TSEM, A Flexible Scenario Based Small Forces Model

Bruce D. Link, Henry D. Shapiro
Sandia Laboratories
Albuquerque, New Mexico

Abstract

An accurate computer model of a small force engagement is useful in evaluating the combat effectiveness of armed escorts for sensitive shipments, security guard forces and military patrols. The Transportation Safeguards Effectiveness Model (TSEM), primarily intended for, but not limited to, the study of ambushes of armed convoys, provides the user with considerably greater flexibility in directing the actions of the combatants than previous models. A user oriented script language is presented, which allows the performance of actions to be made contingent on the occurrence of significant events during the battle, providing the means for describing a wide variety of realistic scenarios. In addition, careful attention is being paid to the research results and experimental data supporting the human factors submodels. A new, more accurate casualty assessment model, the use of which can change battle times by as much as an order of magnitude, is discussed.

INTRODUCTION

The needs of the military in the Vietnam era and the needs of civilian authorities charged with transporting nuclear weapons and materials have given rise to a number of small force engagement models. Computer run, discrete combat simulation models became of interest to analysts because they realized that they lacked quantitative data on overall system vulnerability and relative effectiveness of various tactics on which to base decisions. While staged engagements using combat personnel provided valuable training and suggested alternative tactics for evaluation, computer modeling offered, and still offers, three advantages:

(1) The ability to model certain aspects of combat more accurately than they can be modeled by the equipment used in the environment of training exercises. Weapons characteristics and casualty assessment are most often pointed to as examples.

(2) The ability to run a sufficient number of replications to achieve statistical significance. The manpower requirements, as well as the changing behavior of combatants as they learn the characteristic behavior of the adversary, make performing an adequate number of staged engagements impractical.

(3) The possibility of rapidly evaluating the effect of changes in system configuration or tactics.

In the past, failure to provide the analyst with sufficient flexibility to express the variety of tactics he wished to compare and concern that human factors effects were too little understood and too complex to model have led to criticism of the modeling technique [11].

In developing the Transportation Safeguards Effectiveness Model (TSEM), a small forces engagement model primarily intended for, but not limited to, the study of ambushes of road convoys escorted by armed guards, the staff of Sandia Laboratories paid particular attention to the criticisms cited above. With the completion of the current phase of development, a working model has been produced which is capable of being used in system studies. A major innovation has been the incorpora-

CH1437-3/79/0331-0345$00.75
1979 Winter Simulation Conference

tion of a front end, user oriented script language as an integral part of TSEM, allowing straightforward specification of a wide variety of scenarios. The ability to rapidly develop scripts for realistic scenarios provides the user/analyst with a model that is flexible enough for meaningful comparative studies. The ability to precondition the activities of a player on the occurrence of specific events on the battlefield allows the user/analyst to describe in detail the tactics employed by each individual player or group of players. This clear separation of higher level combat functioning from more elemental behavior serves two related purposes. First, by developing a sufficient number of scenarios, the analyst can explore to what degree and in what manner the system under evaluation is vulnerable. Second, the scenarios, formally expressed in the easily understood script language, document exactly what alternatives were studied. The script facility gives a flexibility not found in previous models, where internal criteria were used to decide what actions to take when a significant event occurred. In these earlier models, the analyst could not try other alternatives (without recoding the model) and could not easily determine what the selection criteria were. Because of the central place the script language holds in TSEM and because this capability is not found in other combat models, the main body of the paper deals with a comprehensive example using a scenario described in the actual script language. This not only shows the degree of flexibility attained, but provides an easy introduction to the script language and its implementation.

While the scenario description facility provides flexibility, there is still the question of reasonable accounting for human factors effects. In many areas, such as suppression effects or the interaction between concealment and firing and movement strategies, the documentation available on prior work cites very few research results or experimental data to support the human factors submodels used. Concern over the importance of human factors has led the TSEM project staff to evaluate each submodel from first principles. Even in areas where data is available or easily obtained, order of magnitude improvements can sometimes be obtained with no loss of model efficiency. The last section of the paper discusses a new casualty assessment model. By utilizing the most current dispersion and lethality data [5,6,7] and new, more accurate information concerning postures assumed during combat [8], the battle times predicted by the model should more closely reflect actual battle times. The simulated battle times differ by as much as an order of magnitude when compared to previous casualty assessment models. Plans to gather behavioral data in poorly understood areas is also briefly discussed.

THE SCRIPT FACILITY

In order to develop an understanding of the capability and flexibility gained by introducing a scenario description facility into a small force combat model, it is helpful to examine an example in considerable detail. Figure 1a shows a plan view of the combat area with the four attackers in their initial ambush configuration. For simplicity of presentation a level terrain was chosen, though the model can approximate virtually any terrain in a straightforward manner. The four defenders and their associated vehicles, a cargo vehicle with only the driver, who is armed, and an escort vehicle, with three armed guards, are shown in the positions where the vehicles come to rest and their occupants dismount. The scenario that is to be enacted is of moderate complexity; the script definition language has been used to describe more sophisticated scenarios.

The strategy that the attackers employ is as follows: Attacker 1 (A1) is the primary saboteur, his goal being to penetrate the truck in the middle of the north side. In order for A1 to begin closing in on the vehicle, however, the driver (D1) must be incapacitated. Attackers 1, 2, and 4 attempt to kill D1 before he can reach the cover of heavy brush. If they succeed, A1 moves in immediately, with A2 and A4 providing covering fire. A2 and A4 also move to better vantage points as well; A2 to the boulder field and A4 to the right lobe of the upper central brush area. If they fail, and D1 reaches the brush, A2 changes his tactics and switches his attention to the defenders from the escort vehicle, moving to the forest to engage them. A1 and A4 continue to center their attention on D1 who, by virtue of reaching the brush, has obtained better cover. A1 moves around the house to get a better angle on D1. If A1 and A4 eventually incapacitate D1, A1 will still attempt to reach the vehicle, though now only A4 will provide covering fire. An additional part of the strategy is that if A1 becomes incapacitated, A4 will assume his duty, and try to reach the truck and penetrate it once the driver is incapacitated.

Because of his position at the far left of the combat area, A3 has a quite different role. He initiates the ambush by firing on the vehicles when they enter the ambush zone. If any of the defenders escorting the cargo make it to the brush near where their vehicle stops, A3 attempts to sneak up on the guards by a circuitous route that goes near the lower left corner of the map. By attacking the guards from the escort vehicle from the

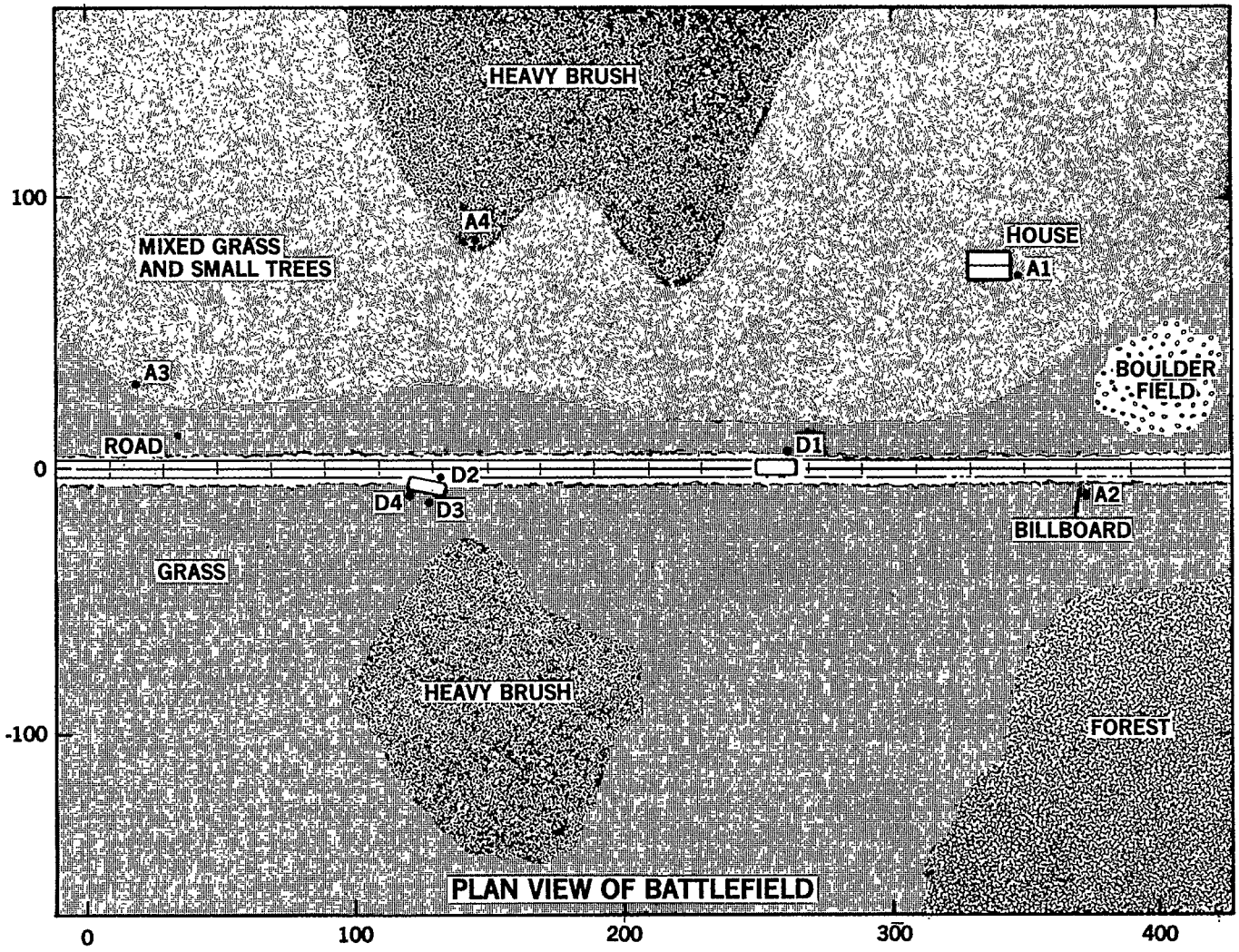
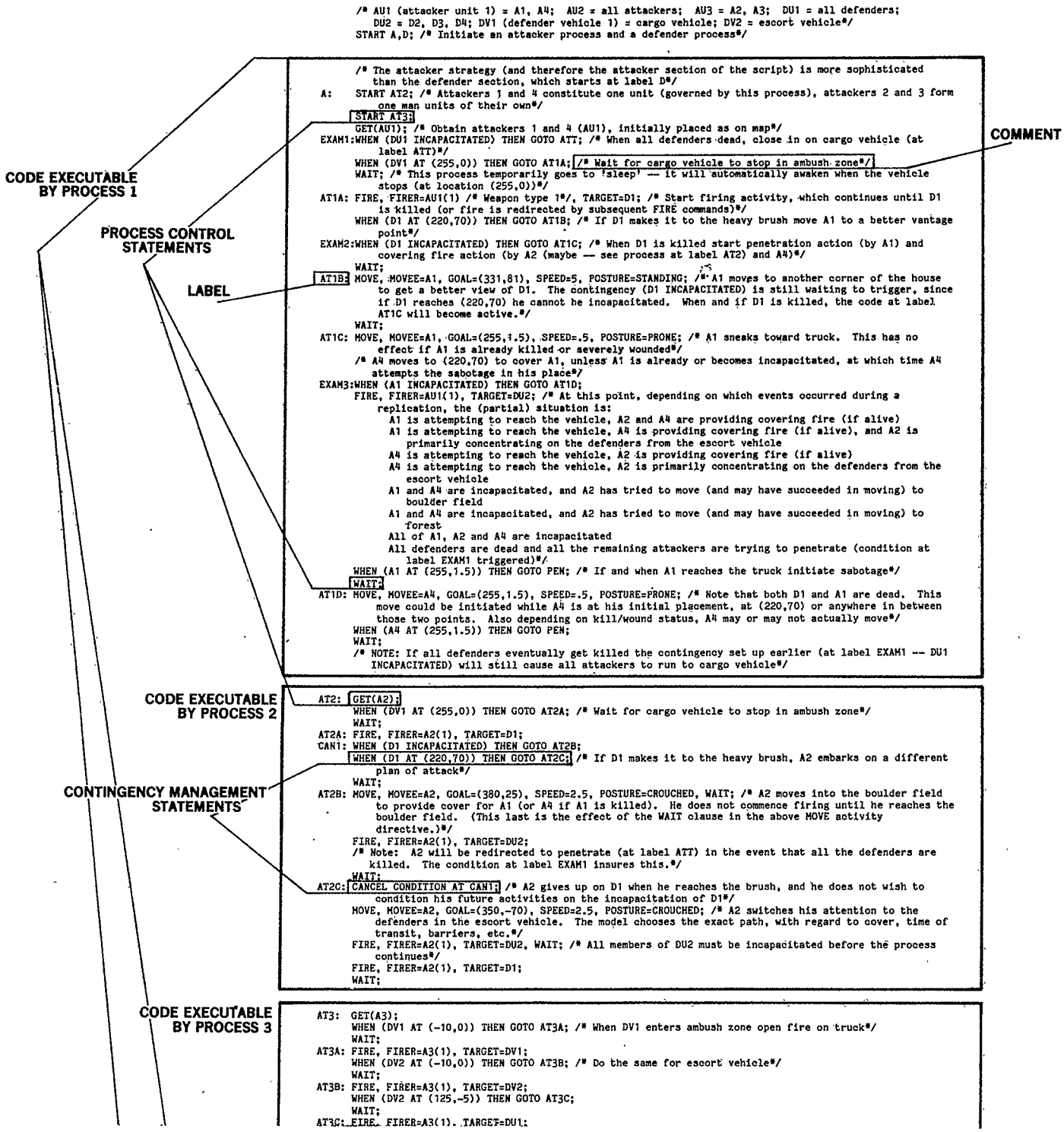


FIGURE 1a

rear he hopes to inflict casualties as well as tie down the escorts, possibly with the help of A2, so that A1 (or possibly A4) can attempt to penetrate the cargo vehicle in relative safety. When all of the defenders are dead, any remaining attackers will attempt penetration.

The strategy of the defenders is simpler. The driver of the cargo vehicle, who is clearly outnumbered, heads for the nearest high quality cover, where he engages in a firefight with the attackers he can see. The armed guards from the escort vehicle head for cover as a unit and move through the brush to the eastern tip of the lower brush patch, where they have good cover, can snipe at any attackers attempting to penetrate the cargo vehicle from the south, and battle with A2 and A3. If they succeed in neutralizing A2 and A3, they move back behind the escort vehicle, which affords them a good view of the cargo vehicle they wish to protect, and allows them to engage A1 and A4.

Due to wounds received by, as well as death of, certain combatants, in any given battle using the strategy just outlined, some of the actions described above will come to pass, and others will not. An important aspect of the means by which the scenario was specified, is the use of contingencies: "If they (A1, A2, and A4) succeed (in killing D1), A1 moves in immediately," and "If they (D2, D3, and D4) succeed in neutralizing A2 and A3, they move," are examples. Unlike earlier combat models, which provided the user only limited and inconvenient means of specifying changes to the strategy should the battle develop along lines that would warrant it, TSEM offers the capability to do this through its front end script definition language which contains statement types designed to make specification and management of contingencies a simple matter. Figure 1b shows the (heavily commented) script, written in the script definition language, which together with data statements (not shown) describing

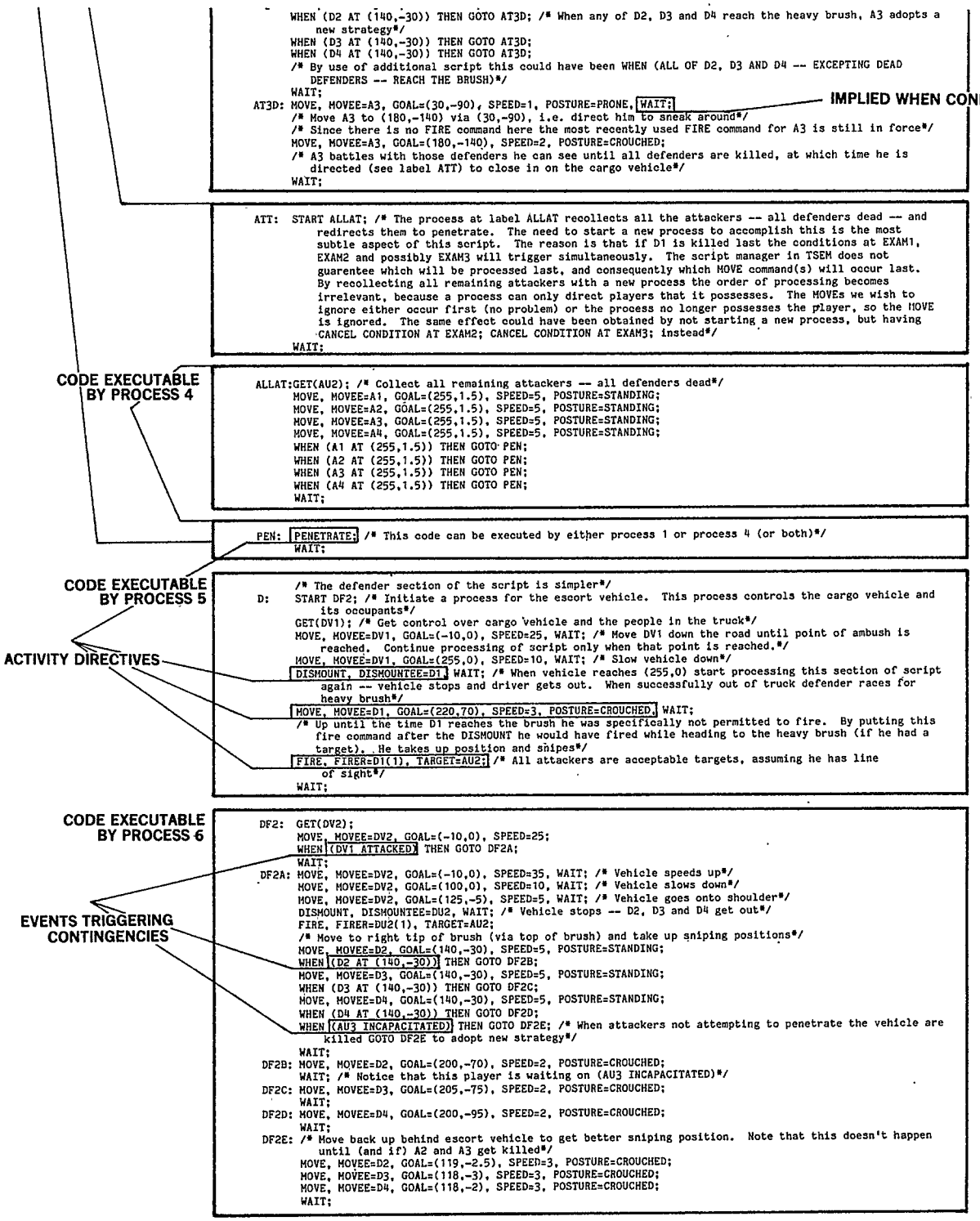



```
/* AU1 (attacker unit 1) = A1, A4;  AU2 = all attackers;  AU3 = A2, A3;  DU1 = all defenders;
   DU2 = D2, D3, D4; DV1 (defender vehicle 1) = cargo vehicle; DV2 = escort vehicle*/
START A,D; /* Initiate an attacker process and a defender process*/

   /* The attacker strategy (and therefore the attacker section of the script) is more sophisticated
      than the defender section, which starts at label D*/
A: START AT2; /* Attackers 1 and 4 constitute one unit (governed by this process), attackers 2 and 3 form
      one man units of their own*/
   START AT3;
   GET(AU1); /* Obtain attackers 1 and 4 (AU1), initially placed as on map*/
EXAM1:WHEN (DU1 INCAPACITATED) THEN GOTO ATT; /* When all defenders dead, close in on cargo vehicle (at
      label ATT)*/
   WHEN (DV1 AT (255,0)) THEN GOTO AT1A; /* Wait for cargo vehicle to stop in ambush zone*/
   WAIT; /* This process temporarily goes to 'sleep' -- it will automatically awaken when the vehicle
      stops (at location (255,0))*/
AT1A: FIRE, FIRER=AU1(1) /* Weapon type 1*/, TARGET=D1; /* Start firing activity, which continues until D1
      is killed (or fire is redirected by subsequent FIRE commands)*/
   WHEN (D1 AT (220,70)) THEN GOTO AT1B; /* If D1 makes it to the heavy brush move A1 to a better vantage
      point*/
EXAM2:WHEN (D1 INCAPACITATED) THEN GOTO AT1C; /* When D1 is killed start penetration action (by A1) and
      covering fire action (by A2 (maybe -- see process at label AT2) and A4)*/
   WAIT;
AT1B: MOVE, MOVEE=A1, GOAL=(331,81), SPEED=5, POSTURE=STANDING; /* A1 moves to another corner of the house
      to get a better view of D1.  The contingency (D1 INCAPACITATED) is still waiting to trigger, since
      if D1 reaches (220,70) he cannot be incapacitated.  When and if D1 is killed, the code at label
      AT1C will become active.*/
   WAIT;
AT1C: MOVE, MOVEE=A1, GOAL=(255,1.5), SPEED=.5, POSTURE=PRONE; /* A1 sneaks toward truck.  This has no
      effect if A1 is already killed or severely wounded*/
   /* A4 moves to (220,70) to cover A1, unless A1 is already or becomes incapacitated, at which time A4
      attempts the sabotage in his place*/
EXAM3:WHEN (A1 INCAPACITATED) THEN GOTO AT1D;
   FIRE, FIRER=AU1(1), TARGET=DU2; /* At this point, depending on which events occurred during a
      replication, the (partial) situation is:
        A1 is attempting to reach the vehicle, A2 and A4 are providing covering fire (if alive)
        A1 is attempting to reach the vehicle, A4 is providing covering fire (if alive), and A2 is
          primarily concentrating on the defenders from the escort vehicle
        A4 is attempting to reach the vehicle, A2 is providing covering fire (if alive)
        A4 is attempting to reach the vehicle, A2 is primarily concentrating on the defenders from the
          escort vehicle
        A1 and A4 are incapacitated, and A2 has tried to move (and may have succeeded in moving) to
          boulder field
        A1 and A4 are incapacitated, and A2 has tried to move (and may have succeeded in moving) to
          forest
        All of A1, A2 and A4 are incapacitated
        All defenders are dead and all the remaining attackers are trying to penetrate (condition at
          label EXAM1 triggered)*/
   WHEN (A1 AT (255,1.5)) THEN GOTO PEN; /* If and when A1 reaches the truck initiate sabotage*/
   WAIT;
AT1D: MOVE, MOVEE=A4, GOAL=(255,1.5), SPEED=.5, POSTURE=PRONE; /* Note that both D1 and A1 are dead.  This
      move could be initiated while A4 is at his initial placement, at (220,70) or anywhere in between
      those two points.  Also depending on kill/wound status, A4 may or may not actually move*/
   WHEN (A4 AT (255,1.5)) THEN GOTO PEN;
   WAIT;
   /* NOTE: If all defenders eventually get killed the contingency set up earlier (at label EXAM1 -- DU1
      INCAPACITATED) will still cause all attackers to run to cargo vehicle*/

AT2: GET(A2);
   WHEN (DV1 AT (255,0)) THEN GOTO AT2A; /* Wait for cargo vehicle to stop in ambush zone*/
   WAIT;
AT2A: FIRE, FIRER=A2(1), TARGET=D1;
CAN1: WHEN (D1 INCAPACITATED) THEN GOTO AT2B;
   WHEN (D1 AT (220,70)) THEN GOTO AT2C; /* If D1 makes it to the heavy brush, A2 embarks on a different
      plan of attack*/
   WAIT;
AT2B: MOVE, MOVEE=A2, GOAL=(380,25), SPEED=2.5, POSTURE=CROUCHED, WAIT; /* A2 moves into the boulder field
      to provide cover for A1 (or A4 if A1 is killed).  He does not commence firing until he reaches the
      boulder field.  (This last is the effect of the WAIT clause in the above MOVE activity
      directive.)*/
   FIRE, FIRER=A2(1), TARGET=DU2;
   /* Note:  A2 will be redirected to penetrate (at label ATT) in the event that all the defenders are
      killed.  The condition at label EXAM1 insures this.*/
   WAIT;
AT2C: CANCEL CONDITION AT CAN1; /* A2 gives up on D1 when he reaches the brush, and he does not wish to
      condition his future activities on the incapacitation of D1*/
   MOVE, MOVEE=A2, GOAL=(350,-70), SPEED=2.5, POSTURE=CROUCHED; /* A2 switches his attention to the
      defenders in the escort vehicle.  The model chooses the exact path, with regard to cover, time of
      transit, barriers, etc.*/
   FIRE, FIRER=A2(1), TARGET=DU2, WAIT; /* All members of DU2 must be incapacitated before the process
      continues*/
   FIRE, FIRER=A2(1), TARGET=D1;
   WAIT;

AT3: GET(A3);
   WHEN (DV1 AT (-10,0)) THEN GOTO AT3A; /* When DV1 enters ambush zone open fire on truck*/
   WAIT;
AT3A: FIRE, FIRER=A3(1), TARGET=DV1;
   WHEN (DV2 AT (-10,0)) THEN GOTO AT3B; /* Do the same for escort vehicle*/
   WAIT;
AT3B: FIRE, FIRER=A3(1), TARGET=DV2;
   WHEN (DV2 AT (125,-5)) THEN GOTO AT3C;
   WAIT;
AT3C: FIRE, FIRER=A3(1), TARGET=DU1;
```




```
        WHEN (D2 AT (140,-30)) THEN GOTO AT3D; /* When any of D2, D3 and D4 reach the heavy brush, A3 adopts a
            new strategy*/
        WHEN (D3 AT (140,-30)) THEN GOTO AT3D;
        WHEN (D4 AT (140,-30)) THEN GOTO AT3D;
        /* By use of additional script this could have been WHEN (ALL OF D2, D3 AND D4 -- EXCEPTING DEAD
            DEFENDERS -- REACH THE BRUSH)*/
        WAIT;
AT3D:   MOVE, MOVEE=A3, GOAL=(30,-90), SPEED=1, POSTURE=PRONE, WAIT;
        /* Move A3 to (180,-140) via (30,-90), i.e. direct him to sneak around*/
        /* Since there is no FIRE command here the most recently used FIRE command for A3 is still in force*/
        MOVE, MOVEE=A3, GOAL=(180,-140), SPEED=2, POSTURE=CROUCHED;
        /* A3 battles with those defenders he can see until all defenders are killed, at which time he is
           directed (see label ATT) to close in on the cargo vehicle*/
        WAIT;

ATT:    START ALLAT; /* The process at label ALLAT recollects all the attackers -- all defenders dead -- and
            redirects them to penetrate.  The need to start a new process to accomplish this is the most
            subtle aspect of this script.  The reason is that if D1 is killed last the conditions at EXAM1,
            EXAM2 and possibly EXAM3 will trigger simultaneously.  The script manager in TSEM does not
            guarentee which will be processed last, and consequently which MOVE command(s) will occur last.
            By recollecting all remaining attackers with a new process the order of processing becomes
            irrelevant, because a process can only direct players that it possesses.  The MOVEs we wish to
            ignore either occur first (no problem) or the process no longer possesses the player, so the MOVE
            is ignored.  The same effect could have been obtained by not starting a new process, but having
            CANCEL CONDITION AT EXAM2; CANCEL CONDITION AT EXAM3; instead*/
        WAIT;

ALLAT:GET(AU2); /* Collect all remaining attackers -- all defenders dead*/
        MOVE, MOVEE=A1, GOAL=(255,1.5), SPEED=5, POSTURE=STANDING;
        MOVE, MOVEE=A2, GOAL=(255,1.5), SPEED=5, POSTURE=STANDING;
        MOVE, MOVEE=A3, GOAL=(255,1.5), SPEED=5, POSTURE=STANDING;
        MOVE, MOVEE=A4, GOAL=(255,1.5), SPEED=5, POSTURE=STANDING;
        WHEN (A1 AT (255,1.5)) THEN GOTO PEN;
        WHEN (A2 AT (255,1.5)) THEN GOTO PEN;
        WHEN (A3 AT (255,1.5)) THEN GOTO PEN;
        WHEN (A4 AT (255,1.5)) THEN GOTO PEN;
        WAIT;

PEN:    PENETRATE; /* This code can be executed by either process 1 or process 4 (or both)*/
        WAIT;

        /* The defender section of the script is simpler*/
D:      START DF2; /* Initiate a process for the escort vehicle.  This process controls the cargo vehicle and
            its occupants*/
        GET(DV1); /* Get control over cargo vehicle and the people in the truck*/
        MOVE, MOVEE=DV1, GOAL=(-10,0), SPEED=25, WAIT; /* Move DV1 down the road until point of ambush is
            reached.  Continue processing of script only when that point is reached.*/
        MOVE, MOVEE=DV1, GOAL=(255,0), SPEED=10, WAIT; /* Slow vehicle down*/
        DISMOUNT, DISMOUNTEE=D1, WAIT; /* When vehicle reaches (255,0) start processing this section of script
            again -- vehicle stops and driver gets out.  When successfully out of truck defender races for
            heavy brush*/
        MOVE, MOVEE=D1, GOAL=(220,70), SPEED=3, POSTURE=CROUCHED, WAIT;
        /* Up until the time D1 reaches the brush he was specifically not permitted to fire.  By putting this
           fire command after the DISMOUNT he would have fired while heading to the heavy brush (if he had a
           target).  He takes up position and snipes*/
        FIRE, FIRER=D1(1), TARGET=AU2; /* All attackers are acceptable targets, assuming he has line
            of sight*/
        WAIT;

DF2:    GET(DV2);
        MOVE, MOVEE=DV2, GOAL=(-10,0), SPEED=25;
        WHEN (DV1 ATTACKED) THEN GOTO DF2A;
        WAIT;
DF2A:   MOVE, MOVEE=DV2, GOAL=(-10,0), SPEED=35, WAIT; /* Vehicle speeds up*/
        MOVE, MOVEE=DV2, GOAL=(100,0), SPEED=10, WAIT; /* Vehicle slows down*/
        MOVE, MOVEE=DV2, GOAL=(125,-5), SPEED=5, WAIT; /* Vehicle goes onto shoulder*/
        DISMOUNT, DISMOUNTEE=DU2, WAIT; /* Vehicle stops -- D2, D3 and D4 get out*/
        FIRE, FIRER=DU2(1), TARGET=AU2;
        /* Move to right tip of brush (via top of brush) and take up sniping positions*/
        MOVE, MOVEE=D2, GOAL=(140,-30), SPEED=5, POSTURE=STANDING;
        WHEN (D2 AT (140,-30)) THEN GOTO DF2B;
        MOVE, MOVEE=D3, GOAL=(140,-30), SPEED=5, POSTURE=STANDING;
        WHEN (D3 AT (140,-30)) THEN GOTO DF2C;
        MOVE, MOVEE=D4, GOAL=(140,-30), SPEED=5, POSTURE=STANDING;
        WHEN (D4 AT (140,-30)) THEN GOTO DF2D;
        WHEN (AU3 INCAPACITATED) THEN GOTO DF2E; /* When attackers not attempting to penetrate the vehicle are
            killed GOTO DF2E to adopt new strategy*/
        WAIT;
DF2B:   MOVE, MOVEE=D2, GOAL=(200,-70), SPEED=2, POSTURE=CROUCHED;
        WAIT; /* Notice that this player is waiting on (AU3 INCAPACITATED)*/
DF2C:   MOVE, MOVEE=D3, GOAL=(205,-75), SPEED=2, POSTURE=CROUCHED;
        WAIT;
DF2D:   MOVE, MOVEE=D4, GOAL=(200,-95), SPEED=2, POSTURE=CROUCHED;
        WAIT;
DF2E:   /* Move back up behind escort vehicle to get better sniping position.  Note that this doesn't happen
            until (and if) A2 and A3 get killed*/
        MOVE, MOVEE=D2, GOAL=(119,-2.5), SPEED=3, POSTURE=CROUCHED;
        MOVE, MOVEE=D3, GOAL=(118,-3), SPEED=3, POSTURE=CROUCHED;
        MOVE, MOVEE=D4, GOAL=(118,-2), SPEED=3, POSTURE=CROUCHED;
        WAIT;
```

SCRIPT FOR AMBUSH
Figure 1b

terrain, vegetation and initial placement, form a formal description of the scenario expressed above in words. A careful study of this script, which consists of 102 script language statements and took less than a man day to compose, should reveal to the reader that indeed the informal and formal descriptions of the scenario are the same. Several scripts, involving more combatants, and showing greater sophistication have been composed for various studies undertaken at Sandia Laboratories. These scripts have run 300-600 script language statements, but even the most complex have not taken more than a man week to develop and debug.

The marginal comments in Figure 1b, which are expanded on below, should aid in understanding the philosophy and use of the script definition language. The script language statements fall into three categories:

(1) Activity direction (MOVE, FIRE, DISMOUNT, PENETRATE),

(2) Process control (START, GET, WAIT), and

(3) Contingency management (WHEN, CANCEL).

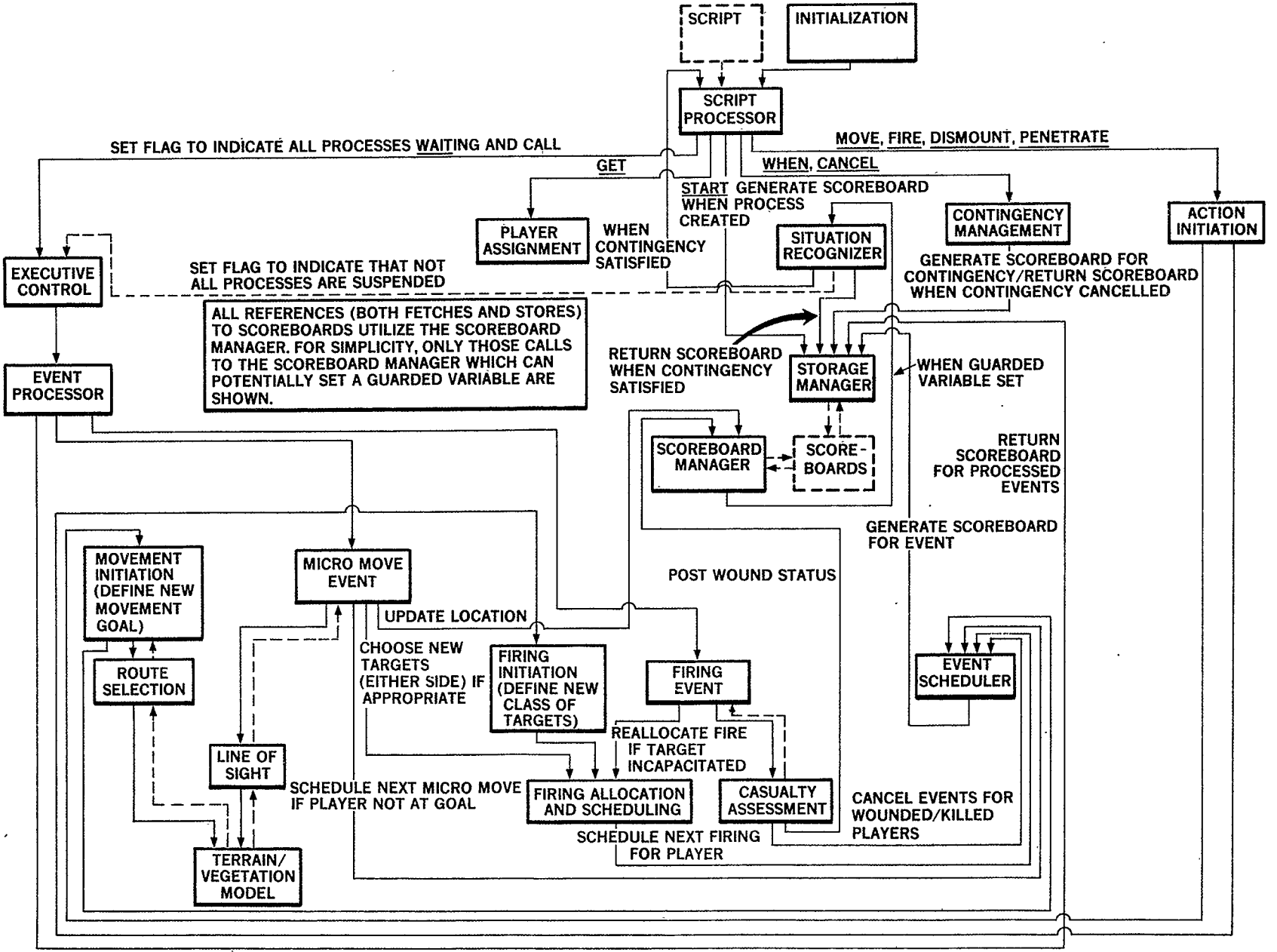
Activity direction is the most straightforward. A statement of this type attempts to initiate an action that occurs over a span of time and continues until the action is: a) completed, b) overridden by a new activity directive or c) halted because the active agent is incapable of continuing. Thus FIRE does not cause a shot to be fired, but directs the FIRER to engage in the firing activity. Based on algorithms within the model, the "best" target from amongst the specified targets will be selected. As player positions change a new target may be chosen to replace the old target, and, if line of sight is lost to all targets the firing activity will automatically be suspended, only to resume without additional FIRE commands, when a potential target reappears. The firing activity continues, repeatedly scheduling firing and casualty assessment events, until either all specified targets are incapacitated, a new FIRE command supercedes the old command, or the firer is himself killed or runs out of ammunition. A logical consequence of this is that if the firer is already incapacitated when the command is issued, the command has no effect.

MOVE is quite similar. A MOVE command does not cause the player to instantaneously appear at the goal location, but only initiates the activity of movement. The exact path between the current position and the goal, with due regard to cover, time of transit and obstacles, is chosen by algorithms within the model. The specified speed is only a nominal value, modified by terrain considerations. The movement activity is aborted (or never begun) if the player's wound/kill status, which can change during the MOVE activity, precludes further movement. Also a new MOVE command issued for the moving player overrides the current MOVE, and a new goal is established. DISMOUNT and PENETRATE have additional requirements; the combatants must be in (DISMOUNT) or near (PENETRATE) a vehicle for these directives to make sense.

The ability to express and manage contingencies is at the heart of the script definition facility. While a full understanding of the effects of the WHEN statement form requires familiarity with the concept of a process, the conditions themselves (see Figure 1b) are specified by events, i.e., changes of state that occur at a precise moment in time. The processing of a WHEN statement during script execution causes the script processor (see Figure 2) to record that the listed condition (player AT, player(s) INCAPACITATED, player(s) ATTACKED) is to be monitored. When, if ever, the occurrence of an event causes the condition to become true, the action specified in the WHEN statement (THEN GOTO...) and, consequently, the script at the appropriate label, is executed.

Changes in battlefield circumstances sometimes cause WHEN conditions which have remained unsatisfied, i.e., those for which no event has yet caused the condition to become true, to become inappropriate. CANCEL provides a mechanism for deleting these unsatisfied contingencies without activating the portion of the script pointed to by the address in the GOTO clause of the WHEN statement. This is not just a matter of housekeeping; it prevents inappropriate sections of script from executing. This situation arises in our sample scenario: A2 attempts to kill D1, but if the defender should reach the heavy brush, A2 abandons this effort and concentrates on the escorts instead. If, subsequent to reaching the brush, D1 is incapacitated by A1 or A4, this is not supposed to effect the behavior of A2. The CANCEL command at label AT2C (in Figure 1b), code which is reached when D1 makes it to the brush, allows A2 to drop his former interest in the incapacitation of D1.

The concept of process is the most abstract. In teaching about classical scientific and business programming languages for the first time, it is traditional to talk about flow of control as if there were a pointer to the currently active statement; a pointer that marches relentlessly on to the next sequential statement, unless a GOTO or other flow of



SIMPLIFIED BLOCK DIAGRAM OF MAJOR COMPONENTS OF TSEM

FIGURE 2

control statement specifies that this pointer to the active statement should be redirected. In the context of this simulation a process is an analog to the notion that a program in execution consists of the code, an associated data workspace, and a pointer to the active statement. A process consists of

(1) A pointer, which points to an active statement or is null (points nowhere),

(2) A set of players (persons or vehicles) which are possessed, and

(3) A set of unsatisfied contingencies that have been encountered.

In a manner similar to the pointer to the active instruction found in FORTRAN and the like, the pointer associated with a process moves from statement to statement. Since activity directives only initiate actions, and WHEN statements, when executed, only cause unsatisfied contingencies to be added to the set of unsatisfied contingencies, the pointer advances through the script without time advancing in the model. The WAIT statement, one of the process control statement types, plays a unique role in relation to script processing. When a WAIT statement is encountered in script processing the effect is to cause the pointer for this process to become null, and, since the pointer for this process no longer advances, the process becomes suspended.[1)]

The frequent occurrence of constructs like:

```
    MOVE, MOVEE = <player>, GOAL = (x,y),
          SPEED = <speed>;
    WHEN  (<player> AT (x,y)) THEN GOTO
          NEXTSTMT;
    WAIT;
NEXTSTMT: <any statement>,
```

where the player and the location are the same in both statements, can be avoided by use of a shorthand form provided by the input language processor. Activity directives have an optional WAIT clause, so that the above can be expressed:

```
    MOVE, MOVEE = <player>, GOAL = (x,y),
          SPEED = <speed>, WAIT;
    <any statement>.
```

1) The exact implementation differs somewhat from the explanation given here. This method of presentation was chosen because it captures the essential features of a process without reference tc internal implementation details.

The WAIT clause sets up an implied WHEN condition, where the contingency matches the objective of the activity directive and the GOTO address is the address of the next sequential script instruction. As with the WAIT statement, the process is then suspended. For example, for FIRE the contingency is <target> INCAPACITATED. A comment about CANCEL is in order here as well. CANCEL can reference a label that designates an activity directive with a WAIT clause. It is the implied WHEN condition that is cancelled, not the activity in progress.

An understanding of how WAIT causes process suspension permits the full effects of the WHEN statement to be explored. It is the mechanism by which a suspended process becomes reawakened at a later time. As model time advances, and the activities that are in progress cause events to happen, eventually an event may occur which satisfies one of a process' unsatisfied contingencies. The pointer associated with the process is then set to point to the address contained in the GOTO clause of the WHEN statement and processing of the script then proceeds. While the script is being processed time does not advance for the model. Thus we have a two phase situation: Script is processed for a while and the clock does not advance. When all processes are suspended, (by encountering WAITs) time advances and events occur until a contingency is satisfied. Then script is processed again until all processes are suspended. This cycle continues until a termination condition (all persons on one side are dead, penetration is successful, etc.) occurs and the simulation stops.

A process as defined in this environment is meant to be quite similar in nature to the manner in which a real person or group of persons operates in battle. In a real battle, the combatants engage in activities that last over a period of time and have goals which are more or less clear. While the activities are in progress the combatant is continually monitoring the situation, searching the environment for events that will cause him to stop and rethink his strategy. This constant reevaluation sometimes initiates a new set of activities. Besides being analogous to human approaches to task selection during battle, a process in TSEM is patterned after the notion of a process found in operating systems. Those readers familiar with this computer science discipline should see the parallel at once. Like a process in the operating system sense, a process in TSEM's scenario description facility must possess resources. This is the purpose of the GET statement. It allows a set of players to be associated with a process. This brings up a subtle point: The active agent in an activity directive must be possessed by the process for the action to be carried out. The

ability to issue GETs subsequent to the start of a process and to condition GET statements on the occurrence of certain events (by use of WHEN statements) allows for regrouping of forces if the situation should warrant it. On the other hand the passive agent in a FIRE command and the players involved in contingency specifications do not need to be possessed by the process. This is in keeping with the view that a process models the tactical behavior of a combatant or group of combatants. The events that cause combatants to rethink their plans do not necessarily involve them directly.

Processes are created using the START statement. Since the number of functional units on each side depends on the scenario, the number of processes the model should support, cannot be fixed in advance. A process creation facility removes any limitations in this area. The only requirement is that the first statement of the script start up one attacker process and one defender process. Additionally, processes started up after these two initial processes inherit the side affiliation of the parent process.

The above description explains how a process acts in isolation; the total view requires an understanding of how they act in parallel and how they interact, or to use the language of operating systems, how they communicate. The computer code keeps a process scoreboard, created when a START is encountered, for each process, so that each process operates essentially independently, but in parallel with other processes. The communication is indirect, through activities and contingencies. For example, one or more processes may have the line

```
WHEN (DV1 AT (255,0)) THEN GOTO...,
```

as do processes ① and ②. However, it is process ⑤ that directs the actions of the vehicle. Thus these processes communicate through events that occur because of activity direction within one process. An important feature to notice is that the order of execution of the statements in two different processes cannot, in general, be predicted, since the stochastic nature of the firing and casualty assessment models determines the order in which the unsatisfied contingencies will become satisfied. Thus the actions specified at label AT1A and AT3B can be initiated in either order, depending on the initial placement of the two vehicles, and the actions at labels DF2E and AT1D can occur in either order, if they occur at all, depending on whether AU3 is incapacitated before or after both D1 and A1 are incapacitated. By choosing different initial seeds for the random number generator, the order of execution of these statements, if they execute at all, may well be different from replication to replication. The different sequences of actions initiated, resulting from the triggering of different WHEN conditions or their triggering in different orders--all due to the stochastic nature of some of the submodels--can yield dramatically dissimilar battles while using the same script. By running a sufficient number of replications for a given scenario, however, both the extreme outcomes and the general tendency can be explored. By repeating this procedure for a set of appropriate scenarios, the analyst can get some measure of overall system vulnerability.

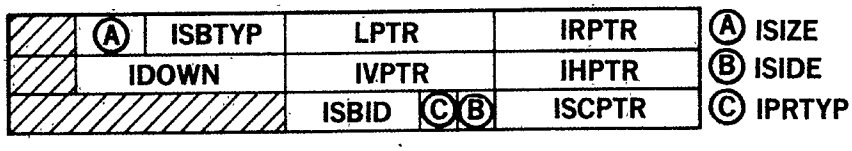
By keying on the same condition, as processes ① and ② do when they key on

```
WHEN (D1 INCAPACITATED) THEN
     GOTO...    ,
```

sections of two different processes can be logically activated simultaneously. Of course, one of the two script sections at labels AT1C and AT2B will be interpreted first on a sequential computer, but since model time remains unchanged during script processing, and both processes become ready to execute due to satisfaction of the same contingency, the two processes appear to be operating simultaneously. Logically simultaneous actions performed sequentially can on occasion lead to anomalous results if the user is not careful. This problem is not unique to the TSEM script facility; it is common to a number of event driven simulation languages that support contingencies (see, for example, [3]). In TSEM, the difficulty arises when two activity directives of the same type are issued, logically simultaneously, for the same player at two different places in the script. Since they occur simultaneously in model time the TSEM script manager does not guarantee the order of their execution, and since the activity directive to execute last takes precedence, the action that becomes effective cannot be predicted. The CANCEL command, discussed earlier, gives the user considerable control over the precedence assigned to logically simultaneous actions. The implementation of CANCEL not only allows contingencies yet to be triggered to be ignored; it can be used to inhibit script processing for those sections of script not yet executed, but made ready by recognition of a condition. The comment at label ATT in the script of Figure 1b discusses this problem in the context of a specific example. Thinking about parallel processes and their interactions takes some getting used to, but the approach discussed here does model how combatants operate during battle. Each person, while he might have group goals in mind, performs his task in parallel with all other combatants.

Because FORTRAN, the implementation language of TSEM, does not provide primitive language constructs for record defi-

nition, parallel process execution or contingency recognition, a discussion of the implementation may shed additional light on how the script concept is supported. Figure 2 shows a block diagram of TSEM. The uniformity gained by the consistent use of the utilities in the record management subsystem (which consists of the scoreboard manager, storage manager, internally encoded record definitions, and primitive data handling functions) provides the basic capabilities necessary for implementation. Every player, process, scheduled event, and unsatisfied contingency has associated with it a scoreboard, where all information about this entity is stored. For example, Figure 3 is the template for the process scoreboard; each process specified by the script has an area of storage assigned to it, laid out in the indicated manner, at the time of process creation. All fetches and stores to scoreboards pass through a few basic subroutines (scoreboard manager) that, using the template definitions, optionally perform type checking (an option which has greatly simplified debugging of the model), pack, unpack and, if necessary, convert the data, and act as interfaces between the data structures and the higher level routines. The fact that all references to the data describing the state of the model pass through one channel allows the implementation of guarded variables. This technique permits extension to user defined conditions, the automatic monitoring provided by the system defined ON conditions of PL/I [1], and is essential to the efficient recognition of contingencies. The scoreboards associated with players contain a special field of bits; each bit associated with one field of the scoreboard template. Whenever a store into a field of a player scoreboard is made, the subroutine within the scoreboard manager responsible for handling stores into scoreboards checks to see if the bit associated with that field is on; if it is, then the situation recognizer is called. The situation recognizer then checks to see if the actual values stored cause any of the unsatisfied contingencies to be satisfied, which in turn will cause the script processor to be invoked, and script to be processed.



Field Name	Size in Bits	Functional Description
IRPTR	18	Pointers to maintain two way linked list of processes
LPTR	18	
ISBTYP	12	Identifies type of scoreboard--in this case a process scoreboard
ISIZE	6	Size of scoreboard in words--for a process scoreboard this is 3
IHPTR	18	Pointer to top of two way linked list of persons attached to this process
IVPTR	18	Pointer to top of two way link list of vehicles attached to this process
IDOWN	18	Pointer to maintain one way linked list of processes which are ready or active
ISCPTR	18	Pointer to script instruction being executed by this process
ISIDE	3	Identifies process as an attacker/defender process
IPRTYP	3	Identifies process as active, ready or suspended
ISBID	12	Identification number for this process

Template for a Process Scoreboard
Figure 3

The internal functioning of a WHEN, therefore, is: At the time the WHEN is encountered during script processing, a situation scoreboard is created. It contains a description of the condition, a pointer to the process for which this contingency is unsatisfied, and a pointer to the section of script that is to be executed when the contingency is satisfied. In addition, in the scoreboard of the players involved in the contingency the bit field that the scoreboard manager uses to implement guarded variables is updated to reflect which fields of the scoreboard are to be monitored. When, at a later time the event which satisfies the contingency occurs, the bit within the bit field is reset (unless some other contingency is also monitoring that field of the player), time is suspended

while the associated section of script is executed, and the storage for the situation scoreboard is returned to the storage manager.

The use of storage management routines and pointers, and the specification of the individual fields within scoreboards as offsets, instead of using specially named arrays with fixed bounds, allows an indefinite number of processes, as well as an indefinite number of unsatisfied contingencies, to be present at any one time. Since pointer oriented access and management of storage are well understood, no further remarks about this aspect of the implementation are provided here.

CASUALTY ASSESSMENT SUBMODEL

The clear separation of specification of tactics from more elemental functioning in the combat environment brings the submodels that implement the tasks internal to the model into sharp focus. Whenever it seemed appropriate, the results of research done for similar models [2,9] were utilized. For example, much of the terrain and vegetation submodels used in SIAF [9] have been incorporated into TSEM, after minor modification allowing coordination with the TSEM data structures. However, because of

(1) the relatively few number of combatants anticipated for TSEM applications,

(2) the scenario based nature of the model, which can condition many subsequent actions on the incapacitation of a single individual,

(3) the unbalanced nature found at the beginning of an engagement, when the occupants of vehicles become exposed to fire without benefit of cover and in unfavorable postures,

(4) the expected physical interspersing of combatants resulting from ambushes of road convoys, and the subsequent exposure to fire from different angles, and

(5) the desire to gain information on battle times and final adversary force levels should the guards escorting the convoy be defeated,

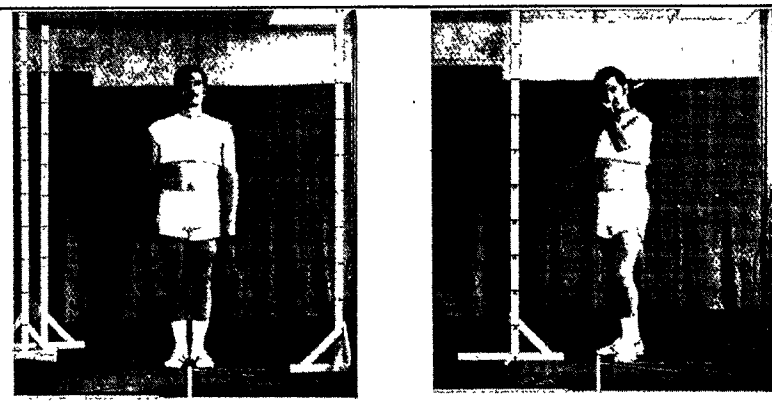
a highly refined casualty assessment submodel was deemed desirable. A careful study was made of the casualty assessment submodels of SIAF [10] and ASARS [2], the two most closely related models, before deciding to develop a new casualty assessment model from first principles.[2)]

SIAF models people as rectangles, the height and width of which are inputs. Shielding and posture are accounted for by adjusting the width and height of the rectangle accordingly, yielding a smaller target area. Based on the dispersion data for weapons, and an assumed aim point in the center of the visible area, a probability of hit is generated. If a round does score a hit the probability of kill is determined by a formula that takes into account weapon type, posture, and range. The varying lethality of a hit on different body parts is only indirectly included in the probability of kill computation, by making posture a parameter of this calculation. As Figures 4c-e show, aspect angle, which was not considered by SIAF, plays a significant role in determining the presented body area. The aspect angle affects the presented area for standing and kneeling postures by as much as a factor of 1.41; prone is much worse, the ratio of presented body area, side view to front view, is 2.81. Figures 4f-g also demonstrate that the fractional amount by which to decrease the width of the target due to immediate adjacent vertical shielding is not the .5 that first comes to mind and is used by SIAF. The use of shielding also confounds the issue by changing the percentage of exposed area due to each class of body part. Thus the probability of kill given a hit for the shielded case cannot be computed using the formula for the unshielded case. One further concern that should be mentioned is the choice of aim point. A standard training doctrine is to use the center of the thorax, if exposed, and the center of the head if it is not. Figure 5 shows the results of a study comparing the SIAF casualty assessment submodel to that employed by TSEM.

ASARS uses a more refined model, dividing the human body into five body part types, each approximated by a rectangle. The dispersion data is used to determine which, if any, body part is struck, and then conditional lethality data is used to calculate the effect of the round. Shielding is accounted for by reducing the vulnerable area on a body part by body part basis. Four postures are supported in the model (standing, kneeling, prone, and foxhole). This model is more accurate than the SIAF model, although aspect angle is still ignored, the presence of immediate adjacent vertical shielding reduces vulnerable area by half, and the model does not account for repositioning of body parts when shielding is used. Also of concern is the choice of the figures from

---

2) The work of the BDM Corporation, operating under contract 03-9584, in aiding in the development of this casualty assessment model, is gratefully acknowledged.



(a)

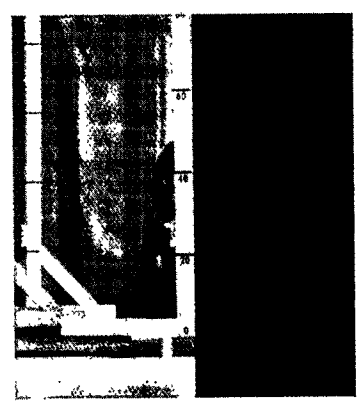
(b)

(c)

(d)

(e)

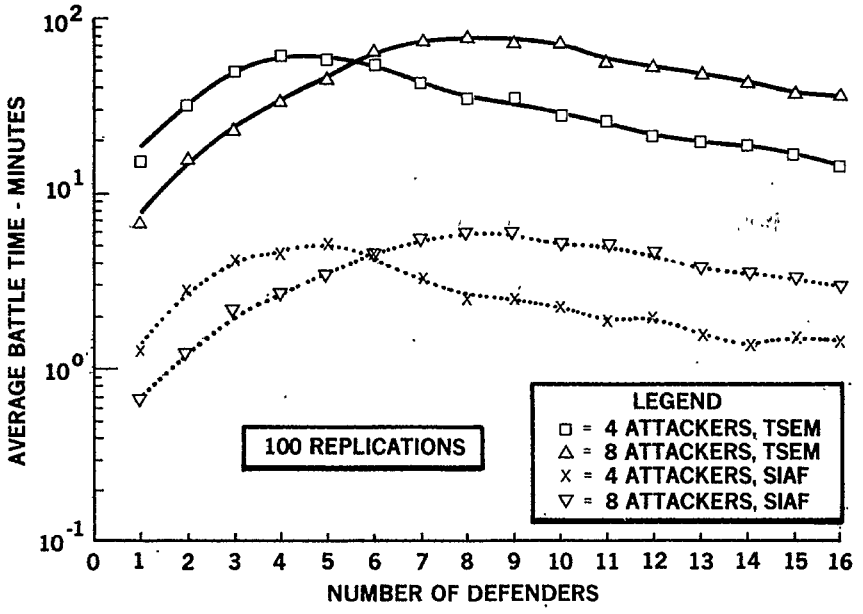
(f)



(g)

Photographs Exhibiting Postures
Figure 4

which posture data was derived. The documentation [2] that discusses the lethality model shows as its only pictorial example the use of the standard anatomical posture (Figure 4a), which exposes 1.51 times as much area as the target in Figure 4b. The associated data statements confirm that standard anatomical posture is in fact used for the standing posture; the arms hang by the sides and the visible area is almost exactly the same as that derived by the methods discussed below when Figure 4a is used. On the other hand, the total amount of exposed area for the prone posture is only one half that derived by using Figure 4f.



COMPARISON OF TSEM AND SIAF CASUALTY ASSESSMENT MODELS
COMBATANTS PRONE WITH IMMEDIATE ADJACENT VERTICAL SHIELDING, RANGE =100 METERS

Figure 5

The concerns listed at the beginning of the section, concerns that may not have been as important in SIAF and ASARS as they are in TSEM, led to development of a more detailed casualty assessment model, generally along the same lines as used by the developers of ASARS. The raw data used included:

(1) photographic studies of a combat veteran [8],

(2) conditional lethality data for various body parts [5,7],

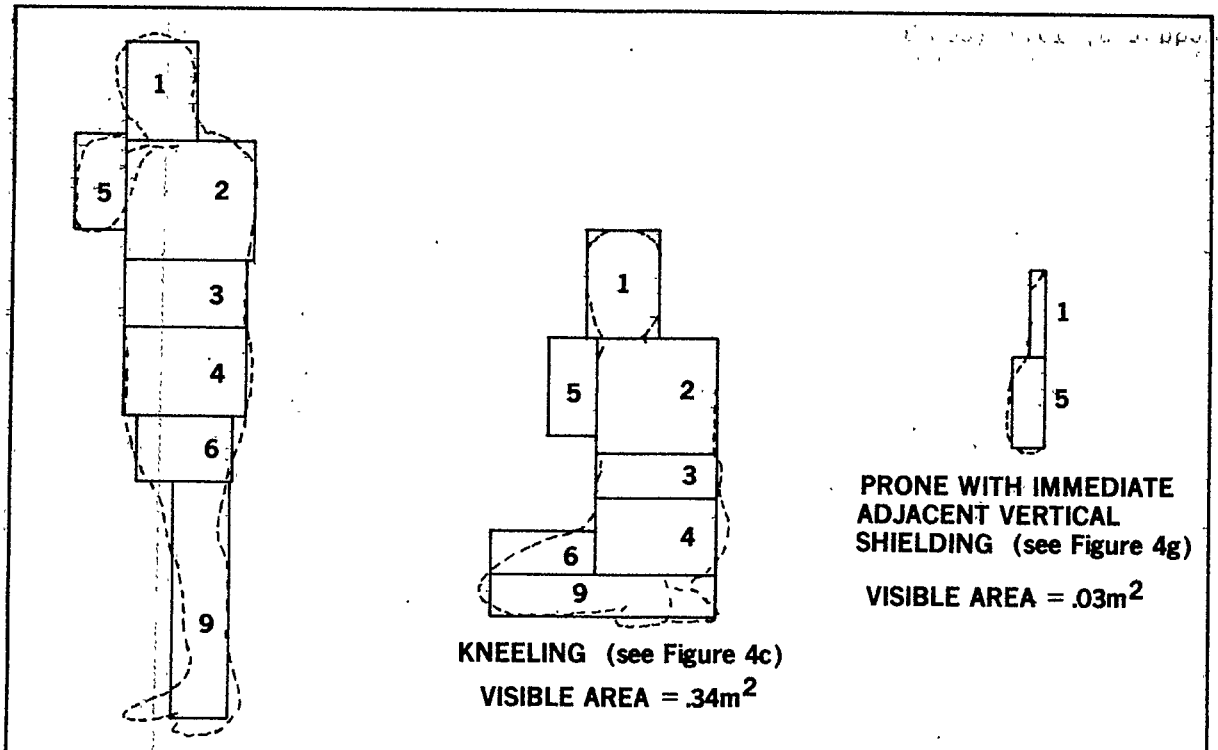
(3) weapons dispersion data [6], and

(4) army field manuals.

The human body was divided into ten rectangles. Each photograph was then analyzed; for each of the ten modeled body parts (i.e., rectangles) a calculation was done to determine the amount of exposed area and its location relative to the preferred aim point. Figure 6 shows the decomposition of three postures into their equivalent rectangle formulation. This information is stored within the casualty assessment submodel, and, like ASARS, each round is evaluated to determine what, if any, body parts are hit. Conditional lethalities, which are a function of range and weapon, are then used to determine the effect of the round. Because of the changed ratios of visible area by body part type when shielding is employed, shielding is treated as defining new postures. This improved casualty assessment model has found acceptance in similar computer models [4,12] currently under development.

In closing, mention should be made of how factors introduced by other submodels affect the casualty assessment model, and what research needs to be done to obtain additional information that will improve the casualty assessment submodel and human factors modeling in general. Sandia Laboratories is currently developing an instrumented Small Force Engagement Range which will be used to gather a wide variety of data. Utilizing harnesses worn by combatants, equipped with sensors and electronic hardware, and M16s modified for blank fire and equipped with laser transmitters, the Small Force Engagement Range allows simulated battles to be run with a sense of realism not previously possible.[3] A number of formal studies, yet to be undertaken, will attempt to measure the human factors effects on combatant behavior and decision making. Anecdotal observations suggest that the weapons accuracy information will need to be improved to reflect the firing strategies employed in small force combat situations. When not engaged in suppressive fire, the firing activity of skilled combatants appears

3) The equipment currently being used on the Small Force Engagement Range was developed by Xerox Electro-Optical Systems, Inc., for the U.S. Army under contract N61139-76-C-0060.



DECOMPOSITION OF POSTURE PHOTOGRAPHS INTO RECTANGULAR AREAS

Figure 6

to consist of a few carefully aimed semi-automatic bursts widely spaced in time. Another area of concern is the interaction of the movement submodel with the casualty assessment submodel. Little is known of the strategies employed in moving from one location to another. Such basic parameters as the percentage of time spent behind cover and the postures assumed during a protracted move have yet to be determined. Additionally, the transmittal letter accompanying document [6], indicates that no information is available on the magnitude of the increased dispersion due to a moving target. Indirect effects, such as the effect of movement on target detection and the influence of skill and suppression levels are even less well understood. The knowledge gained from the studies Sandia Laboratories is planning to conduct on the Small Force Engagement Range should be of value to the entire modeling community.

BIBLIOGRAPHY

[1] American National Standard Programming Language PL/I, American National Standards Institute, X3.53, 1976.

[2] ASARS Battle Model, United States Army Combat Development Command, Systems Analysis Group, Report no. USACDCSAG-TR-9-73, March 1973.
"Book 2, Volume II-A, Narrative Description"
"Book 5, Volume II-B, Phase B and C Charts"
"Book 9, Part 1, Volume III, User Manual"

[3] Dahl, O. J. and K. Nygaard, "SIMULA--an ALGOL-Based Simulation Language," Communications of the ACM, Volume 9, no. 9, September 1966, pp. 671-678.

[4] De Laquil, P., III, SABRES II: An Individual Resolution Small Arms Combat Simulation Model, Sandia Report no. 79-8249 (in preparation).

[5] Dzieman, A. J. and A. G. Oliver, Wound Ballistics Assessment of M14, AR15 and Soviet AK Rifles (CONFIDENTIAL), Chemical Research and Development Laboratories, Edgewood Arsenal, Maryland, November 1964.

[6] Data from Garrett, A. W., Acting Chief Ground Warfare Division, Department of the Army, U. S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, Maryland, August 5, 1977 (CONFIDENTIAL). (Data also reported in Branch, G. A. and T. J. Thurgate, Small Arms Weapons Data and Analysis for Combat Simulation Model (CONFIDENTIAL), SRI International, Menlo Park, California, Report no. SRI 78-719, November 1978.)

[7] Kokinakas, W. and J. Sperrazza, Criteria for Incapacitating Soldiers with Fragments and Flechettes (CONFIDENTIAL), Ballistics Research Laboratories, Aberdeen Proving Ground, Maryland, Report no. BRL 1269, January 1965.

[8] The Probability of Single Shot Incapacitation, P(SSIN), Algorithm for TSEM, Report no. BDM/A-77-274-TR, The BDM Corporation, Albuquerque, New Mexico, June 1977.

[9] SIAF Model Development Validation and Implementation, Final Report, TRW Systems Group, August 1971.
Volume I, Report no. 16905-6012-R0-00
Volume III, "Model Subroutines (Terrain, Weather, Targets)," Report no. 16905-6014-R0-00

[10] SIAF System Model User's Manual: Small Independent Action Forces, TRW Systems Group, Report no. 20660-6007-R0-00, Volume VI, "Combat Execution Subroutines," December 1973.

[11] Stockfisch, J. A., Models, Data, and War: A Critique of the Study of Conventional Forces, Report no. R-1526-PR, Rand Corporation, Santa Monica, California, March 1975.

[12] Wagner, N. R., Small Arms Casualty Effects Model (SACEM) (CONFIDENTIAL), Sandia Report no. 79-8018.