*Proceedings of the 2007 Winter Simulation Conference*
*S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.*

# USING A LOW-RESOLUTION ENTITY MODEL FOR SHAPING INITIAL CONDITIONS FOR HIGH-RESOLUTION COMBAT MODELS

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

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

John Ruck
Rolands & Associates Corp.
500 Sloat Ave
Monterey, CA 93940, U.S.A.

## ABSTRACT

Determining the initial conditions for high-resolution combat models presents a challenging modeling problem. These initial conditions can have a major impact on the outcome of the analysis, and yet there is a significant difficulty setting those conditions in a manner that spans the important areas of the input factor space. This paper presents a method for setting those initial conditions using a low-resolution, entity-level combat model, Joint Dynamic Allocation of Fires and Sensors (JDAFS). Like its predecessor DAFS, JDAFS models entities on the battlefield, but to a lower degree of detail than most high-resolution combat models. This allows substantial exploration of the input factor space, and can help make the eventual high-resolution simulation runs more effective.

## 1 JOINT STARTING CONDITION REQUIREMENTS

When using high resolution ground combat simulations, scenarios often do not start running in these high resolution simulations on D-day. For instance, if the high resolution starts on D+10, then initial conditions for the high resolution simulations must be developed. The process for setting these initial conditions often has relied on a single Intelligence, Reconnaissance, and Surveillance expert to determine detection and identification percentages. Then, an air campaign expert determines the destruction percentage and dispersion of remaining enemy assets throughout the area of operation. This overall process is difficult to defend to an analysis review board which brings into question the results of the high resolution runs due to the lack of traceability to certifiable algorithms and experimental performance data when setting these initial conditions. A process that is approved by the scenario, intelligence, threats, and Joint community is desired.

The starting condition input parameters that the high resolution simulations require fall into three categories: unit, geographical, and operating environment parameters.

Unit starting conditions consist of location, orientation, and velocity information for each entity in the high resolution model. In addition, the disposition to include current morale, strength, and training may be required. For each unit its current information state or situational understanding must be available for the high resolution model. This understanding may consist of a representation of the ambiguities present in actual conflicts. Each units status of supplies, especially fuel and ammunition, is required to include status until next re-supply. Finally, each units location and disposition of intelligence assets must be provided as starting condition input.

Geographical information that is required as starting condition input is the status of dams, weather, trafficability of terrain, minefields, and contaminated areas.

Operational environment information that is required as starting condition input is civilian distribution, allegiances and attitudes of civilians, economic conditions, religious attitudes, and others.

Within the high resolution simulation these information requirements are critical in the representation of the intelligence preparation of the battlefield that all military units perform prior to major operations. By providing a traceable methodology of determining these initial starting conditions, the high resolution simulations, which are already traceable and whose results are well accepted by senior military leaders, can provide defendable results to analysis review board and senior military leaders that underpin key decisions.

The Joint Dynamic Allocation of Fires and Sensors (JDAFS) simulation helps set the unit and geographical starting conditions for high resolution simulations by using validated algorithms and data. JDAF provides a traceable methodology to justify these initial Joint starting conditions.

## 2 JOINT DYNAMIC ALLOCATION OF FIRES AND SENSORS (JDAFS)

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

The framework of DAFS 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 to be processed 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 an open source simulation that 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 following sections, 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.

## 3 HIGH RESOLUTION VS LOW RESOLUTION APPROACHES

For 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.

## 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 1: 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 faster 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 modeling 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 2. 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 instance, 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.

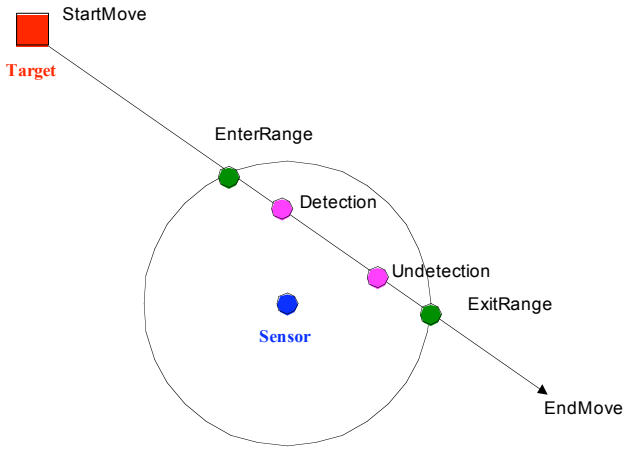


Figure 2: Canonical event sequence (after Buss and Sanchez 2005).

The simplest example of a Mediator is the CookieCutterMediator, 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.

Joint starting conditions requires that JDAFS methodologies are traceable back to approved algorithms and data. JDAFS data and methodologies account for delivery accuracy, target location error, and mean point of impact error for indirect fire weapons and biases, dispersion from movement, and random error for direct fire. These methodologies are derived from U.S. Army Material Systems Analysis Activity approved algorithms to ensure traceability.

### 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 an

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.

### 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 RandomLocationMoverManager 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

JDAFS uses the Referee/Adjudicator approach with traceable data and methodologies 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 NETWORK ENABLED DECISION-MAKING IN JDAFS

Periodically in JDAFS the fires and sensors 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 or sensor platforms and $J$ is the set of available potential targets at the time the optimization is run, and $X_{ij}$ is 1 if weapon/sensor platform $i$ is assigned to potential target $j$, 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.

Currently the CVO re-optimizes periodically according to an input parameter. After the optimization is run, the CVS gives each weapon/sensor platform its assigned targets.

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/sensors platforms.

The optimization is solved in DAFS using the LpSolve library (LpSolve 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.

### 6.1 Weapons Allocation

The fires formulation adds flexibility for the user to determine how fires are allocated. The VPA computes coefficient $C_{ij}$ taking into account the expected value of the outcome of the engagement, communication time, and munition travel time:

$$\left[\left(P_k \cdot V_{enemy} - \left(P_{k,enemy} * V_{friendly}\right) r\right) a\right] e^{(t_{ci} - t_{c\min})c1} \cdot e^{(t_{fi} - t_{f\min})c2},$$

where $r$ is a binary factor equal to 0 if friendly is already in range of enemy, 1 otherwise, and $a$ is a binary factor equal to 0 if munition $P_k$ is not acceptable or effects are not acceptable, 1 otherwise. The first exponential discounts for additional time to send a request to a higher C2 node. The second exponential discounts for choosing a munition that takes longer to put steel on target. Some easy options are $c1$=0, which does not consider communication time or $c2$=0 which does not consider munition time. Additionally, $r$ can be set constantly equal to 0 or 1

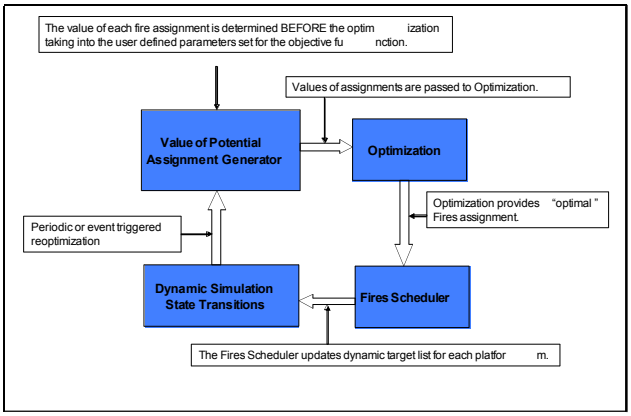


Figure 3: Dynamic fires optimization.

The Dynamic Allocation of Fires and Sensors (DAFS) simulation calculates the value of all potential assignments BEFORE handing them to the optimization to perform optimal matching as shown in Figure 3. This allows the problem to be an integer linear programming problem while maintaining maximum flexibility of the factors that can be considered in the optimization.

The formulation above can bias fires toward using indirect fires when possible since it discounts for moving direct fire platforms within enemy weapon range. It also accounts for time delays in requesting fires and time delays once the weaponeering decision has been made. Additional terms that can be addressed are:

- Collateral Effects Risk Reduction
- Controlled Supply Rate and Munition Resupply Forecast
- Commander's Preference
- Designator Available
- Designator Accuracy
- Fratricide Risk Reduction (currently on or off; potential for discount function)
- Mission Prioritization (i.e., time sensitive, High Payoff Target)
- Target Posture

A key part of fires representation is the Attack Guidance Matrix (AGM). The AGM will not be stove piped as it is in our current models. All munitions/target pairing that meet target location error considerations will be considered. If a tie exists, currently, the considerations for deciding on a particular munition are: Munition availability, unit activity, and range to target (closer the better).

The AGM will be different at different echelons. The AGM should also be phase-based (usually pre-planned) as well as being able to quickly change on the fly based on current battlefield conditions. JDAFS effectively captures these relationship through its robust optimization of fires.

### 6.2 Dynamic Sensor Allocation

The sensor formulation takes into account the capabilities of each sensor asset and enables them to be automatically compared in order to maximize the total sensor coverage given the simulations current state and near future sensor requirements. The following optimization problem is solved at each optimization event:

- Let $A$ be the set of all mission areas with at least one mission active within the global time horizon.
- Let $L$ be the set of all active LRS's.
- Let $G$ be the set of all GCS's.
- Let $G_L$ the set of all GCS's assigned to LRS L.
- Let $C_g$ the number of UAV's GCS $g$ is capable of controlling.
- Let $I_l$ be the subset of all UAV's at LRS $l \in L$ determined as follows: For each LRS $l \in L$, get $n$ UAV's where $n$ is the min(# of ready UAV's, LRS launch limit - # UAV's airborne, the sum over $G_L$ of $C_g$ - # UAV's assigned to the GCS).
- Let $J_l$ be the sub-set of all sensor packages currently located at LRS $l \in L$.
- Let $Y_{ga}$ = 1 if mission area $a$ is assigned to GCS $g$, 0 otherwise (by a heuristic discussed below).
- Let $c_{ja}$ = the reward for a UAV with sensor package $j$ being assigned to mission area $a$ from the soonest possible arrival time of the UAV at the area to the end of the time horizon, $t + \Delta t^i$, for UAV $i$.
- $X_{ja}$ = 1 if a UAV with sensor package $j$ is assigned to mission area $a$, 0 otherwise.

Then the formulation is:

$$\max \sum_{ja} c_{ja} X_{ja} \tag{1}$$

s.t.

$$\sum_{j} X_{ja} \le 1 \; \forall a \in A \tag{2}$$

$$\sum_{a} X_{ja} \le 1 \; \forall i \in I \tag{3}$$

$$\sum_{ja} Y_{ga} X_{ja} \le C_g \; \forall g \in G \tag{4}$$

$$\sum_{j \in J_l a} X_{ja} \le |I_l| \; \forall l \in L \tag{5}.$$

These equations are summarized as follows:

(1) Maximize the value of mission areas covered.
(2) Assign only 1 UAV per mission area.
(3) Assign only 1 mission area per UAV.
(4) Do not exceed the GCS control limit.
(5) The number of sensors assigned cannot exceed the number of UAVs available to carry them.

The heuristic for determining $Y_{ga}$ (assignment of mission areas' to GCS) is as follows:

- For each LRS and mission area $a$
  - Let $N_a$ be the number of GCS's $g$ that are in range of a UAV assigned to mission area $a$.
  - For each mission area $a$, sorted by $N_a$
    - For each GCS $g$, sorted by $\sum_{a} Y_{ga}$ (so far), if $a$ is in range of $g$ set $Y_{ga} = 1$.

The determination of $c_{ia}$ is as follows:

- For each UAV $i$
  - For each mission area $a$
    - Let $t_0$ = the first time after the soonest arrival time that UAV $i$ can gain value by being assigned to mission area $a$.
    - Let $t_1$ = the min(The latest UAV $i$ can remain at mission area $a$, the end of the time horizon, $t + \Delta t^i$, for UAV $i$).
    - Let $K_a$ = the set of all missions located at mission area $a$.
    - Let $V_{i,k,t0,t1}$ = the value that UAV $i$ gains from mission $k$ by being at mission area $a$.
    - Then, $c_{ia} = \sum_{k \in K_a} V_{i,k,t0,t1}$.

The optimization models described here are the ones that have been implemented in the current version of JDAFS. It should be noted that JDAFS has the capability of using different formulations, albeit with a modest amount of programming effort. For example, it is very straightforward to keep the same optimization structure while changing the exact formula used to compute the objective function coefficients. With slightly more effort, an entirely different optimization model can be utilized. The key feature is that very little of the other parts of JDAFS need be affected by these changes.

## 7 DISCUSSION

A number of characteristics of the JDAFS model make it a particularly good candidate for use in setting joint starting conditions for a high resolution simulation model. Most significant is the rapidity with which JDAFS scenarios can be created and the fast execution times relative to high resolution models. This allows for considerable exploration and analysis of the starting conditions, especially in determining sensitivity to certain factors.

The ability of JDAFS to formulate and solve optimizations problems on-the-fly ids particularly useful for determining "best" (or "worst") case scenarios, because the allocations are likely to at least be very good ones due to the optimizations performed during the runs. The development of JDAFS for establishing joint starting conditions is an ongoing effort.

## 8 CONCLUSIONS

The Joint Dynamic Allocation of Fires and Sensors (JDAFS) simulation provides a traceable approach that sets the unit and geographical starting conditions for high resolution simulations by using validated algorithms and data. JDAFS effectively uses an event graph approach and a innovative optimization in the loop schema that results in fast simulation runs that credibly represents all types of warfare from legacy AirLand Battle doctrine to exploratory future network-enabled warfare. Through optimization in the simulation loop, JDAF provides a traceable and flexible methodology of network-enabled allocation of fires and sensors to justify initial Joint starting conditions for high resolution simulations.

## ACKNOWLEDGMENTS

The work of the second author was supported by U.S. Army TRADOC Analysis Center, TRAC-Monterey. 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.

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.

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 D. Phillips. 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> *Accesses June 2006.*

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

## 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 Lieutenant Colonel in the United States Army, he received his Ph.D. in Operations Research from Boston University. His e-mail address is <dkahner@nps.edu>.

**JOHN RUCK** is a senior software engineer for Rolands and Associates. A retired Naval officer, he received his BSE from Tulane University, and MS degrees from the University of Central Michigan and the Naval Postgraduate School. His e-mail address is <jlruck@nps.edu>