterRange and Detection that is exponentially distributed. The LowResAcquire sensor is based on a meta-modeling of the Acquire algorithm and has two levels to its logic. First, the probability that there will be a detection at all in the interaction is computed. A uniform random number is generated to determine whether or not a detection would occur. If not, then nothing further is done for that interaction. If a detection will occur, then the time to detection is generated as a single random variable with a distribution that has been fitted to the parameters of the sensor and the target. That time is used to schedule the Detection event following the EnterRange event. For all sensors the ExitRange and Undetection events coincide.

DAFS uses the Referee/Mediator pattern to implement sensing. The Referee listens for all changes in movement for potential targets and sensors and then schedules (or cancels) EnterRange and ExitRange events as necessary. When EnterRange events occur, the Referee delegates scheduling the Detection events to the appropriate Mediator, based on the type of sensor and type of target. Similarly, ExitRange events are delegated to the appropriate Mediator to schedule Undetection events.

5.3 Weapons

DAFS uses the Referee/Adjudicator approach discussed in Section 4.3 previously. The WeaponsTargetAdjudicator utilizes a LinearKillProbability instance whose parameters are specified in the data input file. This object gives a minimum range, a maximum range, and the probabilities of a munition killing the target at each range. If a weapon's range is between the minimum and maximum ranges, the actual probability of kill for that round is computed by linearly interpolating between the two extreme ranges. If the weapon is outside the range interval, the probability of kill is 0.0. Each munition/target pair can have a different KillProbability, thus giving great flexibility in how munitions affect targets.

Each weapon has a set of potential munitions that can be used. Which munition is chosen for a particular shot is determined by availability and by which is more effective (i.e. has a better probability of kill) against that target.

When a round is fired, DAFS dynamically creates a Munition object, which is actually an extremely fast-moving Mover instance. The time to reach the target is thus explicitly modeled. When the munition impacts, the MunitionReferee determines which platforms are within the effects radius, then delegates the actual outcomes to the appropriate MuntionTargetAdjudicator. This in turn uses the appropriate KillProbability for each munition/target pair to determine the actual outcome of the round.

6 OPTIMIZATION IN DAFS

Periodically in DAFS the fires assignments are updated using a simple optimization. This optimization problem is formulated and solved in an entity called the Constrained Value Optimizer (CVO). When applied, the CVO solution enables the forces in the simulation to revise their collective engagement tactics to increase the near term probability of success.

In the current implementation ,the CVO solves a simple assignment problem:

$$Maximize \sum_{i \in I, j \in J} C_{ij} X_{ij}$$

Subject to:

$$\sum_{j \in J} X_{ij} \le MaxAssign$$

$$\sum_{i \in I} X_{ij} \le MaxCover$$

$$\sum_{i \in I} X_{ij} \ge MinCover$$

$$X_{ij} \in \{0,1\}$$

Where I is the set of available weapons platforms and J is the set of available potential targets at the time the optimization is run, and X_{ij} is 1 if weapon platform i is assigned to potential target j, and 0 otherwise.

The values of the objective function coefficients is determined by another entity called the Value of Potential Assignments (VPA). Different instances of a VPA can be used to produce different objective values to be optimized. The current default VPA computes coefficient C_{ij} as the net expected value of the outcome of the engagement: $p_{ij}V_j - q_{ji}V_i$, where p_{ij} is the probability the weapon will kill the target, q_{ji} is the probability the target will kill the weapon platform, and V_i and V_j are the values of weapon i and target j, respectively.

Currently the CVO re-optimizes periodically according to an input parameter. After the optimization is run, the

CVS gives each weapon platform its assigned targets, which go on the platform's list.

The CVO and VPA allow considerable flexibility in implementing different optimization possibilities in DAFS. The formulation itself can be changed by writing a different CVO class, and the existing VPA can be left as-is. Alternatively, a different scheme for determining the objective function coefficients can easily be implemented by developing a new VPA, without having to necessarily change the CVO formulation. Of course, new versions of both classes could be created if there were a desire to implement an entirely different optimization problem to allocate the weapons platforms.

The optimization is solved in DAFS using the LpSolve library (Lp_Solve 2006). LpSolve is Open Source software that supports formulation and solution of linear and mixed integer programming problems. Although LpSolve is written in C, it comes with a wrapper that uses the Java Native Interface (JNI) to connect with the LpSolve library.

7 INPUT AND OUTPUT

Input to DAFS is currently done using a single XML file. The input file defines which entities are to be created as well as specifying the various types of platforms, sensors, and weapons. Data for detections and munition-target interactions are all specified in the data. Since nearly all the information that defines these attributes is in data, there is considerable flexibility on the part of the user to define new types of sensors, munitions, or platforms.

DAFS output currently consists of two reports. One details all munition/target interactions, listing the time of the engagement, which entities were involved, their respective locations, and the outcome (killed or missed). The second table details all Detection and Undetection events by sensors, listing what time, the platforms involved, and whether the event was a Detection or an Undetection.

These reports can currently be saved to an Access database by clicking on the Save icon in the toolbar. The user is first prompted for a file to save the results in.

It is very straightforward to modify DAFS to produce other reports, but currently it does require modification of some of the DAFS code.

8 THE GRAPHICAL USER INTERFACE

DAFS can be run in command-line mode for multiple replications or using a Graphical User Interface (GUI) to visually observe a single run. Viewing a single run is extremely useful for debugging scenarios and for briefing results.

DAFS uses an Open Source map display application called OpenMap (2006). When DAFS is run in GUI mode, an empty map is displayed, as shown in Figure 4.

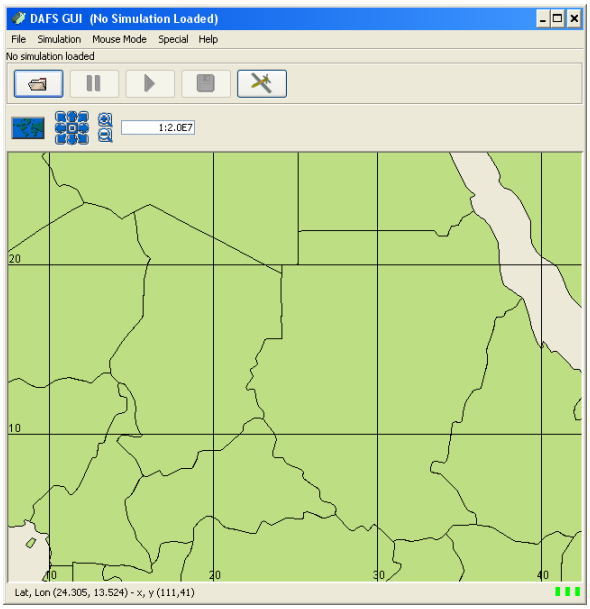

Figure 4: DAFS GUI

A scenario file can be loaded by clicking on the File Open icon on the toolbar. The user can select the desired input file. Then DAFS creates all the platforms specified in that file and displays them in the GUI.

OpenMap has a rich set of mapping features, including zooming and scrolling. Figure 5 shows a scenario in progress in which the map has been zoomed in to get a better view of the battle. The opposing sides are shown in blue and red colored icons, and units that have been killed are shown as an 'X.'

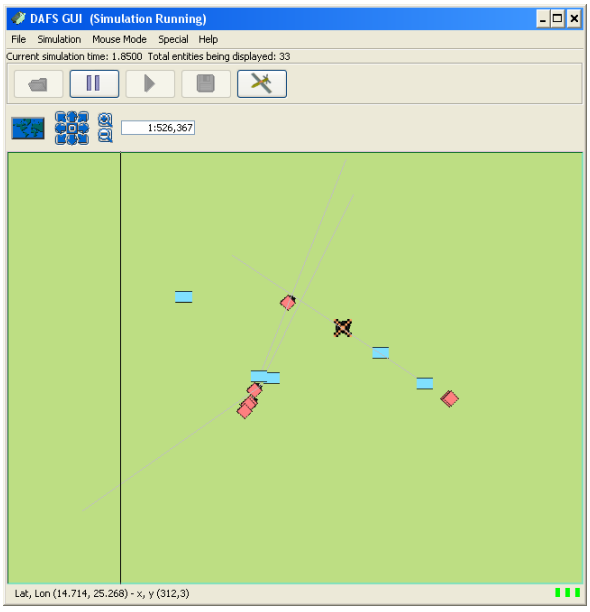

Figure 5: Small Scenario Executing in GUI

9 CONCLUSIONS AND FUTURE WORK

Low resolution entity-level simulation models are a useful addition to the military analyst's toolkit. They address a number of important problems facing the military analyst today. Although not a panacea for all analytical needs, there fill an important gap in the current suite of simulation tools available.

DAFS is an example of such a low-resolution combat model. It has many characteristics that are crucial to a modern simulation tool: rapid construction of scenarios, fast execution times, and flexibility of configuring scenarios. DAFS also incorporates some unique capabilities, such as being able to dynamically formulate and solve optimization problems within the simulation.

Development of DAFS is ongoing. Some of the areas currently being addressed include:

- Improved optimization formulation,
- More user-friendly input,
- Improved sensor allocation methodology,
- Modeling communications networks, and
- Better representation of command and control.

ACKNOWLEDGMENTS

Development of DAFS and of the first author has been supported by the U.S. Army TRADOC. This support is gratefully acknowledged.

REFERENCES

Ahner, D.., L. Jackson, and D. Phillips. 2005. DAFS: A low resolution modeling approach: architecture and implementation. *Proceedings of The 10th Annual International Conference on Industrial Engineering Theory, Applications & Practice*, December 2005.

Buss, A.H. 2001. Discrete Event Programming with Simkit. *Simulation News Europe*. 32/33: 15-24.

Buss, A. 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 A.H. and P.J. Sanchez. 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. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Buss, A. H. and P.J. Sanchez. 2005. Simple movement and sensing in discrete event simulation. *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Free Software Foundation Web Site. <http://www.fsf.org>. [Accessed June 2006.]

Havens, M.E. 2002. Dynamic allocation of fires and sensors. Masters Thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA

Jackson, J. and Phillips, D. 2005. Using a low resolution entity level modeling approach. *The Bulletin of Military Operations Research: Phalanx*, 38-2: 15-26.

Lp_Solve Web Site. <http://sourceforge.net/projects/lpsolve> [Accessed June 2006.]

OpenMap Web site <http://openmap.bbn.com/> [Accessed June 2006].

Schruben, Lee 1983. Simulation modeling with event graphs. *Communications of the ACM*. 26(11): 957-963.

AUTHOR BIOGRAPHIES

ARNOLD BUSS is a Research Assistant Professor in the MOVES Institute at the Naval Postgraduate School. His e-mail address is <abuss@nps.edu>.

DARRYL AHNER is an analyst at TRAC-Monterey. A Major in the United States Army, he received his Ph.D. in Operations Research from Boston University. His e-mail address is <dkahner@nps.edu>.