Rapid Simulation-based Evaluation of Operational Plans

ABSTRACT

In military operations, traditional plan evaluation tools have always been used during the mission planning phase. However, in today's fast-paced and dynamic battlefields, the generated plans frequently do not survive their first contact with adversarial forces, forcing commanders to react to the situation and re-plan within tight time constraints. Fortunately, technological advancements in networked command and control systems now enable battlefield commanders to access a wealth of timely and detailed information and to use the data to simulate and extrapolate the possible results of each suggested plan. This proactive form of plan evaluation can complement the commander's decision-making processes during the mission execution phase. In the process of working on the plan evaluation simulator, we have become aware of various user issues and concerns, and this article will detail our experiences with them. Examples include user concerns about the underlying simulation model fidelity and the fundamental technical differences when porting a validated time-based simulation system into the realm of discrete event simulation. Other areas include methods to generate simulation scenarios from user plans, visual grouping and presentation of data automatically, which allow users to navigate through simulation time and an analysis engine quickly to make sense of batch mode results.

Choo Kang Looi
Jerry S/O Tamilchelvamani
Daniel Lee Jek Han
Daryl Lee Chin Siong

Rapid Simulation-based Evaluation of Operational Plans

INTRODUCTION

At the most basic level, military operations can be divided into two phases – planning and execution. During the planning phase, the commander and staff develop a set of possible future scenarios with the intelligence officers, making use of their experience and insights to develop the best course of action to be issued as a plan for execution in the next phase. Once the plan has been initiated, the commander and staff must recognise when the actual situation deviates from the plan and re-plan accordingly. They do not have the luxury of time and may have to react and develop the new course of action in a matter of minutes. Therefore, they rely on their experience in analysing the often incomplete data to develop a new course of action quickly.

For a plan evaluation tool to retain relevance beyond the planning phase, it needs to be able to react quickly to dynamic inputs and update its recommendations accordingly. The system needs to assimilate the input of incoming plans, reports and tracks into its symbiotic simulator to generate predictions. The system can then perform 'what-if' analysis by processing the results from multiple runs of the updated scenario to produce a course of action (COA) analysis report that helps to validate good solutions and warn of potential pitfalls. A successful and robust implementation can speed up the execution phase re-planning process considerably, resulting in better and more timely plans that significantly enhance operation capabilities. A major prerequisite is a scalable and highly accelerated simulation engine. Figure 1 shows a possible Operations (Ops) Case.

Military simulators built for training and war-gaming are primarily time-based. A simulation model updates the states of its various simulation attributes once every simulation tick. As this is computationally intensive, time-based simulators are constrained in their ability to accelerate their execution, peaking at a soft limit of 20 to 50 times clock speed, depending on the scenario and resources. Beyond this point, solutions typically involve simplifying the models, adopting a coarser tick size or scaling down the scenario in exchange for incremental increases in simulation speed. Even distributing the simulation load across different processors has limitations due to the interconnected nature of war-gaming scenarios, in particular that of the interaction between sensors and targets.

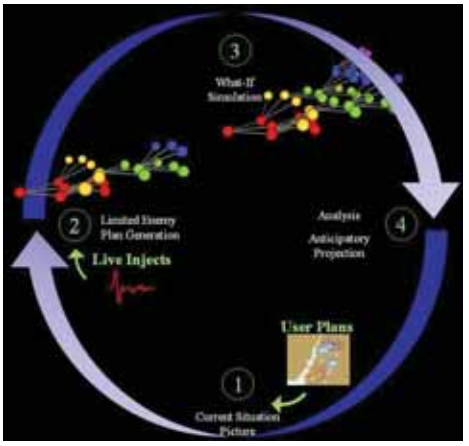

Figure 1. Ops concept – Symbiotic Simulation

Discrete event simulation (DES) works on a different paradigm, and updates at every event instead of every simulation tick. This typically works out to far fewer updates by many orders of magnitude, and the number can be further reduced by conscientious model design. For example, a movement model for an aircraft that updates every second might only need to update upon reaching its first way-point in a discrete event simulator, 30 minutes after take-off. This inherent advantage lends credence to the belief that DES is preferable for any time-critical application such as real-time plan evaluation.

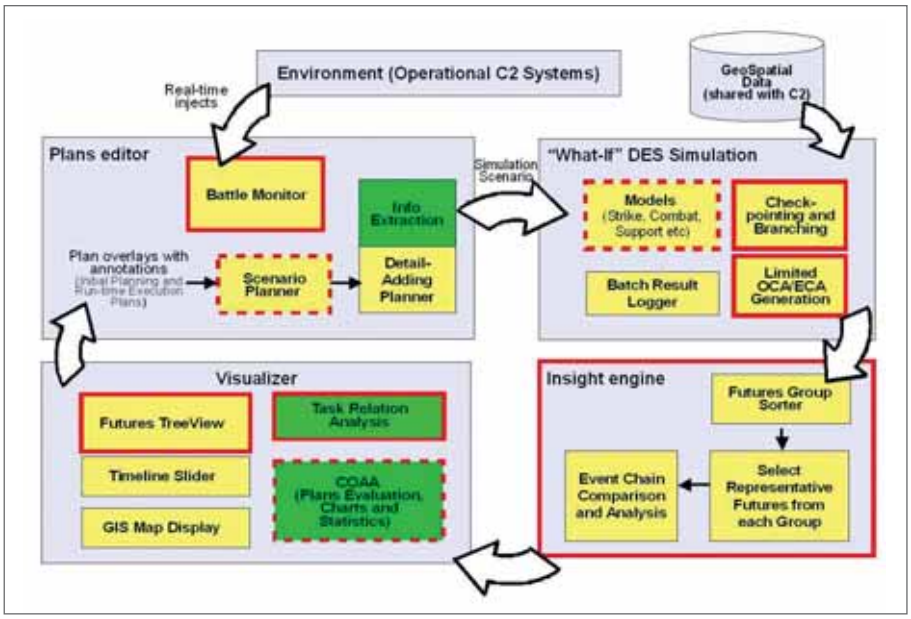

Figure 2. Proposed system diagram

Prototype Plans Evaluation Simulator

We have been working on a prototype rapid plan evaluation system (See Figure 2). The idea was to automate the operations plans to simulation pipeline, thus enabling a quick generation of results for analysis. The process starts by extracting structured mission data from annotated graphical user plans. This data is supplemented automatically with additional details such as terrain type, then packaged into a simulation scenario and sent for execution. The discrete event simulator supports two modes of operation – fast-forward and batch analysis.

In the fast-forward mode, the simulation is run to completion once quickly and then made available for playback via a time slider control and a map situation display. Just like a media player, users can vary the viewing speed and skip forward or backward to the sections which interest them.

In the batch analysis mode, the simulation is executed for a specified number of repetitions, varying one or more critical factors of interest or uncertainty. Significant events and statistics are recorded by the simulation for processing by the COA analysis engine. In its current incarnation, the analysis engine produces charts and plots displaying the relative performances of different user plans based on various simple selectable metrics. It attempts to do a correlation analysis between the effects of various parameters in the simulation. This is a work in progress and the goal is to create an engine that can help to suggest better plans, identify the factors and reasons for their relative success, as well as allow for evaluation using different user-selected metrics.

The prototype system covers a wide spectrum of research areas. This article will primarily focus on the development of the discrete event simulator, its interfaces and its two modes of operation in the context of the overall plan evaluation objective. We will share insights into the issues we encountered while working on this system as well as our experiences in managing them.

SIMULATION SCENARIO GENERATION

Plan Extraction from User Graphical Input

To input plan data, we made use of an experimental planning system that aimed to facilitate plan creation via overlay drawings. The overlays consisted of graphical objects (which could be annotated with important data such as time of execution), links to objectives and even operational intent. The system allowed for the export of these graphical objects into structured ontological formats, which we then mapped into simulation scenario files. We also made use of natural language processing technologies to extract the operational intent for task relation analysis. The general idea was to automate the creation of simulation scenarios via an interface that the users are comfortable with. This is done by emulating command and control (C2) interfaces.

User Plans to Simulation Scenario – Filling the Gaps

During the process of scenario generation, we found that the user plan inputs frequently lacked some data required for simulation execution. Thus, when the given plans stated that an armoured infantry unit designated as 87 X was in the scenario, we needed to extract details from the Order of Battle about 87 X.

There is also rarely any explicit information of the units' starting location. The simulator is thus obliged to make certain assumptions, such as having units start at the first way-point of their first mission's route, a pre-defined home base or the centre of the objective area they are tasked to defend.

Frequently, the simulator has to make assumptions regarding the execution of high-level plans, preferably by following supplied doctrines and standard operating procedures where available. Examples of execution details that need to be fleshed out include unit advancement speeds, rules of engagement under different mission types and artillery volley quantities.

Automatic Plan Generation

During the development of the prototype, we did a survey of literature on automatic COA generation. This study was done to assess the viability of providing an alternative COA generator to produce alternative own force plans to be executed by the simulator, and analysed in comparison with the user-generated plans. We also considered the possibility of generating opposing force plans automatically. However, due to resource constraints and the difficulties involved in implementing automatic plan generation, the conclusion of the study was not to implement it for the initial system prototype.

Going forward, we believe that even a limited form of plan generation will be useful to the system. The system needs to have some ability to predict both own and opposing force plans in the face of changing operational conditions and even simplistic plans are better than none. The most suitable form of artificial intelligence, customisable scripting, or experiential referencing system for this requirement has yet to be determined.

DISCRETE EVENT MODEL

When we first started on discrete event modelling, we found that it was very difficult to obtain reference sources for discrete event military simulation model algorithms due to the dominance of time-based simulators. As there were user concerns on the fidelity of our DES models, we chose to port from an established time-based war-gaming system. It is a 'chicken and egg' issue. Benefiting from years of user feedback, established systems and models have been able to match the user's perception of reality and be specific

to the local context. Without this user confidence, it is difficult to persuade them to use, evaluate and give feedback on the simulation models in the first place. Thus, validated sets of models such as those we ported from are nearly irreplaceable resources.

Adapting algorithm sources from a time-based to an event-based paradigm was a task that varied greatly in difficulty, depending on the subject being modelled. Static (i.e. non time-varying) algorithms can of course be reused without change. Algorithms that can be reduced to relatively simple equations, such as those for linear movement and radial sensing models, can be ported after some mathematical problem-solving. Complex models, such as those for an umanned aerial vehicle's articulated sensors tracing a scan pattern across uneven terrain littered with visual obstructions, defy the possibility of implementing non brute-force solutions without a large loss in fidelity. In all cases, it is important to document clearly what the simulator is and the aspects it is not able to consider in its model implementation. Otherwise, discerning observers will simply assume the worst.

To ensure that the algorithms we ported over were working correctly, we validated the simulation results of our own discrete event simulator with that of the time-based war-gaming system. In this example, we set up a scenario in which a battery of artillery guns was firing at a battalion of ground infantry in both systems.

Parameters such as gun and round types, number of rounds fired, distance from target and troop sizes were kept constant for both systems. The results yielded from the two systems were very similar, validating our discrete event model porting effort.

The following sections describe how we tackled some of the more interesting discrete event modelling problems we faced during the model development.

Attrition Modelling

Several distinct types of direct and indirect fire systems were modelled. Direct fire systems are further differentiated by whether they belong to the machine gun family. In all cases, different heuristics are used to arrive at 'hit' and 'kill' chances for the various weapon-ammunition combinations.

Weapon fire which occurs continuously over time at a high frequency cannot afford to have every projectile individually modelled. Adopting the discrete attrition model for such a fire system would overwhelm the DES engine with events. Instead, we used a mathematical model to describe the attrition as a function of time. We adopted the Lanchester model (Lanchester, 1956) for this purpose, using it to describe continuous attrition for direct fire machine gun systems.

For weapons which fire in bursts of lower frequency, we adopted a volley of shots concept where each volley is an event. A volley of shots is fired at discrete points in time at a frequency determined by the operational firing rate of the system. The indirect fire systems and the direct fire non-machine gun systems adopt this method of discrete attrition modelling.

Lanchester's Attrition Model

F.W. Lanchester was a British engineer who attempted to translate a wartime battle into a series of mathematical problems. Lanchester systems can be used to model specific combat situations. More precisely, they can be used to predict the outcome and the number of soldiers surviving a given battle. With the proper number of variables, these equations can be used to analyse extremely complex battles (Brown, 2001).

According to Lanchester's Square Law, in a mechanised warfare with modern weapons, each soldier can attack multiple targets but is also subject to incoming fire from many directions. The outcome is dependent on the weapon efficiency of the individual combatants and the combat strength of the army. An illustration of the law is depicted in Figure 3 and represented mathematically by differential equations (1) and (2):

$$\frac{dx}{dt} = -ay \quad \text{where } x(0) = x_0 \tag{1}$$

$$\frac{dy}{dt} = -bx \quad \text{where } y(0) = y_0 \tag{2}$$

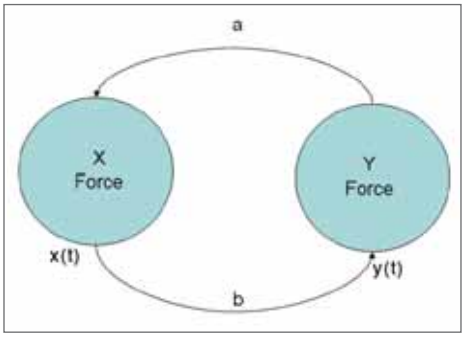

Figure 3. Lanchester Attrition Model

The *x* and *y* represent the number of troops in each army respectively. The *a* and *b* in each equation represent the killing efficiency of the respective armies. An army with superior firepower or better trained troops would have a higher coefficient of destruction. The larger number of troops in the opposing army leads to a higher rate of death for the army.

The solution to the differential equations gets complicated when more than two atoms are involved in conflict resolution. The Euler's method, an iterative solution, was used to solve the problem:

$$T_{n+1} = f(T_n, h) \tag{3}$$

We were able to use it to determine when units were reduced to pre-agreed fractions of their initial sizes and schedule the 'neutralised' events at these times. The complexity order using Euler was O(*n*). As this schedule remained valid only while the initial size and Lanchester Attrition-Rate coefficient stayed constant, it had to be re-assessed each time the hypothesis changed (e.g. when additional units enter the fray before combat ended). Therefore, in truth, the complexity was in the order of O(*cn*). The solution was slow and a direct numerical solution was required.

The eigenvalue method of solving systems of differential systems (The Open University) was used as the direct numerical solution. Equations (1) and (2) can be represented in matrix notation in the form of $\dot{x} = Ax$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & -a \\ -b & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

The eigenvalue and eigenvector pairs are obtained using JLAPACK, a Java numerical library originally written in FORTRAN (BLAS and LAPACK). Using the eigenvalues and the eigenvectors, we arrive at a general solution in the form of (5):

$$\begin{bmatrix} x \\ y \end{bmatrix} = C_1 \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} e^{\lambda_1 t} + C_2 \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} e^{\lambda_2 t} \tag{5}$$

The arbitrary constants, C*1* and C*2*, are solved using the initial value theorem when *t=0, x(0) = x0* and *y(0) = y0*. We are thus able to arrive at the force size decomposition of atoms involved in conflict resolution in any array of contact patterns. The complexity order of the solution is *O(1)*. Assuming re-assessments of *c* times where *c<<n*, the order is still in the range of *O(c)*.

The next problem we encountered was calculating the times at which atoms were reduced to intermediate percentages of their initial force size which we used as breakpoints for changes in readiness status. We used an approximation algorithm – Newton's method – to obtain the answers. Newton's root-finding algorithm can be mathematically represented as (6):

$$t_{n+1} = t_n - \frac{f(t)}{f'(t)} \text{ where } f(t) = K_1 e^{\lambda_1 t} + K_2 e^{\lambda_2 t} \text{ and } f'(t) = \lambda_1 K_1 e^{\lambda_1 t} + \lambda_2 K_2 e^{\lambda_2 t} \quad (6)$$

The terminating criteria of the algorithm is when the difference between *tn+1* and *tn* is within an acceptable Δt. Due to the decreasing exponential nature of *x(t),* the Newton's root-finding algorithm had an empirical iteration of less than 10.

The ammunition consumption was also obtained from the force size decomposition curve, *x(t)*. The ammunition used is basically the summation of *x(t)* from 0 to *t* multiplied by the rate of fire.

$$ammo(t) = rof \times \int_0^t x(t) \cdot dt \quad (7)$$

The duration of the conflict is passed in to equation (7) to obtain the ammunition consumption. The Newton's root-finding algorithm (6) is applied in conjunction with (7) to determine the time at which the atom will run out of ammunition.

Interaction of Continuous and Discrete Attrition Models

The discrete attrition models modify the force size at discrete points in time whereas the continuous attrition models represent the force size as a decreasing exponential curve. Both the models modify the same attribute.

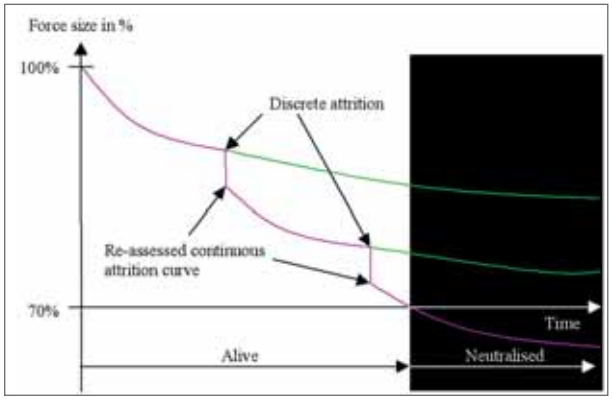

Figure 4. Illustration of discrete and continuous attrition

Through effective use of property change listeners, we managed the interaction between the discrete and continuous models. If an atom is involved in continuous attrition and is undertaking discrete attrition, we obtain the force size of the atom at the point in question from the continuous attrition curve and decrease it by the damage done by the discrete attrition model. The continuous attrition model is informed of the change through the property change listeners and a new curve is calculated using the updated initial force sizes (see Figure 4).

Line-of-Sight Checking

Unfortunately, not every modelling problem has an elegant solution. When faced with such cases, it is perfectly fine to treat the problem in a fashion similar to how one develops models for time-based simulators i.e. by scheduling events for it at regular intervals. Naturally, since this brute force method is processor-intensive and thus entails costs to execution speed, some optimisation should be applied to it, similar to any code run frequently. An example of such a problem is that of determining the times during which two units have line of sight to each other across uneven terrain.

To determine the times during which the sensor-target pairs are within detection range, we use geometry and relative velocity concepts and take into account their locations, velocities and sensor shapes. Once the simulation executes to these times of potential detection, a line-of-sight check event is scheduled. At this event, the sensor model projects forward in a 'time-stepped' manner until the line of sight is achieved. As this is not a true time-stepped simulation, it is possible to vary the step size dynamically based on sensor-target speeds and the terrain height resolution i.e. there is no point in doing a new line-of-sight check when neither sensor nor target has moved off from their previous terrain height grid locations.

As this area was recognised as a potential resource hog, various other optimisations were applied. These include keeping multiple levels of Digital Terrain Elevation Data height information in memory caches, hashing and ordering the database queries to favour those tending to lead to exit conditions first. In essence, we attempt to apply the most flexible and practical solution. We are avoiding terrain pre-processing for now, as it adds a barrier to the sharing of terrain datasets.

PLAN EVALUATION – FAST FORWARD MODE

The fast forward mode runs the simulation once and then plays it like a recorded video through the map display. Essentially, it allows a user to view the simulation in an intuitive time-based manner while at the same time skip to the sections which most interest him.

Timeline Slider

A timeline slider (see Figure 5) is used to control the current simulation time. It allows one to skip ahead or traverse back in simulation time simply by dragging a pointer. The slider is synchronised with the map display as shown in Figure 6, which refreshes to show the updated locations of all simulated units based on the indicated current simulation time on the slider. The map display also includes any animations for current events such as combat and artillery bombardment. Other controls of the time slider include 'play', 'pause', 'speed up', 'slow down', 'zoom', 'jump to next event' and 'jump to previous event'.

All critical events are displayed directly on the timeline slider as icons, providing an instant compressed overview of the simulation results. These also act as bookmarks for easy navigation through the timeline using the 'jump to next' or 'previous event' functions. Intuitive icons are used to represent events such as start mission, end mission (i.e. successful), fail mission, readiness condition status change and engagement.

Here is a sample of how the timeline slider might be used. Picture a simple scenario in which an infantry company moves to an objective area to capture it. First, a user would start by using the play mode of the timeline slider. As the movement to the objective area is not of interest to the user, he might increase the play speed or jump to the next event, which would be the engagement of enemy units at the objective area. During engagement, several events occur close together in time, cluttering the display with events. The user can use the slider's zoom feature to stretch them out for a clearer view.

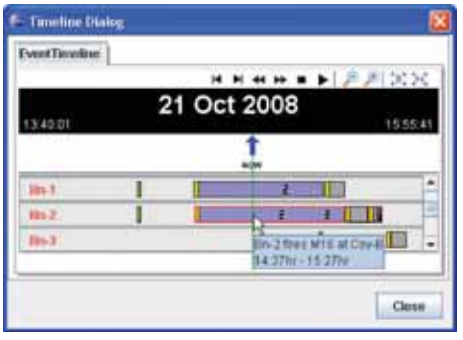

Figure 5. Timeline slider

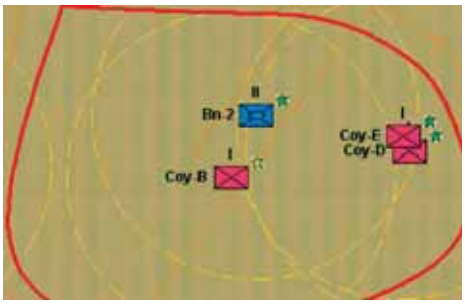

Figure 6. Map display according to timeline slider

All these controls allow the user to navigate through the simulation scenario quickly and easily.

Interpolation: From Discrete Events to Continuous Play

It is impractical to record the location of every unit at every possible point in the simulated time. Therefore, we only record key events such as unit velocity change. These are recorded by an event logger in the back end. These events are then sent to the front end in XML format. Recorded inside each event is the time of the event, the unit initiating the event and the location of the event or unit. The events are sorted and grouped according to their initiating units.

When the timeline slider moves or is dragged to a time *t*, the next nearest velocity change event, *E,* that occurs to each unit prior to time *t* is selected. From the event *E*, we have both the location and velocity of that unit. Using this method, we can easily interpolate the location of every unit at time *t*. Therefore, the locations of all units are known at time *t* even though *t* is most likely not the time of occurrence for any of their velocity changes.

Previously, the interpolation logic was placed in the back end. When the timeline slider moved, the new current time was sent to the back end via XML where the interpolation was done. After doing all the calculations, the new locations of the units were then sent back via XML to the front end to refresh the location displays. Unfortunately, the result was a jerky playback as the back-and-forth communication incurred a noticeable lag time. When we shifted the interpolation logic to the front end, the performance improved tremendously.

Event Representation in Timeline Slider – Making it Uncluttered and Intuitive

Initially, the slider displayed all events as instantaneous and independent. To declutter the slider display, we combined related events into combined representations. For example, a Start Engagement event and an End Engagement event are combined into an Engagement event – a single bar icon on the Timeline Slider.

To make the timeline slider more organised and intuitive, every event is linked to at least one force and unit and the display is sorted as such. This makes it easier to isolate a series of events related to the unit or units of interest.

Map – Slider Synchronisation

We found early on that there was a need to synchronise the timeline slider and the map display. Without synchronisation, the slider would proceed faster than the map display could update during the play mode. This lag would become increasingly noticeable as time went by. The solution was to have the slider pause at a regular time interval and wait for its cue from the map display that it was done rendering before proceeding on with the play.

PLAN EVALUATION – BATCH ANALYSIS MODE

The inherent speed advantage of the DES engine gives rise to the possibility of a batch run that can be completed within a matter

of minutes rather than as a job left running overnight. Hundreds or even thousands of repetitions are possible in this fashion. With randomness injected into each simulation run, a stochastic process can be built from which we can analyse and draw conclusions. This is the function of the Batch Analysis Mode – run the simulation multiple times, collect the resultant data and feed it into the COA Analysis Module for processing. The data undergoes some statistical analysis before its output in the form of plots, charts and graphs that will aid the commander in plan evaluation.

Batch Run

Varying Each Run

Traditionally, batch runs are used for Operations Analysis studies, where thousands of runs are executed over days or even weeks while operators carefully plan out the parameter variations between those runs. After some initial tests, we concluded that the DES engine was fast enough to be used in batch mode for plan evaluation. However, since we no longer have days for operators to tweak simulation parameters deliberately, the runs need to be varied in a useful way by the system itself.

One key criterion for the success of the batch analysis mode is that there must be sufficient realistic randomness in each run such that the batch run generates results that approximate a live exercise. In other words, the batch run should ideally cover all conceivable ways a scenario would play out. Practically, we do this by varying certain critical variables for each run. Some possible variables we have identified include enemy unit starting locations, travel routes, initial sizes, weapon effectiveness and hit as well as damage parameters.

Implementation

We allow the user to specify the number of simulation runs in each batch. In practice, for a scenario involving a ten-battalion-strong friendly force versus a ten-battalion enemy force of tanks, infantry and artillery, 100 runs would take about half a minute while 1,000 runs would take about five minutes. For every new simulation run, we reset all runtime variables to their initial states. The randomness we introduced in each simulation run includes: starting positions of units, hit and damage calculations for direct fire (non-machine gun) and indirect fire weapon systems, as well as the weapon effectiveness coefficient for direct fire weapons. We did not cover all critical variables due to time constraints.

Data collected from the runs can be broadly categorised into three classes: i) event data, ii) task data, and iii) scenario summary data. Event data is data from every event that occurs in the simulation run. Events include detection, fire, stop fire, start of run, end of run, start mission and end mission. Task data is data from each task/mission for each unit in the simulation run. Tasks could include advance, attack, defend or patrol. The start and end states for each task are captured. Scenario summary data is data collected at the unit level for all the runs. Start and end states for each run are captured. It includes specific data like atom sizes for infantry, tank and artillery, as well as the quantities of the various weapon and ammunition types. Examples of the data collected are shown in Table 1.

The data is grouped in such a manner for the benefit of the COA Analysis Module. All data is stored in memory for every run until the end of the entire batch run, after which the data output is in Comma Separated Value (.CSV) files and text (.txt) formats. The COA Analysis Module reads in these file formats.

Event Data		Task Data	
MissionID:	Advance_123	Mission Name:	111 Advance
Event ID:	109	Mission Status:	Success
Event Name:	Start Mission	Objective:	(100,200,300)
Time:	0.0	Start Time:	0.0
		End Time:	3600.0
Unit Name:	111	Unit Name:	111
Unit Location:	(0,0,0)		
Unit Size:	63	Start Size:	63
Unit Infantry Size:	63	Start Infantry Size:	63
Unit Tank Size:	0	Start 556 Ammo:	3000
Unit Redcon:	REDCON1	End Size:	63
Unit 556 Ammo:	3000	End Infantry Size:	63
Unit 762 Ammo:	0	End 556 Ammo:	1400
Scenario Summary Data			
Atoms at Start		*Ammo at Start*	
Unit 111: Tank	= 9	Unit 111: 556mm	= 3000
Unit 111: Infantry	= 126	Unit 111: 105mm	= 90
Unit 112: Tank	= 0	Unit 112: 556mm	= 3000
Unit 112: Infantry	= 63	Unit 112: 105mm	= 0

Table 1. Event, task and scenario summary data

Optimisation - Terrain

Each time any model needs to know the terrain type of a specified location, whether it is for movement or combat, a query is made to a database to retrieve the terrain type. Database queries are expensive and can potentially affect the performance of the batch run. To optimise the batch run, the terrain types retrieved in each simulation run are cached. As the enemy and friendly routes in each simulation are fairly similar across runs, caching terrain types for each run would significantly cut down database calls in subsequent runs and improve the overall performance.

This process is further optimised by hashing the cached terrain types based on a location grid. Movement in the DES engine is via way-points. An exact hit on the same coordinates is unlikely as we sometimes vary the unit's movements. To handle this, coordinates in the same grid square are assumed to be of the same terrain type. This grid size is configurable. The cache is maintained using a Least Recently Used algorithm.

COA Analysis Module

The output data from the batch run is fed into the COA Analysis Module which in turn presents the data in a form that facilitates the COA analysis and comparison by the user. The COA Analysis Module is made up of three components – i) COA mission outcome, ii) COA comparison and iii) COA detailed analysis. For the purposes of this module, a COA comprises a friendly course of action and an enemy course of action.

COA Mission Outcome

The COA mission outcome is a simple first-level view of the results showing the average percentage mission success rate of both the Blue (friendly) and Red (enemy) forces, the average percentage of Blue force killed and the average percentage of Red force remaining.

Currently, the overall success rates are an aggregation of the individual units' mission success rates. After some iteration, we realised that this did not always reflect the overall success of the campaign. For example, in a scenario where out of four blue units attacking an objective, three are neutralised but the last succeeds, a human operator would deem the campaign a success. However, a simple aggregation would only report a 25% success.

COA Comparison

The COA comparison module compares different COAs so that the best COA may be selected. Unit-level attributes taken from the scenario summary data are compared between the two specified COAs. The COAs are ranked based on each attribute and a lower rank value indicates the better COA.

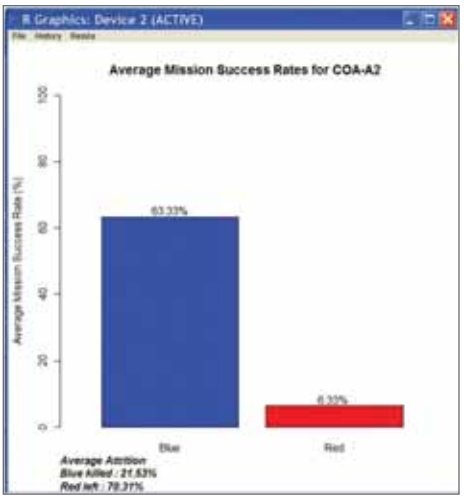

Figure 7. Mission success bar chart

This comparison can also be done visually via parallel plots. Each marking on the horizontal axis represents an attribute that is used in ranking. Each COA is represented by a solid line of unique colour so that the different COAs are easily distinguishable. The vertical axis is configured to facilitate analysis from Blue's perspective i.e. for each attribute, the best COA is the one that is placed highest on the axis as compared to other COAs. Thus, for every attribute where smaller values are preferred from Blue's perspective (e.g. Red mission success rate), the values of the attributes for the COAs are negated so that the most preferred value appears above those of the other COAs in the vertical axis. Two thin grey horizontal lines above and below the vertical axis are added to the plot to mark the higher and lower boundaries of the possible values.

COA Detailed Analysis

The analysis carried out by this component is to determine which attributes that describe engagements are most highly correlated with the mission outcome of the simulation. Examples of attributes that describe the engagements include: 'who wins the first engagement', 'who detects the opponents first' and 'how many times a unit wins attacks it has initiated'. These 'intermediate' attributes may help characterise a COA. 'Outcome' attributes describe the status of the Blue and Red forces at the end of the simulation. Examples of 'outcome' attributes are those that describe the number of Blue units killed, the mission success rate of Blue forces, and the numbers of the Red units remaining. Correlation analysis between 'outcome' and 'intermediate' attributes can help to answer questions such as 'Does early victory improve the mission success rate?' and 'Which unit influences overall mission success or failure?'.

Statistical tests of association are carried out on the pairs of intermediate outcome data attributes. The choice of test is based on the data type of the attributes. There are two data types – numerical and categorical. An example of a numerical attribute is 'mission success rate' which is a real number in the range [0,100]. An example of a categorical data attribute is 'who detects opponent first' – it has two possible nominal values, 'Blue' and 'Red'. In the case of categorical data attributes, some numerical statistics must be derived to describe this attribute. For

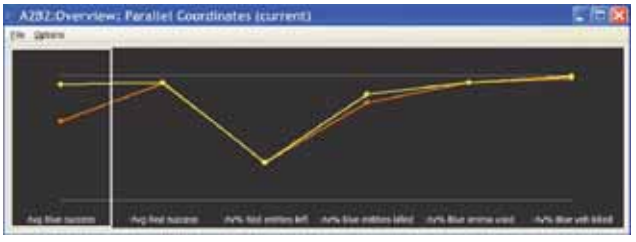

Figure 8. Parallel plot

example, we should compute the frequency with which Blue or Red is first to detect its opponent. The tests used are shown in Table 2.

Attribute 1	Attribute 2	Test	Aim of Test
Numerical	Numerical	Spearman	Tests for linear correlation between the variables
Numerical	Categorical	Kruskal-Wallis	Tests whether the medians of the numeric data among the categories are equal
Numerical	Categorical	Fligner	Tests whether the variances of the numeric data among the categories are equal
Categorical	Categorical	Fisher	Tests whether there is association between the categories of attribute 1 and attribute 2

Table 2. Statistical tests for COA detailed analysis

After the tests are carried out, the attribute-pairs are sorted in ascending order of the *p*-values, where *p*-value is the probability that random sampling would result in a correlation coefficient as far from zero as observed in the test, given that there is no correlation between the two tested attributes.

The top five attributes are selected (according to the five smallest *p*-values). The data for the attributes in these top five pairs will be displayed in scatter plots and/or bar charts. A text file showing the tests that were carried out and statistics related to the tests (e.g. the correlation coefficient, medians, etc.) will also be displayed.

THE WAY AHEAD

Local reception to the prototype plan evaluation system has been highly positive. However, there is still a lot of work to do before it can be deployed. At the time of writing, we are working to pool resources towards enhancing this system for deployment within a focused domain instead of the current more general approach.

Foremost among the enhancements planned for the next phase is the ability to update the simulation based on the latest intelligence reports as well as changes in own force plans. The simulator will need to take in these injects and re-execute the scenario on the fly. Only then can it remain relevant beyond the planning phase and into the execution phase.

There are also plans to enhance the batch run capability to support the simultaneous display of multiple possible futures instead of just generating statistics for analysis. To achieve this 'crystal ball' effect, two requirements have to be met. First, as alluded to earlier in this article, the individual simulations need to vary in a meaningful way. For instance, randomising the routes and behaviour of enemy units generates more learning value than merely varying the attrition coefficients of a weapon system. After all, enemy behaviour is tricky to predict whereas hardware data should be well known. We recognise that some aspects of the system require a limited form of COA generation, but a full implementation is beyond the scope of the project.

The second requirement is to have an analysis tool capable of processing the results of the batched simulation runs and extracting from them a manageable number of representative scenarios to display to the user for more detailed analysis. This dovetails well with plans to upgrade the COA Analysis Module. The ability to identify key decisions and events that leads to major changes in outcomes would help to differentiate and efficiently categorise the different simulation predictions.

While these are relatively frontier technologies, we are aware of similar efforts

in the international arena. We hope to learn from their experiences while focusing our resources and efforts towards local needs such as automated C2 plan importing and Sim-C2 integration.

CONCLUSION

Discrete event simulators have been successfully deployed across many domains, military or otherwise, via products like Simscript and Simula. However, while it remains less prevalent than its time-based counterpart, we believe it has great potential for expansion into areas favouring speed and automation over manual input, real-time plan evaluation being just one case.

Nevertheless, we need to remain aware of the strengths and weaknesses of such a system. It is great for visualising a plan in action and has the potential to help its operators identify key turning points. However, the results are only as good as the inputs and should be taken as suggestions of what to look out.

ACKNOWLEDGEMENTS

We would like to thank the then Directorate of Research & Development of the Defence Science and Technology Agency for funding this research, DSO National Laboratories for their work in the COA analysis, and the Naval Postgraduate School for their Simkit engine.

REFERENCES

Brown, Greg. 2001. Lanchester's Square Law Modelling the Battle of the Atlantic. College of the Redwoods. http://online.redwoods.cc.ca.us/instruct/darnold/deproj/Sp01/GregB/battle_s.pdf (accessed 4 June 2009)

Buss, Arnold H., and Paul J. Sanchez. 2005. Simple Movement and Detection in Discrete Event Simulation. Paper presented at the Winter Simulation Conference, 4–7 December, in Orlando, USA.

George Mason University. SYST-683: Models, Gaming and Simulation. http://classweb.gmu.edu/ralexan3/SYST683/ (accessed 29 December 2008)

James O. Henriksen. 1988. One System, Several Perspectives, Many Models. Paper presented at Winter Simulation Conference, 12–14 December, in San Diego, USA.

Lanchester, F.W. 1956. Mathematics in Warfare. The World of Mathematics, Vol. 4.ed J. Newman, 2138 – 2157. Simon and Schuster (New York).

Naval Postgraduate School. Simkit and Event Graph Models. http://diana.nps.edu/Simkit/ (accessed 29 December 2008)

Pratt, David R., Robert W. Franceschini, Robert B. Burch, and Robert S. Alexander. 2008. A Multi Threaded and Resolution Approach to Simulated Futures Evaluation.

Savelli, Rafael Moreira, Gustavo Henrique Soares de Oliveira Lyrio, and Roberto de Beauclair Seixas. The Manoeuvre and Attrition Warfare Simulation System. http://w3.impa.br/~rbs/pdf/SPOLM2004_mawss.pdf (accessed 29 December 2008)

The Open University. Systems of Differential Equations. http://openlearn.open.ac.uk/course/view.php?id=2747 (accessed 29 December 2008)

This article was first presented as paper 09S-SIW-028 at the 2009 Spring Simulation Interoperability Workshop held from 23–27 March 2009 at San Diego, CA, USA and has been adapted for publication in DSTA Horizons, with the permission of the Simulation Interoperability Standards Organization, Inc. (SISO).

BIOGRAPHY

Choo Kang Looi is a Senior Engineer (C4I Development). He has worked in Modelling & Simulation (M&S) across multiple projects spanning engine development, Command and Control (C2) integration and deployment, research and experimentation. Kang Looi has co-authored and presented two papers at the Simulation Interoperability Workshop in USA. He is presently working on M&S architecture and exploring the application of Discrete Event Simulation in process experimentation and training. He received his Bachelor degree in Computer Engineering (First Class Honours) from Brown University, USA in 1999.

Jerry S/O Tamilchelvamani is an Engineer (C4I Development). He is involved in C2-Sim Interoperability and Discrete Event C2 simulation-based projects. His research interest is in computer algorithms and he was the co-developer of the Discrete Event Simulation models. He presented this article at the annual Simulation Interoperability Workshop in USA. Jerry received his Bachelor degree in Computer Engineering (Honours) from Nanyang Technological University (NTU) in 2006.

Daniel Lee Jek Han is an Engineer (C4I Development). He has worked specifically on projects that deal with C2 integration, Man Machine Interaction and engine development. These projects have led to collaborations with DSO National Laboratories and Intel Solutions. He presented this article at the annual Simulation Interoperability Workshop in USA. Daniel received his Bachelor degree in Information Systems (Honours) from the National University of Singapore in 2007.

Daryl Lee Chin Siong is a Principal Engineer (C4I Development) and currently oversees the M&S programme. He has co-authored two papers at the Simulation Interoperability Workshop in USA. He also represents DSTA as an Editorial Board member of the POINTER journal published by the Singapore Armed Forces. Daryl obtained a Bachelor degree in Computer Engineering from NTU in 1995 and a Master of Science degree in Computer Science from the Naval Postgraduate School, USA in 2004.