

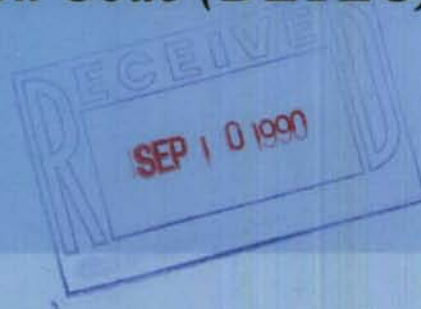
**LA-10547-MS**

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

**DISTRIBUTION STATEMENT A**  
Approved for Public Release  
Distribution Unlimited

***Conceptual Specification for  
Defensive Technology Evaluation Code (DETEC)***

**20070822088**



**Los Alamos** Los Alamos National Laboratory  
Los Alamos, New Mexico 87545

UL ~~504007~~

**DISCLAIMER**

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

LA-10547-MS

UC-32

Issued: January 1986

## **Conceptual Specification for Defensive Technology Evaluation Code (DETEC)**

Ronald D. Christman  
Arthur L. Dana  
Dale B. Henderson  
Barry P. Shafer  
Juan A. Wood  
Merri M. Wood

## CONTENTS

<b>1. ABSTRACT</b>	<b>1-1</b>
<b>2. INTRODUCTION</b>	<b>2-1</b>
<b>3. DESIGN APPROACH</b>	<b>3-1</b>
<b>4. SCENARIO EXAMPLE</b>	<b>4-1</b>
<b>5. LEVEL 0 DATA FLOW AND DESCRIPTIONS</b>	<b>5-1</b>
<b>6. LEVEL 1 DATA FLOW AND DESCRIPTIONS</b>	<b>6-1</b>
<b>7. LEVEL 2 &amp; 3 DATA FLOW AND DESCRIPTIONS</b>	<b>7-1</b>
<b>8. DATA DEFINITIONS</b>	<b>8-1</b>
<b>9. APPENDIX</b>	<b>9-1</b>
<b>A. Structured Analysis Conventions</b>	<b>9-2</b>
<b>B. Naming Conventions</b>	<b>9-4</b>
<b>C. Alternative Proposals</b>	<b>9-5</b>

## CONCEPTUAL SPECIFICATION

for

### DEFENSIVE TECHNOLOGY EVALUATION CODE (DETEC)

by

Ronald D. Christman, Arthur L. Dana, Dale B. Henderson,  
Barry P. Shafer, Juan A. Wood, and Merri M. Wood

#### 1. ABSTRACT

The Defensive Technology Evaluation Code (DETEC) is being developed to assess the potential of a realistically diverse assortment of strategic defense and offense assets deployed on all sides in possible global conflict. Principal applications of the code will be to study the roles of the various weapon technology concepts being explored for strategic defense. Technology requirements and sensitivities will be studied in the context of complete wars built up from many individual one-on-one engagements. DETEC will also provide an important vehicle with which to develop and test various possible algorithms for battle management and for communications and control.

The DETEC simulation features a wide span of capabilities. It includes each important object with separate modular representations of each replica decoy, re-entry vehicle, missile, submarine, satellite, etc. Such complete offensive and defensive inventories are employed on all sides of an arbitrarily many-sided conflict. The one-on-one engagement modules are statistical processes based upon accurate physical models. Damage to individual assets is simulated by operating parameters that may be continuously varying, giving simulated performance from full to zero capability.

Countermeasures are explicitly included. Each sensor, weapon, and control and communications system is modeled by a library of code modules allowing the user to balance the needs of his problem, the fidelity levels, and the computational costs. Stresses on the operating environment include both natural (sun, moon, storms) and battle-driven (jamming, weapon, nuclear) effects. DETEC is event driven, with both instantaneous and time-extended events allowed. Conflicts between extended events and instantaneous or other extended events are identified and explicitly accommodated. Separate files are maintained for the "real" data, the perceived data of each combatant, and that of each combatant's subsystems.

The modular code structure is developed for efficient execution on our Cray-X computer. A true restart capability allows a simulation to be restarted with or without modifications, which may include arbitrary interventions. Convenient user interaction and powerful graphics-based postprocessing are important design functions. Although we draw on the experience of existing military simulations, DETEC is a new product that relies heavily on our experience with efficient supercomputer codes for the design of nuclear weapons.

Our design procedure has utilized both the formal methods and concepts of structured program design. The substance of this document describes the design specifications (data flow, data definitions, and control flow).

Ideas and suggestions for this document have been incorporated from a large group of people including Laboratory consultants, other employees, researchers at many military operations research establishments, and their contractors. For this help and encouragement we are grateful.

---

ABSTRACT

## 2. INTRODUCTION

The Defensive Technology Evaluation Code (DETEC) simulation is an analytical tool for general application to systems studies within the strategic defense program. The code is intended to be capable of simulating engagements between any arbitrary strategic offensive and defensive systems, given code modules describing the set of weapons effects of interest against assets of interest. In order to do this, DETEC has been designed as a very general simulation system with scenario-dependent coding confined to modular code subunits that may be "plugged into" the general DETEC framework. It is intended that a large catalog of the individual modules will be acquired, and it is hoped that many of them will be contributed by members of the community outside the DETEC team and outside Los Alamos National Laboratory. The capability to simulate virtually any arbitrary engagement requires treatment of the entire earth and its near vicinity (at least out to geosynchronous orbit) and the ability to simulate and/or keep track of a large number of assets. Estimates range from some thousands of objects in a limited engagement to upwards of fifty thousand in a full-scale exchange.

The corresponding code may therefore be thought of as a large, specialized data base code, with the state of the "world" being represented in several million words of computer memory. Several computer processes are invoked to set up this data base, others are invoked to move it forward in time, while still others are used to report the progress to the users both interactively and through various postprocess reports.

DETEC simulation capability allows a user to investigate the efficacy of weapons, sensor, battle management, and communications systems, as well as the effect of policy and strategy. The sensitivity of a system's effectiveness to technology assumptions and design and deployment parameters can be studied under full campaign conditions. Both natural (e.g., weather, electromagnetic environment) and engagement-induced stresses (e.g., nuclear effects, electronic warfare) are simulated. Countermeasures can be explicitly simulated. The modular nature of individual assets and physics simulators allows them to be written to the fidelity required for a particular application without modifying any other part of the DETEC system. Medium to high-fidelity battle management simulation requires the concept of "perceived worlds" as opposed to the "real world." This refers to the inevitable incompleteness and inaccuracy of the information available to a battle manager—his "perceived world" as derived from his sensor. In contrast, the code will maintain a data file containing an accurate (by definition) description of each object included in the simulation. This file is termed the REAL-WORLD.

Each object description is a data structure that has been named a STATE\_VECTOR. DETEC STATE\_VECTORS are hybrid structures—they contain a complete parameterized description of an object's state and kinematics (like a quantum mechanical state vector), and they also contain an evaluation, for a particular instant in time, of a subset of the possible descriptive variables. This subset will always include position and may include other quantities, depending on the class of object described.

The volume of space from the surface of the earth out to some user-defined limiting radius may in DETEC be divided into some number of angle and radius limited sectors. This, like the evaluation of the position, is a calculational efficiency feature. The sector in which an object is located will be stored in the object's state vector and changed as updated positions require. The sector locations are used to limit the number of state vectors that must be considered for position-dependent calculations (such as what a sensor can see or with what a weapon effect will interact). The positions data in the state vector can be used as is for relatively slowly moving objects, or it can be updated to the exact current simulation time if the physics and fidelity require it.

Geographical and environmental data will also be referenced by sector. Environmental influences at a particular point will be treated as caused by two additional components; a slowly varying component that is calculated at routine update intervals and parameterized by sector, and a rapidly varying component that

## INTRODUCTION

## 2. INTRODUCTION

is calculated from individual "source" state vectors as needed.

The processes simulated by DETEC may be considered to fall into two classes: those whose time evolution is (relatively) easily parameterized and predicted and those whose time evolution is potentially complex. Examples of the first class of processes are both powered and free motion, the development and evolution of a localized weather system, and the expansion of a nuclear fireball. Examples of the second class are the functioning of a sensor, weapon, battle manager or communications installation, or the interaction of a weapons effect with an asset. Processes of the first type are not explicitly simulated by DETEC; their behavior is built into pertinent subroutines by algorithms using state vector parameters to provide specific evaluations as needed. Changes in these parameters (analogous to a transition between quantum mechanical states) are explicitly simulated. Processes of the second type compose the bulk of DETEC's simulation task. These calculations are performed by individual asset and physics simulation modules chosen from the DETEC module catalog. These simulation calculations are triggered by a precalculated instruction termed an EVENT.

An EVENT is a data structure containing a specification of what asset or physical process is to be simulated and when it is to occur. EVENTS can be created at code initialization or directly by the user, but, as a rule, they are constructed by asset or physics simulation modules themselves.

DETEC calculates what subsequent occurrences are required (as in the case of battle managers) or caused by a particular simulated action, and EVENTS to trigger the simulations of these processes are constructed. EVENTS may be treated as either instantaneous (being of negligible duration) or extended. Instantaneous events are marked by a single EVENT. Extended events require an EVENT to mark the beginning of the process and a second EVENT to mark its end. It is possible for an extended event to occur over a period during which the state of an asset involved in the process is being changed by another event. Such situations have been termed conflicts and are identified and resolved by one of two methods. If the structure of the conflict allows and the specific code modules are able, a conflict may be resolved by saving copies of the changing state vector. This information would then be used to calculate, at the simulation time corresponding to the end of the event, the process simulation for the entire time period of the event. Otherwise, the conflicting events are divided into a number of time steps suitable for the characteristic times of the processes involved, and the state vectors are used as constants during each time step.

The time-ordered series of EVENTS is termed the EVENT\_Q, and the simulation proceeds from event to event in the queue, with no calculational investment in the parameterized "easy" processes between EVENTS.

The EVENT\_Q also contains EVENTS that are, using the above definition, "nonevents." These are markers that trigger periodic displays, logging functions, and updates of the evaluated portion of the state vector and environmental data. Their inclusion as "nonevents" or "pseudo-events" allows a simple control structure for the main execution loop.

The general DETEC code structure reflects a division by function into five main units. The user interface and code initialization are performed by the code MANAGER. The user interface accommodates interactive graphical setup, checkout, and monitoring functions. The setup of a simulation requires specification of the order-of-battle (numbers, characteristics, and employment of assets); battle manager "policy" instructions (parameter sets tailored to the design of the particular module selected); physics and asset module selection consistent with the preceding natural environment parameters; and any specific events desired by the user. This may be accomplished by commands entered directly from the terminal or by assigning a file containing a list of commands (an INFILE). Setup may be "from scratch" or based on any other partial

## INTRODUCTION

## 2. INTRODUCTION

or complete DETEC problem specification defined through a restart file or one or more INFILES.

A restart modification is conceptually identical to a referee-type intervention, and, in DETEC, this intervention mechanism will allow user intervention in virtually all aspects of the simulation.

The code initialization functions performed by the MANAGER include construction of the REAL\_WORLD and PERCEIVED\_WORLD files, as well as the initial EVENT\_Q and setup of code and system parameter tables. Module and data libraries will be assigned.

Running and monitoring parameters, such as selection of a graphics display, how frequently to update the display, the type and frequency of data logging, and how often to write restart dumps will be part of a simulation run setup. The user also specifies the set of optional variables to be written, in addition to a large set of standard variables, into postprocessor files. These data provide periodic "snap-shots" of the simulation. From these files, the postprocessor can provide the following: graphical or tabular data for specific simulation times, time histories of simulated quantities, and comparisons of these data for different times, variables, or different DETEC simulation runs. For convenience, the postprocessor has the same user interfaces as the main DETEC code, with all common function commands being identical.

Control functions for the DETEC simulation execution are centralized in the code, EXECUTIVE. The EXECUTIVE sequentially processes the events in the EVENT\_Q, identifying and calling the appropriate code unit to execute that particular event. For each extended or instantaneous event, the executive identifies conflict situations and either provides for storage of changing state vectors or constructs an event to trigger the subdivision of one or more extended events into shorter events in order to resolve the conflict as discussed above.

The periodic evaluation of some parameterized state vector variables is performed in a code unit nicknamed MOTHER NATURE. This code unit is itself functionally divided, with modules for each class of state vector and environmental phenomena. Modules with varying degrees of sophistication may be selected at run time to allow choices of fidelity and concomitant computing costs. The time periods between regular updates are chosen to reflect the characteristics time scale for the processes involved.

Each time MOTHER\_NATURE is called, it calculates an update interval and puts an appropriate EVENT in the EVENT\_Q. Each MOTHER\_NATURE call will be for a set of classes of STATE\_VECTORS and/or environmental quantities, and every member of each of these classes is updated in a single call. Individual MOTHER\_NATURE subroutines may be called by other code modules to allow accurate evaluation of critical or sensitive parameters, but the updated information is used only locally and is not stored in the REAL\_WORLD.

The salient features of the engagement simulation are accomplished in the portion of the code termed ENGAGEMENT. ENGAGEMENT contains the asset simulation modules and communications and interaction physics subroutines. It is itself divided by function into BATTLE MANAGERS, SENSORS, COMMUNICATIONS, WEAPONS, AND EVENT PHYSICS.

These names are applicable in the very general sense: any analytical function, be it human or a microprocessor, is termed a battle manager, and any means of acquiring information is a sensor. Because of this structure, co-located but distinguishable functions of a particular installation (such as the sensor, communications, and analysis functions of an intelligent surveillance platform) will be simulated separately. All communications are treated as discrete messages contained in data structures named INFO\_PACKETS.

## INTRODUCTION

## 2. INTRODUCTION

It is in the ENGAGEMENT that the modularity of DETEC is most important, because the simulation of a variety of systems will require a wide range of asset simulation capabilities. Studies with a different emphasis will, in general, have different fidelity requirements, all of which are accomplished by selecting modules suitable to the application. The array of modules is specified by the user as part of the setup processes, and DETEC extracts the required modules from the library and links them as part of the run initialization.

The data LOGGER is the fifth main unit of DETEC. It may be called directly by the user from the MANAGER or, periodically, by the EXECUTIVE during execution. In addition to writing a variety of code data to one or several output files, the LOGGER creates user displays, restart files, and postprocessor files.

## INTRODUCTION

### 3. DESIGN APPROACH

In order to develop high-quality and maintainable software, the DETEC project is utilizing structured design and implementation techniques organized into a project development cycle. This approach is motivated by the following numbers for typical software projects: In a normal development project 15% of the time is spent on programming while 50% is spent on debugging. The average electronic data processing (EDP) organization spends 50% of its software budget on maintenance. About 80% of software maintenance cost is repairing design errors. Using the structured techniques, coupled with careful reviews reduces the number of design errors and produces high-quality and maintainable programs. The cost over the lifetime of the software is expected to be significantly less than that of programs developed using conventional techniques.

The development cycle for the DETEC project is proceeding as described below:

1. Define requirements: This step involved talking to potential users and visiting other sites that might have a competing product. No formal documentation was developed as a result of this step. It did provide essential input to the steps below.
2. Do a high-level logical design: This step resulted in a design that described the logical operation and functions of the simulation. It described how the product would work independently of the target operating system and implementation languages. The output from this design step is data-flow diagrams showing the processes required for the system and the data flow between the processes. The processes were defined using verbal descriptions and a Problem Definition Language (PDL). The standard form of the PDL is defined in the internal project guidelines document, Ref. 1. The processes and data are defined to a level of detail required for an adequate description. The method used to develop this specification is called Structured Analysis and is given in Ref. 2. The specification you are reading is the output from this step. An important part of this step is several reviews and walkthroughs to determine if the proposed product meets requirements and will perform as expected.
3. Convert to a high-level physical design: This step converts the logical design into a structured design that incorporates the details of the target operating system and implementation language. Using structured design techniques allows the design to be completed and the result evaluated. The documentation from this step is a set of structure charts that show the hierarchical structure of the program modules and their data flow. The data definitions and process definitions from the logical design are refined as necessary. Internal project reviews of this step are conducted as sections of the program design are completed. The method used to develop this specification is given in Ref. 3 and 4.

Step 4, 5, and 6 are done as functional sections of the program are designed and implemented. For example, the detailed design for a battle-manager simulator would be done followed by coding, checkout, and putting the new function on the system. This is then repeated for other functions.

4. Prepare the detailed design: This step completes the detailed design of the internals of subroutines and other modules. This is the precise solution to the problem. The level of detail is such that these specifications can be given to a good software technician for coding, integration into the system and top down checkout. The detailed specifications are done using the PDL notation developed by the project, Ref. 1. Reviews are conducted on detailed specifications as sections of the specifications are completed.
5. Code and check out the detailed design: This step involves converting the PDL to Cray FORTRAN, having a clean compile reviewed, installing the code into the test system, and checking it out. This is then followed by another review if major changes were made. The coding is done using the standards for Cray FORTRAN given by the project guidelines document, Ref. 1.
6. Place the program under change control: The program is placed under change control using the HIS-

### DESIGN APPROACH

### 3. DESIGN APPROACH

TORIAN system, Ref. 5. It becomes available to the users when the next new system is brought up. The method for implementation change control has not yet been specified.

This specification was developed using the design technique called Structured Analysis, Ref. 2. The standard notation that we adopted is given in Appendix A. There are several parts to the specification. The most obvious is a data-flow diagram. Here the processes (modules) composing the system are shown along with the data that are being transferred between modules and to/from files and data sources/sinks, e.g., users. This is a model of the system from the perspective of the data flowing through the system and the transformation of the data by the identified processes. The data are defined in more detail in a data dictionary using a standard data definition. When defining a system, it is useful to start at a high level with just a few processes and data flow and then further refine the processes into additional processes with their data flows and files. This design refinement continues until, at some point, the function of some particular process is well understood and its internal workings are specified in detail using a verbal description and a short PDL. The design process is complete when all the lowest level processes have been defined by a PDL and all the data have been defined.

The notation used is that the well understood areas in the pdls are defined using all upper case letters. Less well understood areas are defined using lower case.

The bulk of the actual specification resides in sections 5-8 of this document.

5. **LEVEL 0 DATA FLOW AND DESCRIPTIONS:** This section contains the highest-level data-flow diagram for the simulation. It is similar to a system-level diagram. It also contains a verbal description of the high-level-processes. The data flows and files used are defined in Section 8.
6. **LEVEL 1 DATA FLOW AND DESCRIPTIONS:** This section contains the first expansion of all the processes from level 0. Some processes are defined using a verbal description while others can be defined using the PDL. The data flows and files are once again defined in Section 8.
7. **LEVEL 2 DATA FLOW AND DESCRIPTIONS:** This section contains a further expansion of some of the processes from level 1 that require a more precise description. All of the processes are defined using verbal descriptions and PDLs. The data flows are defined in Section 8.
8. **DATA DEFINITIONS:** This section is a data dictionary that defines all data flows and files needed by the level 0, 1, and 2 data-flow diagrams and process specifications. The data definitions are done to a level of precision needed by the rest of the specification. Some contain only a verbal description while other are defined down to the data-element level. In a very real sense, this is an essential part of the specification because it provides a functional grouping for the system data and files.

### DESIGN APPROACH

## REFERENCES

1. R. D. Christman, "DETEC Guidelines for Problem Development Language and FORTRAN," Los Alamos National Laboratory Preliminary Internal Report (October 31, 1984).
2. T. DeMarco, STRUCTURED ANALYSIS AND SYSTEM SPECIFICATION (Yourdon Press., NEW YORK, NY. December 1978).
3. E. Yourdon and L. L. Constantin, STRUTURED DESIGN, 2nd ed.(Yourdon Press., NEW YORK, NY. 1978).
4. W. P. Stevens, USING STRUCTURED DESIGN (John Wiley & Sons, INC., NEW YORK, NY. 1981).
5. Opode Inc., HISTORIAN PLUS, Houston, TX. 1978.

## REFERENCES

#### 4. SCENARIO EXAMPLE

Thorough code walkthroughs are an essential part of the DETEC code-specification process. The purpose of walkthroughs is to verify that the code, as specified, will perform as expected. Walkthroughs are therefore required to exercise the major code elements and functions. It must also be assured that all major types of events, especially all types of potentially troublesome events, occur during the walkthrough.

The DETEC preliminary walkthrough excluded the MANAGER functions. It included most of the EXECUTIVE functions, MOTHER\_NATURE, and all classes of engagement modules (BATTLE MANAGERS, SENSORS, COMMUNICATIONS, WEAPONS, and EVENT\_PHYSICS).

The walkthrough was accomplished by defining a set of initial conditions, which included one or more driving, or "trigger" events or conditions (such as launch boosters), and then following the simulation step by step through the module PDLs. For a discussion of PDLs see Section 3. Simple but typical characteristics were defined for the individual physics and asset simulator modules. The PDLs were written to reflect these characteristics and the requirement for a variety of event types (instantaneous, extended, and conflicted). MOTHER\_NATURE was designed to handle both discrete objects and grid quantities.

The following assets were chosen for a two-sided, offensive/defensive engagement:

- 10 ICBMs with busses (4 RVs per bus)
- 6 sensors (exoatmospheric)
- 1 AWACS platform (associated with terminal defense)
- 1 terminal defense system (Sprint-like)
- 2 space-based defensive weapons
  - 1 particle beam
  - 1 laser
  
- 2 "BOSS" battle managers
- 10 local battle managers
  - 1 on terminal defense
  - 1 on AWACS
  - 6 on sensors
  - 1 on laser
  - 1 on particle beam

The necessary MOTHER\_NATURE functions for these assets are the following:

- booster burn
- exoatmospheric ballistic flight
- endoatmospheric ballistic flight with weather effects
- nuclear effects
  - thermal
  - blast
  - prompt nuclear radiation
  
- charged particles in the earth's magnetic field
- powered endoatmospheric flight with weather effects
- weather.

#### SCENARIO EXAMPLE

#### 4. SCENARIO EXAMPLE

EVENT PHYSICS included the following:

- laser effects on
  - boosters
  - midcourse vehicles
  - RVs
- particle beam on
  - boosters
  - midcourse vehicles
  - RVs
- nuclear effects
  - thermal on RVs
  - blast on AWACS
  - prompt radiation on Boss battle manager
- firing of
  - boosters
  - interceptors
- deployment of RVs
- STATE VECTOR creation and destruction for all weapons.

Each BATTLE\_MANAGER and SENSOR had associated communication functions.

The walkthrough geometry was limited to a single plane with all orbits circular and within that plane. Correct physics was not as important to the walkthrough as representative behavior, so circular orbits of any altitude and orbital velocity were allowed.

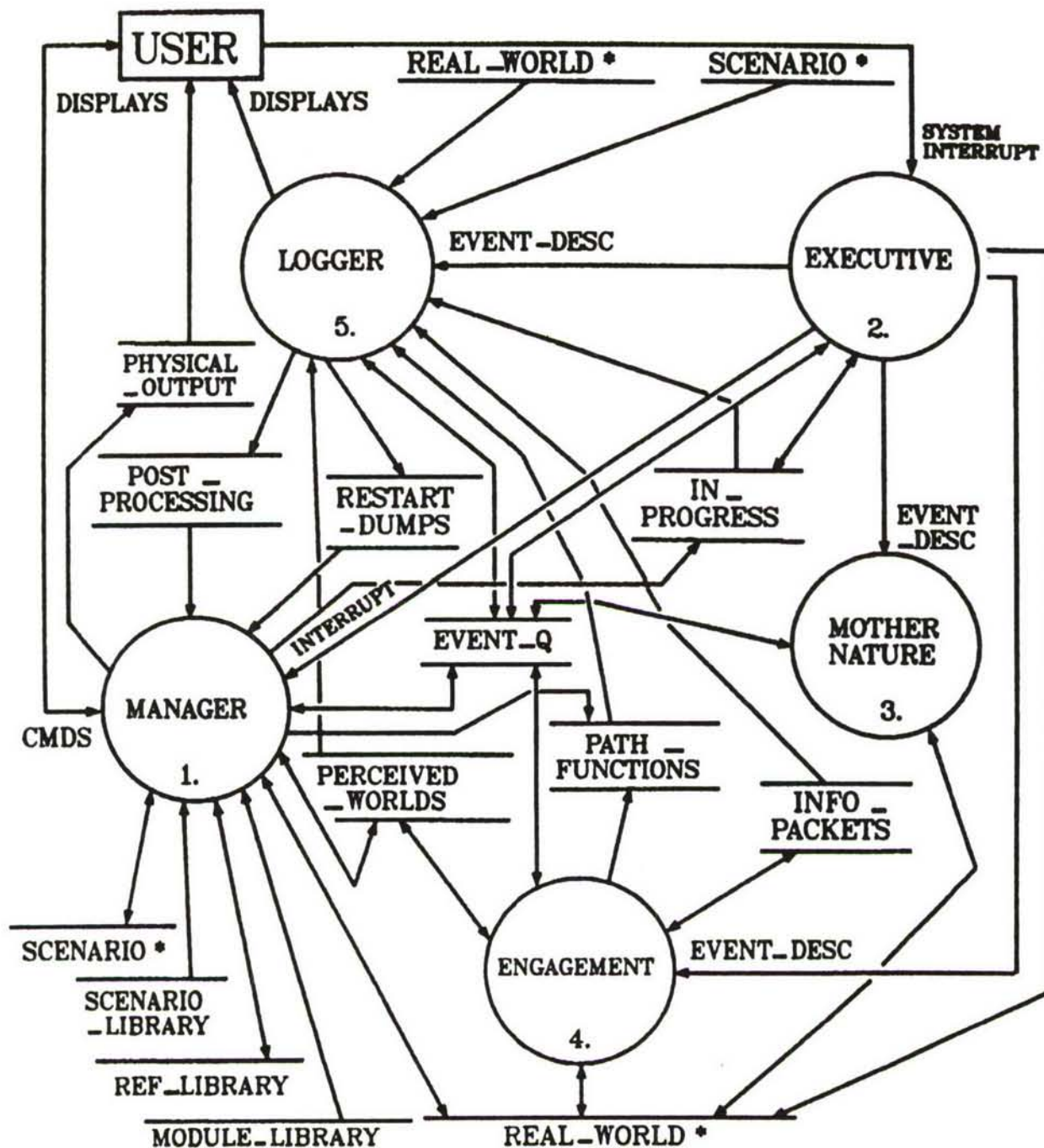
The walkthrough was documented by a summary of each EVENT and copies of the contents of essential files at the end of the EVENT. The files documented were the following: EVENT.Q, IN\_PROGRESS, REAL\_WORLD, PERCEIVED\_WORLDS, and PATH\_FUNCTIONS. The other files shown in the DETEC level 0 data-flow diagram are associated with MANAGER or LOGGER functions, which were not treated in the walkthrough.

#### SCENARIO EXAMPLE

## **5. LEVEL 0 DATA FLOW AND DESCRIPTIONS**

**Level 0**

# DETEC Level 0 Data Flow



## **5. LEVEL 0 DATA FLOW AND DESCRIPTIONS**

### **1.0 MANAGER**

The Manager provides the user with a human-engineered interface to the simulation. The user can set up simulation initial conditions, observe and change execution, and view results after the run.

**EXAMPLE:** Simulate a two-combatant war with one side having offense only and the other having offense and defense.

### **2.0 EXECUTIVE**

The Executive provides overall control of the simulation execution, uses the list of time-ordered events (EVENT\_Q) to determine which functions to run next, and resolves conflicts between simulation assets.

**EXAMPLE:** Start a sensor module that looks at space once per second.

### **3.0 MOTHER NATURE**

Mother Nature does the orderly evolution of currently existing real-world objects and also updates the real-world environment.

**EXAMPLE:** Update the position of all RVs.

### **4.0 ENGAGEMENT**

Engagement simulates the actual assets, decisions, and interactions involved in the battle and also does the birth and death of natural objects.

**EXAMPLE:** An ICBM was intercepted by a laser beam; determine the extent of the damage to the ICBM.

### **5.0 LOGGER**

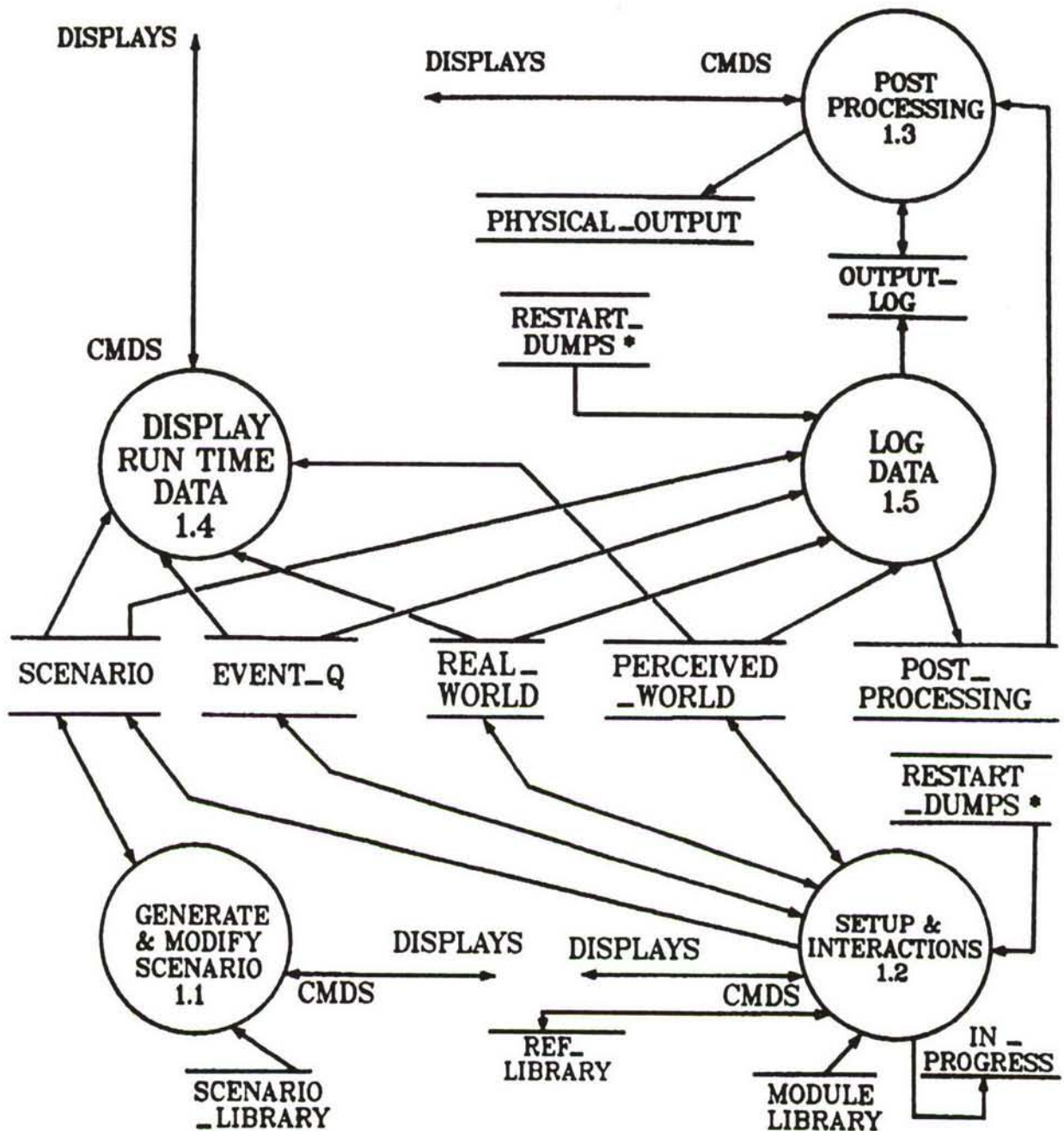
The Logger outputs log messages, restart dumps, and postprocessing data and updates real-time displays.

## **LEVEL 1 DATA FLOW AND DESCRIPTIONS**

**Manager  
Executive  
Mother Nature  
Engagement  
Logger**

# 1. MANAGER

## Level 1 Data Flow



## 1. MANAGER

### 1. MANAGER

#### 1.1 GENERATE AND MODIFY SCENARIO

Based on user instructions, this process creates a new or modifies an existing scenario file. This includes defining the simulation configuration environment, battle-manager instructions, and any planned interventions.

**EXAMPLE:** Define the scenario for war between countries A and B. Side A has two ICBMs and side B has five anti-ICBM weapons. The side-B battle manager uses kill-at-first-chance tactic.

#### 1.2 SETUP AND INTERVENTIONS

This process allows the user to complete the simulation setup before to execution. The user may also modify the course of an executing simulation by stopping the simulation and changing any parameters.

**EXAMPLE:** Select (or change) the simulation fidelity for a particular asset.

#### 1.3 POSTPROCESSING

This process provides detailed tabular and graphical output about the simulation. It also provides post-processed figures of merit. The output is available at the user's terminal or on paper or film hardcopy.

**EXAMPLE:** Provide graphical output on all side-B weapon trajectories.

#### 1.4 DISPLAY RUN TIME DATA

This process allows the user to request the display of various data so that she may observe the progress of the simulated battle.

**EXAMPLE:** Display trajectories off all objects in space.

#### 1.5 LOG DATA

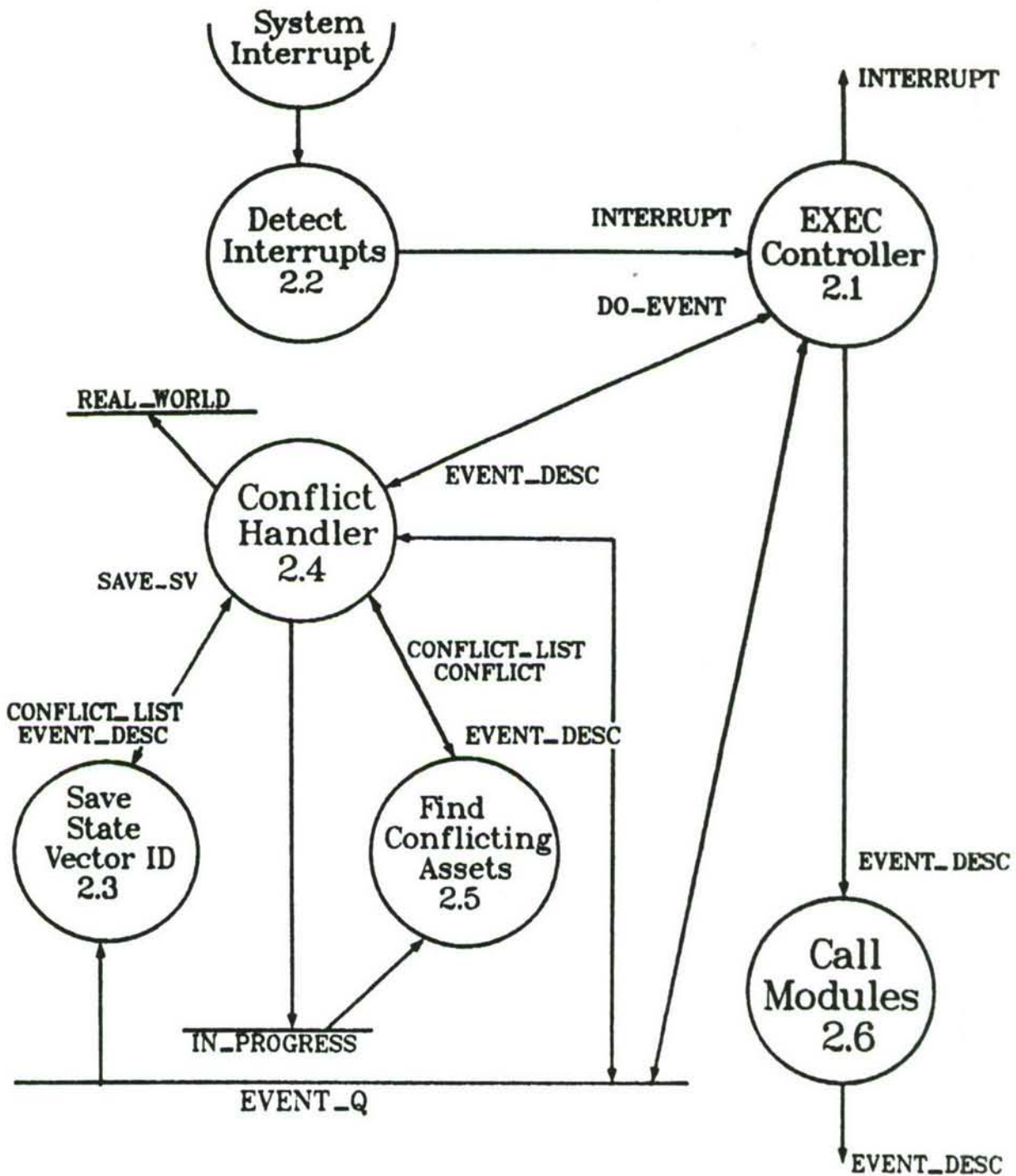
This process allows the user to request the logging of REAL\_WORLD, PERCEIVED\_WORLDS, SCENARIO, EVENT\_Q and other useful information. This information is written to a postprocessing file during the simulation.

## 1. MANAGER

```
1
2
3 -----
4     MGR - SIMULATION CODE MANAGER
5     1.
6     -----
7
8     FUNCTION - THIS MODULE IS THE HIGHEST LEVEL
9     OF CONTROL FOR THE DETEC SIMULATION. IT OBTAINS
10    INPUT FROM THE USER AND PASSES IT TO THE
11    VARIOUS MAJOR USER-INITIATED FUNCTIONS.
12
13    INPUT - NONE
14
15    OUTPUT - NONE
16
17 -----
18    MGR ()
19
20    INITIALIZE FOR THE MANAGER
21
22    DO UNTIL terminated by user
23    GET VALID USER COMMAND (CMDS)
24    CASE (cmd type)
25        WHEN (scenario)
26            GENERATE AND MODIFY SCENARIO (CMDS, DISPLAYS) /*1.1*/
27        WHEN (setup | interventions)
28            SETUP & INTERVENTIONS (CMDS, DISPLAYS) /*1.2*/
29        WHEN (post)
30            POSTPROCESSING (CMDS, DISPLAYS) /*1.3*/
31        WHEN (display)
32            DISPLAY RUN TIME DATA (CMDS, DISPLAYS) /*1.4*/
33        WHEN (log)
34            LOG DATA (CMDS, DISPLAYS) /*1.5*/
35        WHEN (run)
36            /*2. - RUN THE SIMULATION*/
37            CALL EXEC (INTERRUPT)
38        OTHERWISE
39            SETUP ERROR RESPONSE (CMDS, DISPLAYS)
40    END CASE
41    UPDATE DISPLAY (DISPLAYS)
42    END DO
43
44    END /*MGR*/
```

Level 1

## 2. EXEC Level 1 Data Flow



## 2. EXECUTIVE

### 2. EXECUTIVE

The executive provides overall control of the simulation execution. It uses a list of time-ordered events to determine which functions to run next.

The executive resolves any conflicts that may exist in the simulation. A conflict is defined as the case where an asset functions while it is being damaged and/or has two or more overlapping performance requests, where the asset simulation has only minimal first-order provisions for such simulations. The provisions are that the asset simulation modules are able to simulate the asset performance using an arbitrary length time step and may, in addition, be able to use modified state vectors that were saved during the conflict period. For example, a simulation conflict may require that all conflicting assets be simulated up to the current time. Another possibility is that conflicting assets be simulated using discrete time steps during the conflict interval. For the case of the saved state vectors, the saved vectors define how an asset, A, is being damaged while it is trying to damage some other asset, B. The simulation then uses the saved state vectors for A to determine the extent of the damage to asset B.

#### 2.1 EXECUTIVE CONTROLLER

The executive controller checks for interrupts and processes EVENTS from the EVENT\_Q in order by calling other EXEC modules.

#### 2.2 DETECT INTERRUPTS

This process will detect interrupts generated by the user during execution of the simulation.

#### 2.3 SAVE STATE VECTOR IDENTIFICATION

This module reviews the conflicts of a particular asset and determines if the related event must be stepped or if it can be in save state vector mode.

#### 2.4 CONFLICT HANDLER

This module determines if a conflict exists for the event that is to be executed. If a conflict exists, this module determines how to resolve the conflict and puts new events into the EVENT\_Q as required. The conflict resolution is done for instantaneous and extended events as described in the following paragraphs.

Instantaneous events are those physical activities that are so short that they can be simulated as occurring in an infinitesimally short time. This module performs the steps necessary to initiate the execution of the modules needed to simulate an instantaneous event for some asset. If a conflict exists, this module puts events into the EVENT\_Q to call the simulation modules to resolve the conflict. The asset simulations are all run to the current time, and then the asset simulation module for the instantaneous event is executed. Functions that are already in step mode do not have to be run again because they are essentially up to date.

Extended events are used to simulate those physical activities that occur over a substantial time span. This module performs the steps necessary to initiate the execution of modules to start and end the simulation of extended-time activities for an asset. If conflicts exist, this module puts events into the EVENT\_Q to call the simulation modules to resolve the conflicts. Conflicts are resolved by either stepping through the conflict period using discrete time steps or by initiating the saving copies of a state vectors that change during the conflict period. The changed state vectors will be used by conflicted action modules(s) at the end of the extended event to calculate the effect of the conflict.

### Level 1

## **2. EXECUTIVE**

### **2.5 FIND CONFLICTING ASSETS**

This module identifies events that are in conflict with the event currently being set up for execution by the executive. The identification of assets from extended-time events and the assets that they affect are stored in the IN\_PROGRESS file. This process will determine if the asset identification from the current event is already in the file. If the asset is in the file, a conflict exists.

### **2.6 CALL MODULES**

This module selects the appropriate simulation module to execute the current event.

## 2. EXECUTIVE

```
1
2
3 -----
4 EXEC - SIMULATION EXECUTION CONTROL
5     2.1
6 -----
7
8 FUNCTION - THIS MODULE WILL SELECT THE NEXT EVENT FROM
9 THE EVENT QUEUE, DETERMINE IF IT IS INSTANTANEOUS
10 OR EXTENDED EVENT. IT PUTS NEW EVENTS INTO THE EVENT_Q
11 TO RESOLVE CONFLICTS AND THEN EXECUTES THE
12 ACTUAL FUNCTION REQUIRED BY THE EVENT.
13 CONFLICTS ARE RESOLVED EITHER BY STEPPING
14 THROUGH THE CONFLICTING FUNCTIONS OR BY
15 SAVING CHANGED STATE VECTORS AND INTEGRATING OVER
16 THE CHANGES AT THE END OF THE CONFLICT.
17
18 INPUT - NONE
19
20 OUTPUT
21     INTERRUPT - THE TYPE OF INTERRUPT WHICH TERMINATED
22                 THE EXECUTIVE
23
24 -----
25 EXEC CONTROLLER (INTERRUPT)
26
27 DO UNTIL INTERRUPT = COMPLETE ! INTERVENE
28     /*2.2 - INTERRUPT PROCESSING*/
29     CALL INTR (INTERRUPT)
30     IF INTERRUPT = NO_INTR, THEN
31         IF EVENT_Q IS EMPTY
32             INTERRUPT = COMPLETE
33     ELSE
34         GET NEXT EVENT FROM EVENT_Q (EVENT_DESC)
35         ADVANCE TIME TO TIME FROM EVENT_DESC
36
37         IF DO_NOW (from event) = TRUE
38             /*2.6 - EXECUTE EVENT*/
39             CALL CMODULE (EVENT_DESC)
40             REMOVE EVENT FROM EVENT_Q (EVENT_DESC)
41         ELSE
42             /*2.4 - RESOLVE CONFLICTS*/
43             CALL CONHAN (EVENT_DESC, DO_EVENT)
44             IF DO_EVENT = TRUE
45                 /*2.6 - EXECUTE EVENT*/
46                 CALL CMODULE (EVENT_DESC)
47             END IF
```

Level 1

## 2. EXECUTIVE

```
48         REMOVE EVENT FROM EVENT.Q (EVENT_DESC)
49         END IF
50     END IF
51 END IF
52 END DO
53 END /*EXEC*/
54
55
56 -----
57     INTR - INTERRUPT PROCESSING
58         2.2
59     -----
60
61     FUNCTION - THIS MODULE DETECTS INTERRUPTS FROM THE CODE
62     USER AND RETURNS THE TYPE OF INTERRUPT
63     TO THE CALLING MODULE
64
65     INPUT - NONE
66
67     OUTPUT
68     INTERRUPT - THE TYPE OF INTERRUPT RECEIVED
69
70     -----
71     INTR (INTERRUPT)
72
73     IF an interrupt occurred, THEN
74     INTERRUPT = INTERVENE
75 ELSE
76     INTERRUPT = NO_INTR
77
78     END /*INTR*/
79
80
81 -----
82     SSVID - SAVE STATE VECTOR ID
83         2.3
84     -----
85
86     FUNCTION - THIS MODULE DETERMINES IF THE SPECIFIED EVENT
87     IS A SAVE STATE VECTOR OR STEP TYPE
88
89     INPUT
90     EVENT_DESC - POINTER FOR THE EVENT IN QUESTION
91     CONFLICT_LIST - LIST OF EVENTS THAT ARE IN CONFLICT
92
93     OUTPUT
94     SAVE_SV = TRUE - EVENT IS SAVE STATE VECTOR
95             = FALSE - EVENT IS STEP
96
```

## 2. EXECUTIVE

```
97 -----
98 SSVID (EVENT_DESC, CONFLICT_LIST, SAVE_SV)
99
100 if conflict_list is empty, then
101     error
102 else
103     do for each event in conflict_list
104         if asset/event is a save_sv type
105             if event is of the form A to B
106                 A must be stepped
107                 save_sv = false
108             else
109                 A can be save_sv type
110                 save_sv = true
111             end if
112         else
113             A must be stepped
114             save_sv = false
115         end if
116     end do
117 end if
118
119 END /*SSVID*/
120 -----
121 CONHAN - CONFLICT HANDLER
122     2.4
123 -----
124
125 FUNCTION - THIS MODULE DETERMINES IF THE ASSETS INVOLVED
126 IN THE EVENT ARE IN CONFLICT WITH ANY EVENTS. IF A
127 CONFLICT EXISTS, APPROPRIATE EVENTS ARE PUT INTO THE EVENT.Q
128 TO RESOLVE THE CONFLICT.
129
130 INPUT
131     EVENT_DESC - THE EVENT BEING PROCESSED
132
133 OUTPUT
134     DO_EVENT = TRUE - THE DRIVING EVENT CAN BE EXECUTED
135               = FALSE - THE EVENT CANNOT BE EXECUTED
136
137 -----
138 CONHAN (EVENT_DESC, DO_EVENT)
139
140     /*2.5 - GET LIST OF CONFLICTING EVENTS*/
141     CALL CONFLI (EVENT_DESC, CONFLICT_LIST, CONFLICT)
142     IF CONFLICT = TRUE, THEN
143         CASE (MODE)
144             WHEN (INSTANT)
145                 do_event = true
```

Level 1

## 2. EXECUTIVE

```
146 do for each event in conflict_list
147   if event is save state vector /*2.2*/
148     find state vector in real_world (event_desc,
149                                     state_vector)
150     set save_sv = true in state vector
151   else
152     put event into event_q to do unconflicted part
153     of newly conflicted event at current time
154     with mode = partial and do_now = true
155     do_event = false
156   end if
157 end do
158 if any conflicting event is stepped, then
159   copy instantaneous event following the event(s)
160   just added to event_q with do_now = true
161   /*the original event is left in place*/
162 end if
163 WHEN (BEGINNING)
164 do for each event in conflict_list
165   if event is save state vector /*2.3*/
166     find state vector in real_world (event_desc,
167                                     state_vector)
168     set save_sv = true in state_vector
169   else
170     if conflicted event starts at same time as
171     beginning event
172       put event into event_q for conflicted event
173       to begin step with mode = first_step
174       and do_now = true
175     else
176       put event into event_q for conflicted
177       event to do unconflicted part and then step
178       with mode = step and do_now = true
179     end if
180   end do
181   if any conflicting event is stepped, then
182     copy beginning event following the event(s)
183     just added to the event_q and set
184     mode = first_step and do_now = true
185     /*the original event is left in place*/
186   end if
187   add sv.id(s) to in-progress
188   DO_EVENT = FALSE
189 WHEN (ENDING)
190 do for each event in conflict_list
191   set flag at ending event of newly unconflicted event
192   to do last time interval (final_event_ptr)
193   if newly unconflicted events are stepped &
194   they no longer need to step, then
```

## 2. EXECUTIVE

```
195         find step event in event_q and remove
196         end if
197         end do
198         remove sv_id(s) from in-progress
199         DO_EVENT = TRUE
200     OTHERWISE
201         DO_EVENT = TRUE
202     END CASE
203 ELSE
204     CASE (MODE)
205         WHEN (INSTANT)
206             DO_EVENT = TRUE
207         WHEN (BEGINNING)
208             add sv_id to in-progress
209             DO_EVENT = FALSE
210         WHEN (ENDING)
211             remove sv_id from in-progress
212             DO_EVENT = TRUE
213         OTHERWISE
214             DO_EVENT = TRUE
215     END CASE
216 END IF
217
218 END /*CONHAN*/
```

```
221 -----
222 CONFLT - FIND CONFLICTING EVENT
223     2.5
224 -----
225
226 FUNCTION - THIS MODULE SCANS THE LIST OF EXTENDED EVENTS
227           THAT ARE BEING SIMULATED AND DETERMINES IF ANY OF THEM
228           ARE IN CONFLICT WITH THE NEW EVENT.
229           IT RETURNS A LIST OF EVENTS THAT ARE IN CONFLICT WITH
230           THE NEW EVENT.  THE NEW EVENT IS IN THIS LIST.
231
232 INPUT
233     EVENT_DESC - THE EVENT DRIVING THIS ACTIVITY
234
235 OUTPUT
236     CONFLICT_LIST - A LIST OF ALL THE EVENTS AND ASSETS THAT
237                   ARE IN CONFLICT WITH THE NEW EVENT
238     CONFLICT = TRUE - A CONFLICT HAS BEEN FOUND
239               = FALSE - NO CONFLICT HAS BEEN FOUND
240
241 -----
242 CONFLT (EVENT_DESC, CONFLICT_LIST, CONFLICT)
243
```

Level 1

## 2. EXECUTIVE

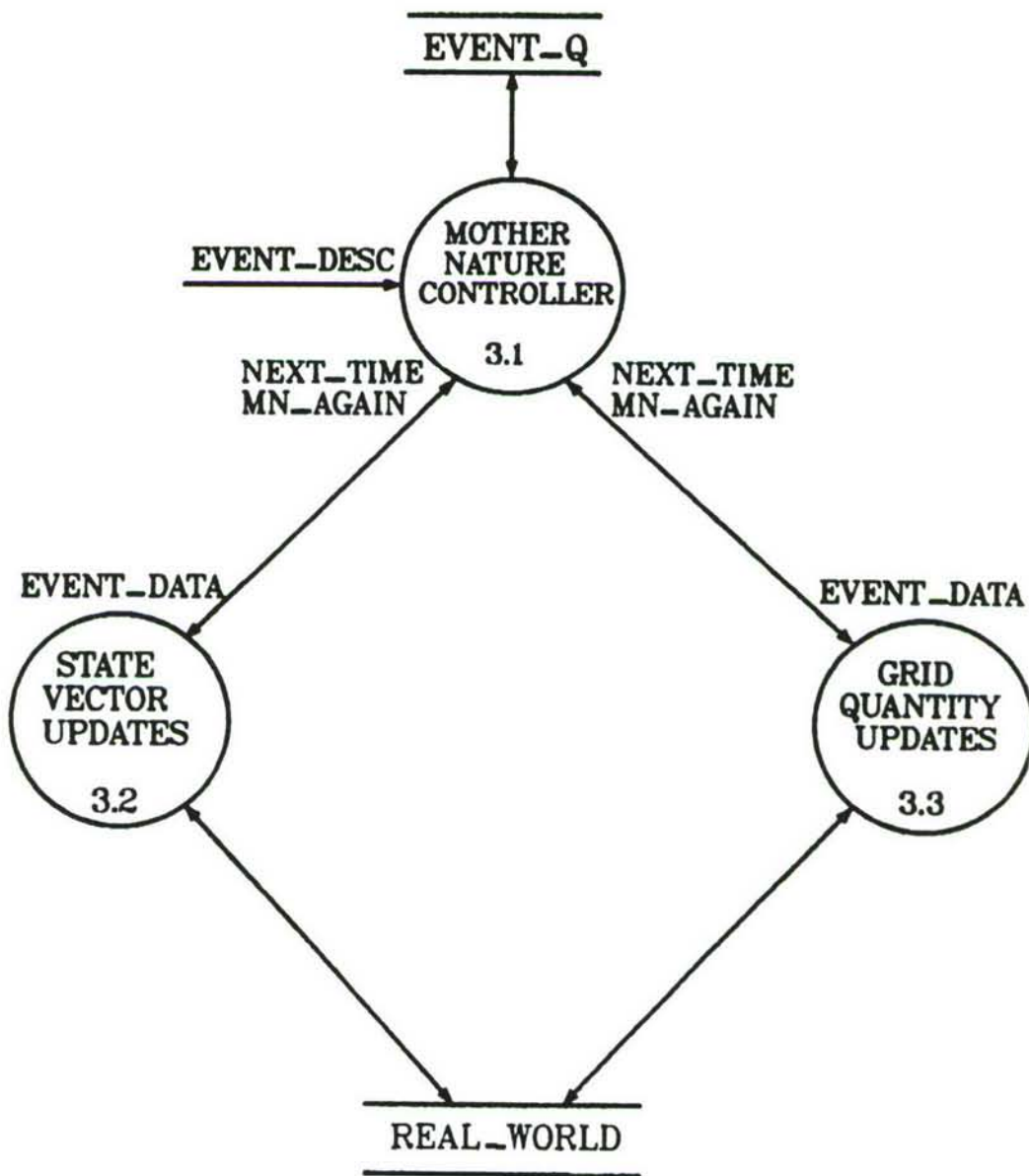
```
244     CONFLICT = FALSE
245
246     do i = 1 to number of entries in in_progress
247         if any sv_id from new event = sv_id(i) from in_progress
248             conflict = true
249             if space is available in conflict_list
250                 copy in_progress entry i to next location in conflict_list
251             else
252                 error
253             end if
254         end do
255
256     END /*CONFLT*/
257
258 -----
259     CMODULE - CALL FUNCTIONS FOR EXEC
260         2.6
261     -----
262
263     FUNCTION - THIS MODULE WILL CALL THE SPECIFIED MAJOR MODULE
264                AS DEFINED BY THE INPUT PARAMETER.
265
266     INPUT
267         EVENT_DESC - STRUCTURE WHICH POINTS TO EVENT_DATA
268                    FOR THE MODULE BEING CALLED
269
270     OUTPUT - NONE
271
272     -----
273     CMODULE (EVENT_DESC)
274
275     CASE (DESTINATION)
276         WHEN (mn id)
277             /*3. - ORDERLY EVOLUTION OF REAL WORLD OBJECTS*/
278             MOTHER NATURE (EVENT_DESC)
279         WHEN (bm id)
280             /*4.1 - DO DECISION MAKING FOR ASSETS*/
281             BATTLE MANAGERS (EVENT_DESC)
282         WHEN (sn id)
283             /*4.2 - CALCULATE SENSOR FUNCTIONING*/
284             SENSORS (EVENT_DESC)
285         WHEN (cm id)
286             /*4.3 - CALCULATE COMMUNICATION BETWEEN ASSETS*/
287             COMMUNICATIONS (EVENT_DESC)
288         WHEN (wp id)
289             /*4.4 - CALCULATE WEAPON FUNCTIONING*/
290             WEAPONS (EVENT_DESC)
291         WHEN (ep id)
292             /*4.5 - INTERACTIONS BETWEEN ASSETS*/
```

## 2. EXECUTIVE

```
293         EVENT PHYSICS (EVENT_DESC)
294     WHEN (lg id)
295         /*5. - DUMPS, LOGS AND DISPLAYS*/
296         LOGGER (EVENT_DESC)
297     OTHERWISE
298         ERROR
299     END CASE
300 END /*CMODULE*/
```

Level 1

### 3. MOTHER NATURE Level 1 Data Flow



## **3. MOTHER NATURE**

### **3. MOTHER NATURE**

#### **3.1 MOTHER NATURE CONTROLLER**

The Mother Nature Controller performs three functions:

- 1) It decodes the `EVENT_DATA` to determine which updates to perform.
- 2) It performs the control functions necessary to accomplish the updates.
- 3) It constructs and emplaces `EVENTs` in the `EVENT_Q` for periodic updates.

#### **3.2 STATE VECTOR UPDATES**

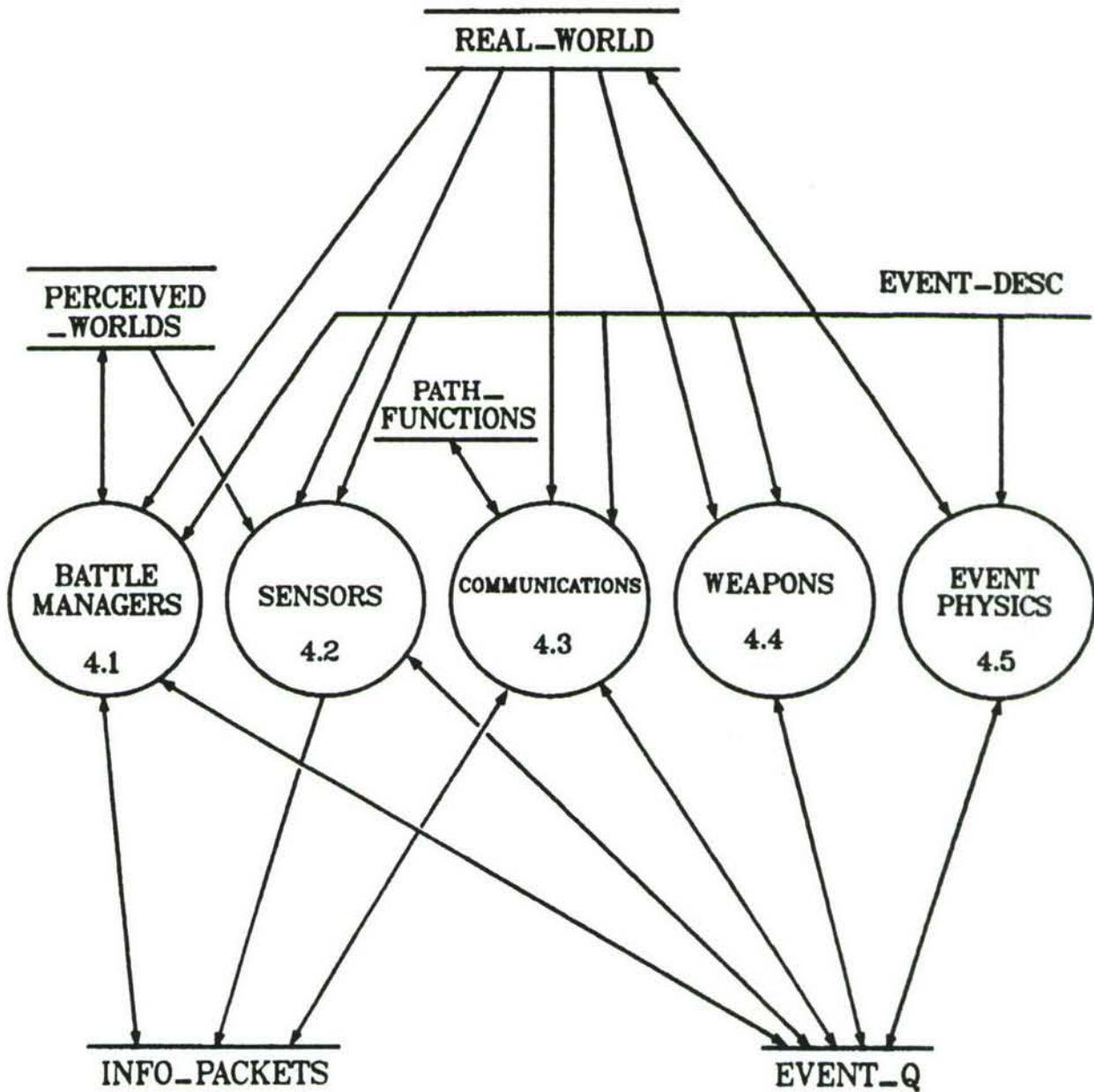
This is a set of modules, each of which updates a class of state vectors. The updates may include position, velocity, orientation and other geometrical type variables as well as conditions such as temperature and fuel remaining.

#### **3.3 GRID QUANTITY UPDATES**

This is a set of modules, each of which updates a grid variable, such as electromagnetic environment or the nonlocal weather. The local weather (storms) is treated on the basis of individual entities described by state vectors and updated by the State Vector Updates modules.

# 4. ENGAGEMENT

## Level 1 Data Flow



## **4. ENGAGEMENT**

### **4. ENGAGEMENT**

#### **4.1 BATTLE MANAGERS**

A Battle Manager uses information sent to it by sensors, weapons, and other battle managers, along with self-generated information and information loaded during the initialization process to make decisions and send messages based on those decisions. All of the above information is stored in a battle manager's own perceived world - no other information is available for the decision process.

#### **4.2 SENSORS**

A Sensor functions as a filter on real- and perceived-world data. It can function either periodically or under explicit orders and send messages about what it has sensed to battle managers.

#### **4.3 COMMUNICATIONS**

This process simulates the communication between various assets used by the simulation. In some cases routing decisions may be made. Also, environmental effects are accounted for.

#### **4.4 WEAPONS**

A weapons module calculates the actual function of a weapon, e.g., nuclear yield for a bomb, beam duration and intensity for a laser, etc.

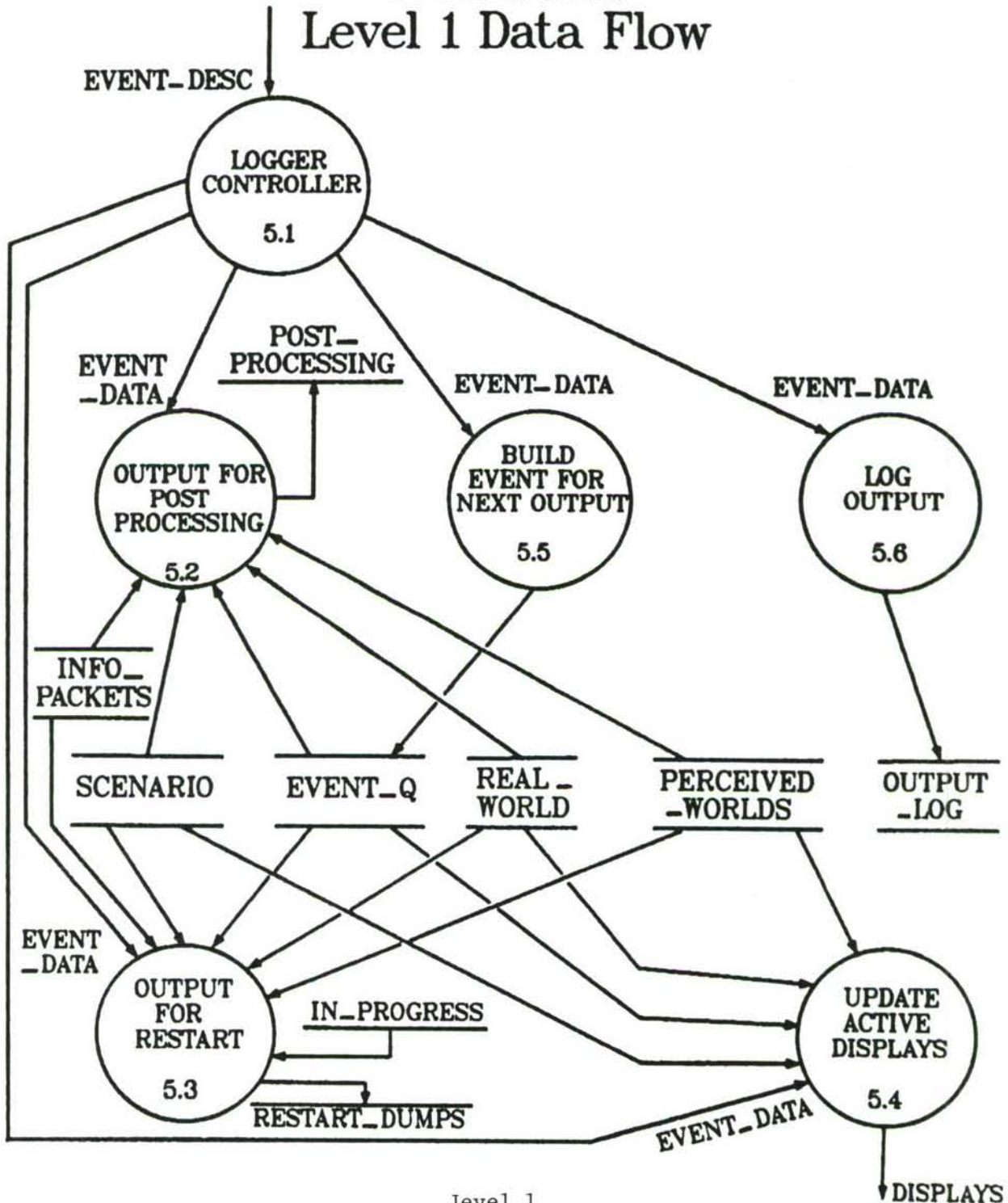
#### **4.5 EVENT PHYSICS**

The Event Physics module(s) calculates the interaction between a directed attack and a specific target (e.g., laser on a RV), calculates the effects of volume stresses on nearby assets (e.g., blast on a radar antenna), and performs the creation and destruction of state vectors (e.g., solar flares, spawn RV, volcanic eruption, etc.).

**Level 1**

# 5. LOGGER

## Level 1 Data Flow



Level 1

## **5. LOGGER**

### **5. LOGGER**

#### **5.1 LOGGER CONTROLLER**

This module decides which modules to run by using information from `EVENT_DATA`.

#### **5.2 OUTPUT FOR POSTPROCESSING**

This module dumps interesting information from the `SCENARIO`, `EVENT_Q`, `REAL_WORLD`, and `PERCEIVED_WORLDS` files. This information can then be reduced and analyzed after the simulation run by using the postprocessing functions.

#### **5.3 OUTPUT FOR RESTART**

This module makes an exact copy of all necessary files so the simulation can be restarted from this exact point at some later time if desired.

#### **5.4 UPDATE ACTIVE DISPLAY(S)**

This module will update any active displays that have been selected by the user. This allows the user to observe the progress of the simulation.

#### **5.5 BUILD EVENT FOR NEXT OUTPUT**

This module decides how long until the next output is required and generates the `EVENT` that will call `LOGGER` at the proper time. The `EVENT` is then added to the `EVENT_Q`.

#### **5.6 LOG OUTPUT**

This module logs what the `LOGGER` function has done during this particular call.

## 5. LOGGER

```
1
2
3 -----
4     LOGGER - DO RUN TIME OUTPUT
5         5.1
6     -----
7
8     FUNCTION - THIS MODULE CALLS THE VARIOUS OUTPUT FUNCTIONS
9               THAT CAN BE USED WHILE THE SIMULATOR IS RUNNING.
10            IT ALSO DECIDES WHEN THE NEXT OUTPUT IS NEEDED AND
11            GENERATES AN EVENT SO THAT IT WILL BE CALLED AT THE
12            APPROPRIATE TIME.
13
14     INPUT
15         EVENT_DESC - THE EVENT THAT TRIGGERED THE CALL
16
17     OUTPUT - NONE
18
19     -----
20     LOGGER (EVENT_DESC)
21
22     GET EVENT FROM EVENT_Q (EVENT_DESC, EVENT_DATA)
23
24     CASE ( )
25         WHEN (postprocessing)
26             OUTPUT FOR POST PROCESSING (EVENT_DATA) /*5.2*/
27         WHEN (restart)
28             OUTPUT FOR RESTART (EVENT_DATA) /*5.3*/
29         WHEN (display)
30             UPDATE ACTIVE DISPLAYS (EVENT_DATA) /*5.4*/
31         OTHERWISE
32             ERROR
33     END CASE
34
35     BUILD EVENT FOR NEXT OUTPUT AND ADD
36         TO EVENT_Q (EVENT_DESC) /*5.5*/
37     LOG OUTPUT (EVENT_DESC) /*5.6*/
38
39     END /*LOGGER*/
```

Level 1

## **7. LEVEL 2 & 3 DATA FLOW AND DESCRIPTIONS**

**Mother Nature**

**Battle Manager**

**Sensors**

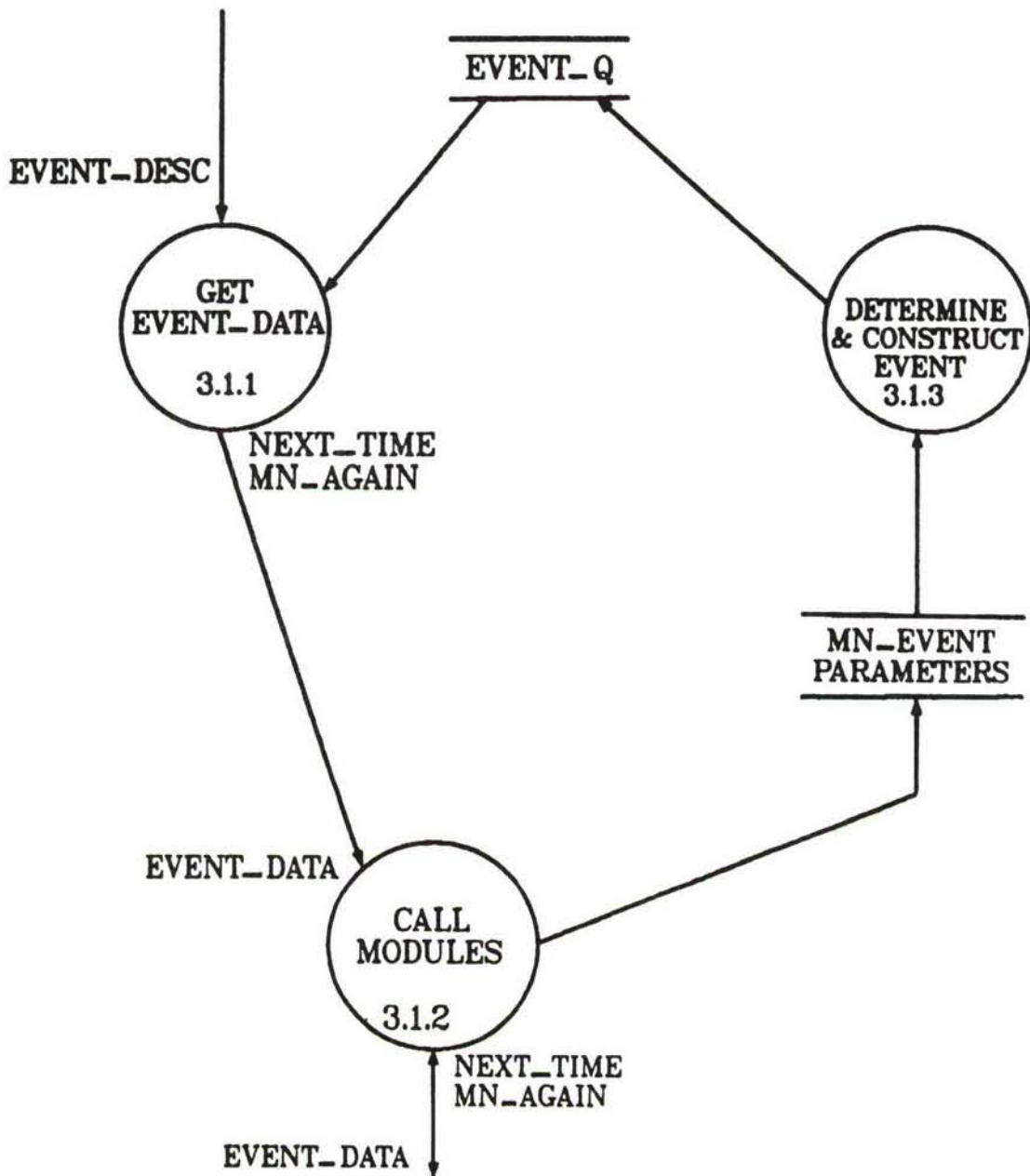
**Communications**

**Weapons**

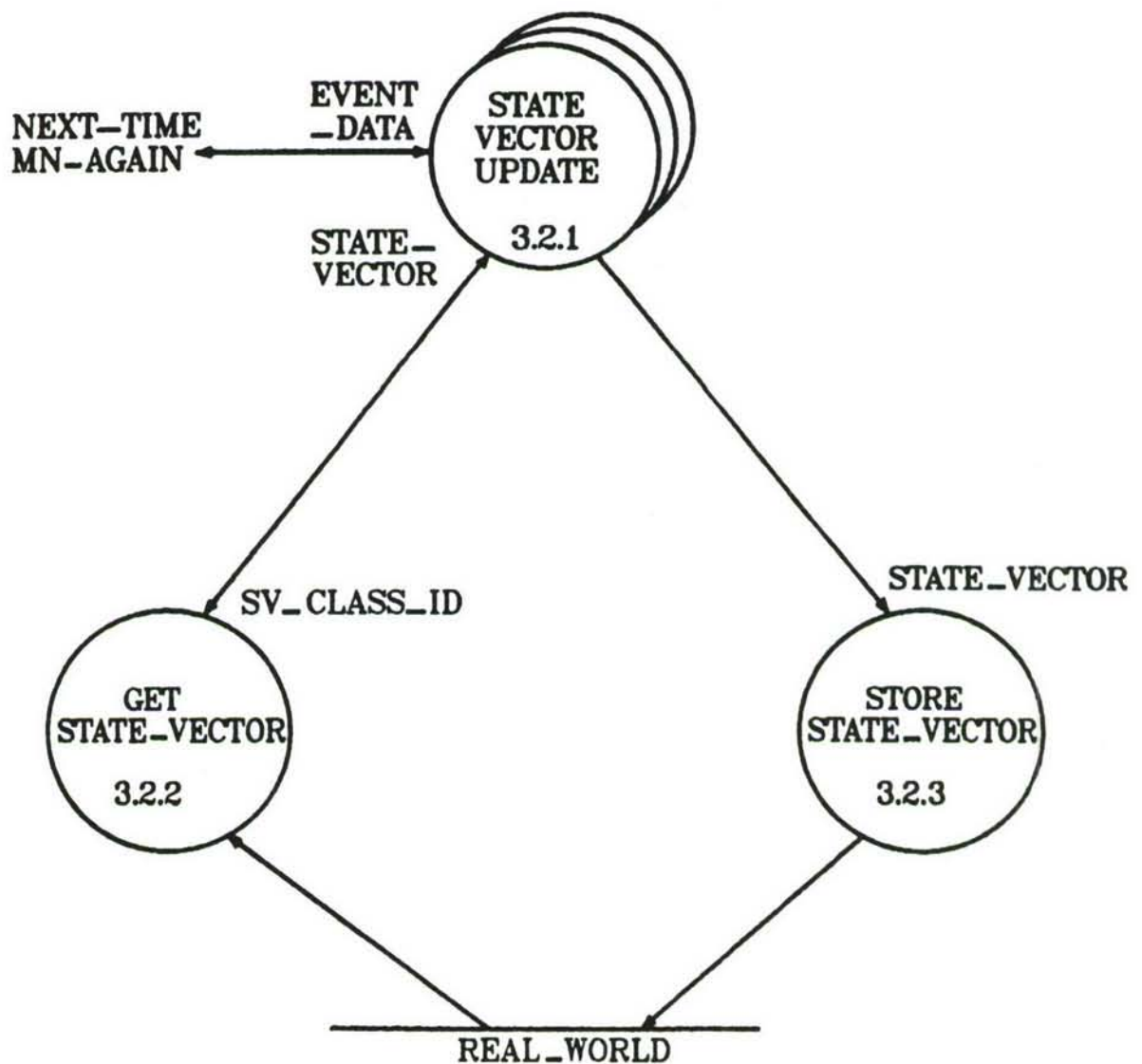
**Event Physics**

# 3.1 MOTHER NATURE CONTROLLER

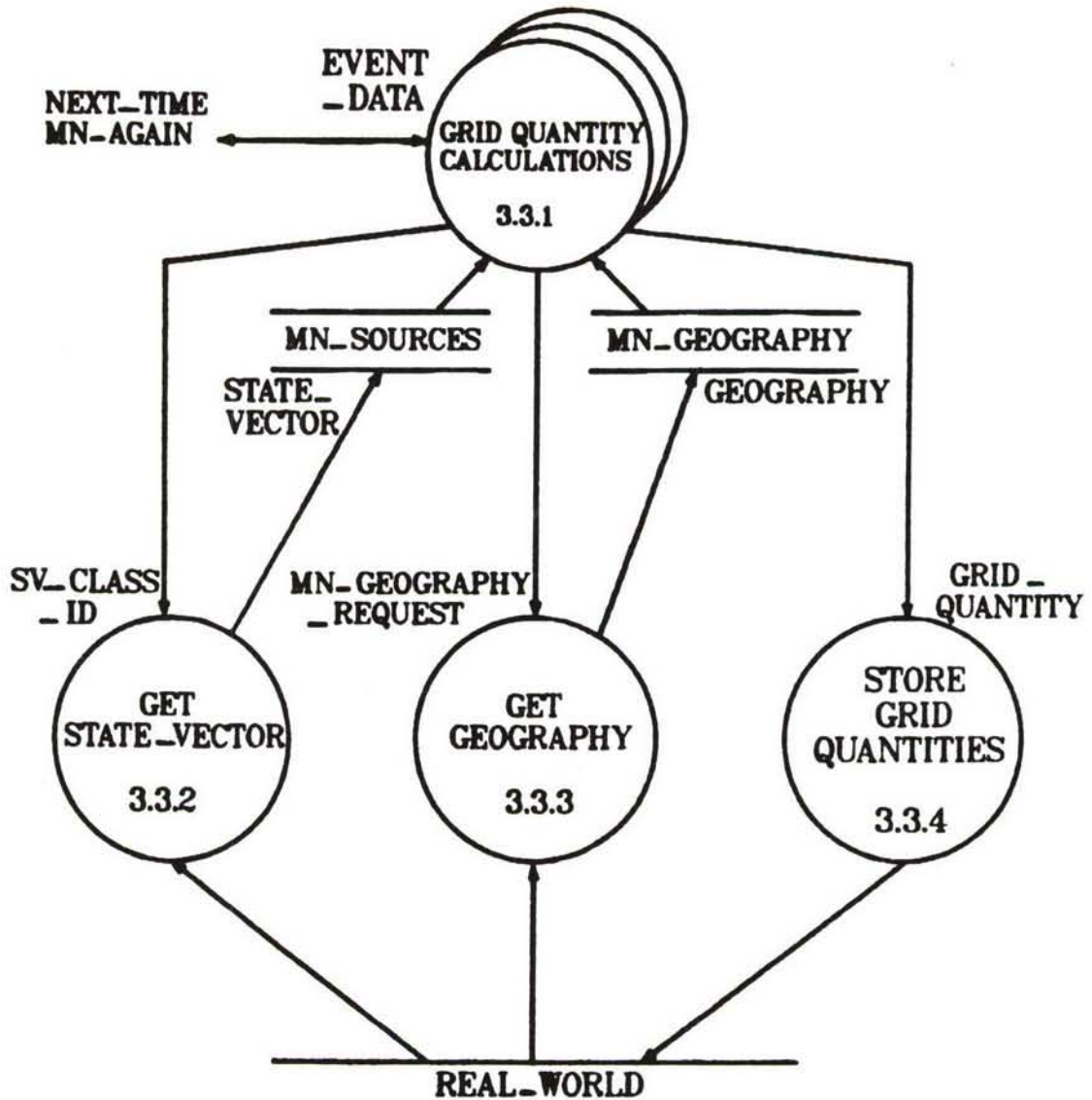
## Level 2 Data Flow



## 3.2 MOTHER NATURE STATE VECTOR UPDATES Level 2 Data Flow



# 3.3 MOTHER NATURE GRID QUANTITY CALCULATIONS Level 2 Data Flow



Level 2 & 3

## **7. LEVEL 2 & 3 MOTHER NATURE**

### **3.1 MOTHER NATURE**

#### **3.1.1 GET EVENT DATA**

This is a standard module that uses the `EVENT_DESC` to fetch the `EVENT_DATA`.

#### **3.1.2 CALL MODULES**

This module uses the `EVENT_DATA` to identify which classes of `STATE_VECTORS` and `GRID_QUANTITY(ies)` require updating or calculating, and calls, in turn, the correct modules to do the calculations. It may receive information from the modules on when they should be recalculated, and it writes this information in the `MN_EVENT_PARAMETERS` file.

#### **3.1.3 DETERMINE AND CONSTRUCT EVENT**

This module analyzes the data in the `MN_EVENT_PARAMETERS` file to determine when and how Mother Nature should be called again. It then constructs the necessary `EVENTS` and puts them in the `EVENT_Q`.

#### **3.2.1 CLASS n STATE VECTOR UPDATE MODULE**

This is a set of update modules, one for each class of `STATE_VECTORS`. Each uses the existing parametric description within the `STATE_VECTOR` to calculate new values of a set of variables for the current time.

#### **3.2.2 GET STATE VECTORS**

This module gets from the `REAL_WORLD` all `STATE_VECTORS` with the `SV_CLASS_ID`.

#### **3.2.3 STORE STATE VECTORS**

This module stores the updated `STATE_VECTORS` in the appropriate places in the `REAL_WORLD`.

#### **3.3.1 GRID QUANTITY CALCULATIONS**

This represents a set of modules, each of which does the calculations of a `GRID_QUANTITY` (e.g. electromagnetic environment or nonlocal weather) for a specific time, and for all grid locations. It uses the source `STATE_VECTORS` and `GEOGRAPHY` put in the `MN_SOURCE` and `MN_GEOGRAPHY` files by Get State Vectors and Store State Vectors and ships the calculated quantities to Store Grid Quantities for storage.

#### **3.3.2 GET STATE VECTORS**

This module gets from the `REAL_WORLD` all `STATE_VECTORS` with a specific `SV_CLASS_ID` and writes them to the `MN_SOURCES` and `MN_GEOGRAPHY` files.

#### **3.3.3 GEOGRAPHY**

This module gets from the `REAL_WORLD` whatever `GEOGRAPHY` information has been requested by Grid Quantity Calculations using the parameter `MN_GEOGRAPHY_REQUEST` and writes it in the `MN_SOURCES` and `MN_GEOGRAPHY` files.

## **7. LEVEL 2 & 3 MOTHER NATURE**

### **3.3.4 STORE GRID QUANTITIES**

This module accepts GRID\_QUANTITY and writes it to the appropriate place in the REAL\_WORLD file.

## 7. LEVEL 2 & 3 MOTHER NATURE

```
1
2
3 -----
4     MN - MOTHER NATURE
5       3.1
6     -----
7
8     FUNCTION - THIS MODULE GETS THE EVENT FOR THE MOTHER NATURE
9               MODULES, CALLS THE VARIOUS MODULES AND CONSTRUCTS THE
10              EVENTS FOR THOSE MODULES REQUIRING CALLS TO DO PERIODIC
11              OR ADDITIONAL MOTHER NATURE CALCULATIONS.
12
13     INPUT
14       EVENT_DESC - THE EVENT THAT TRIGGERED THE CALL
15
16     OUTPUT
17       NONE
18
19     -----
20     MN (EVENT_DESC)
21
22     GET EVENT FROM EVENT_Q (EVENT_DESC, EVENT_DATA) /*3.1.1*/
23     DETERMINE FROM EVENT_DATA THE MODULES TO BE CALLED
24     DO FOR EACH MODULE TO BE CALLED
25       CALL module (TIME, NEXT_TIME, MN_AGAIN) /*3.1.2*/
26       IF MN_AGAIN = TRUE
27         WRITE NEXT_TIME & SV_ID TO MN_EVENT_PARAMETERS
28       END IF
29     END DO
30     ANALYZE ENTRIES IN MN_EVENT_PARAMETERS
31     DETERMINE AND CONSTRUCT EVENTS /*3.1.3*/
32     END DO
33     END /*MN*/
34
35
36 -----
37     MNSVUP - MOTHER NATURE STATE VECTOR UPDATE
38       3.2
39     -----
40
41     FUNCTION - THIS IS A SERIES OF MODULES EACH OF WHICH UPDATES
42               A CLASS OF STATE VECTORS SUCH AS POSITION, VELOCITY, AND
43               FUEL REMAINING.
44
45     INPUT
46       EVENT_DATA - THE EVENT WHICH TRIGGERED THE MODULE CALL
47
```

## 7. LEVEL 2 & 3 MOTHER NATURE

```
48     OUTPUT
49     NEXT.TIME - THE TIME WHEN THIS MODULE IS TO BE CALLED AGAIN
50     MN.AGAIN = TRUE - CALL THIS MODULE MN.AGAIN
51     = FALSE - DO NOT CALL THIS MODULE MN.AGAIN
52
53     -----
54     MNSVUP (EVENT.DATA, NEXT.TIME, MN.AGAIN)
55
56     GET ALL STATE.VECTORS FROM REAL.WORLD FOR SV.CLASS.ID /*3.2.2*/
57     DO FOR EACH RETRIEVED STATE.VECTOR
58     UPDATE THE STATE.VECTOR (EVENT.DATA, STATE.VECTOR) /*3.2.1*/
59     END DO
60     STORE ALL THE MODIFIED STATE.VECTORS INTO REAL.WORLD /*3.2.3*/
61     if module needs to run again
62     compute NEXT.TIME
63     MN.AGAIN = TRUE
64     else
65     MN.AGAIN = FALSE
66     end if
67     END /*MNSVUP*/
68
69     -----
70
71     MNGRID - MOTHER NATURE GRID QUANTITY CALCULATIONS
72     3.3
73     -----
74
75     FUNCTION - THIS IS A SET OF MODULES EACH OF WHICH
76     UPDATES A GRID VARIABLE SUCH AS ELECTROMAGNET
77     ENVIRONMENT OR THE NON LOCAL WEATHER.
78
79     INPUT
80     EVENT.DATA - THE EVENT WHICH TRIGGERED THE MODULE CALL
81
82     OUTPUT
83     NEXT.TIME - THE TIME WHEN THIS MODULE IS TO BE CALLED AGAIN
84     MN.AGAIN = TRUE - CALL THIS MODULE MN.AGAIN
85     = FALSE - DO NOT CALL THIS MODULE MN.AGAIN
86
87     -----
88     MNGRID (EVENT.DATA, NEXT.TIME, MN.AGAIN)
89
90     GET ALL STATE VECTORS FROM REAL.WORLD FOR SV.CLASS.ID /*3.3.2*/
91     WRITE RETRIEVED STATE.VECTORS TO MN.SOURCES
92     GET GEOGRAPHY FOR MN.GEOGRAPHY.REQUEST FROM REAL.WORLD /*3.3.3*/
93     WRITE RETRIEVED GEOGRAPHY TO MN.GEOGRAPHY
94     DO FOR EACH.GRID QUANTITY
95     CALL module TO CALCULATE VARIABLE GRID QUANTITY /*3.3.1*/
96     STORE GRID.QUANTITY INTO REAL.WORLD /*3.3.4*/
```

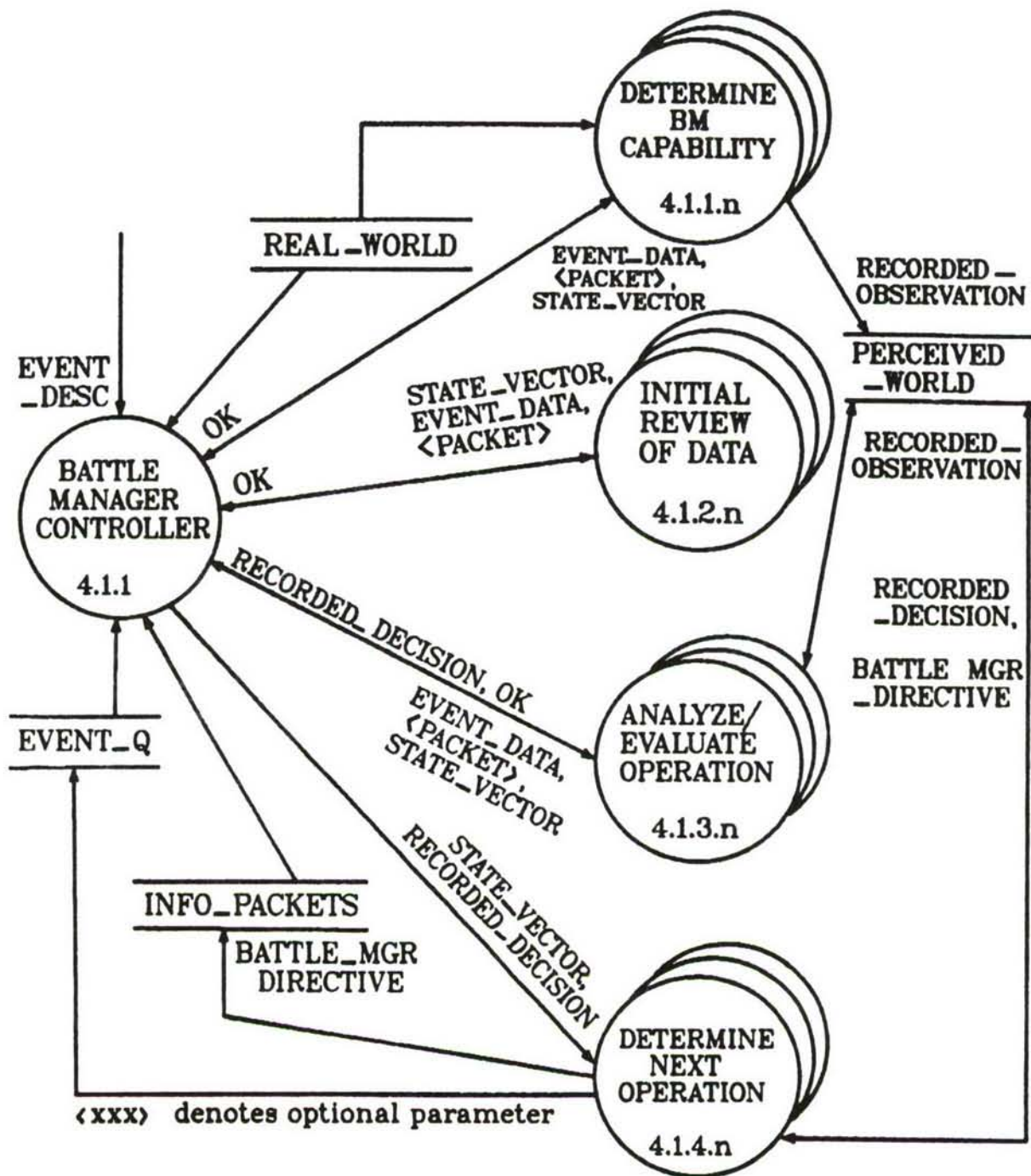
Level 2 & 3

## 7. LEVEL 2 & 3 MOTHER NATURE

```
97      END DO
98      if module needs to be run again
99          compute NEXT_TIME
100         MN_AGAIN = TRUE
101     else
102         MN_AGAIN = FALSE
103     end if
104
105     END /*MNGRID*/
```

# 4.1 BATTLE MANAGER

## Level 2 Data Flow



## **LEVEL 2 & 3 BATTLE MANAGERS**

### **4.1 BATTLE MANAGER**

#### **4.1.1 BATTLE MANAGER CONTROLLER**

There is a battle manager for every asset that is required to make any decisions and/or report results. The controller process gets data for the battle manager processes and determines which processes to call.

##### **4.1.1.n DETERMINE BATTLE MANAGER CAPABILITY**

This process determines what is required of the battle manager in the execution of this event. Based on the battle manager's state vector and perceived world information, this process determines if the battle manager is capable of performing its function.

##### **4.1.2.n INITIAL REVIEW OF PACKET**

This process gets all data that may be received at this time and may perform a cursory first review of the data, including consistency and applicability checks.

##### **4.1.3.n ANALYZE/EVALUATE OPERATIONS**

This process examines the battle manager's perceived world and other data and performs analyses and evaluations of the current and/or projected situation. This can be triggered by the reception of unexpected information, the reception of information for an existing operation, an interruption for an existing operation, or a self-triggered evaluation. At this time the battle manager may record its observations in its perceived world.

##### **4.1.4.n DETERMINE NEXT OPERATION**

Based on the above analysis, the battle manager may decide to start a new operation, continue an existing operation, or terminate an operation. This process generates the EVENTS and PACKETS required to do this and may record the battle manager's decisions in its perceived world.

## LEVEL 2 & 3 BATTLE MANAGERS

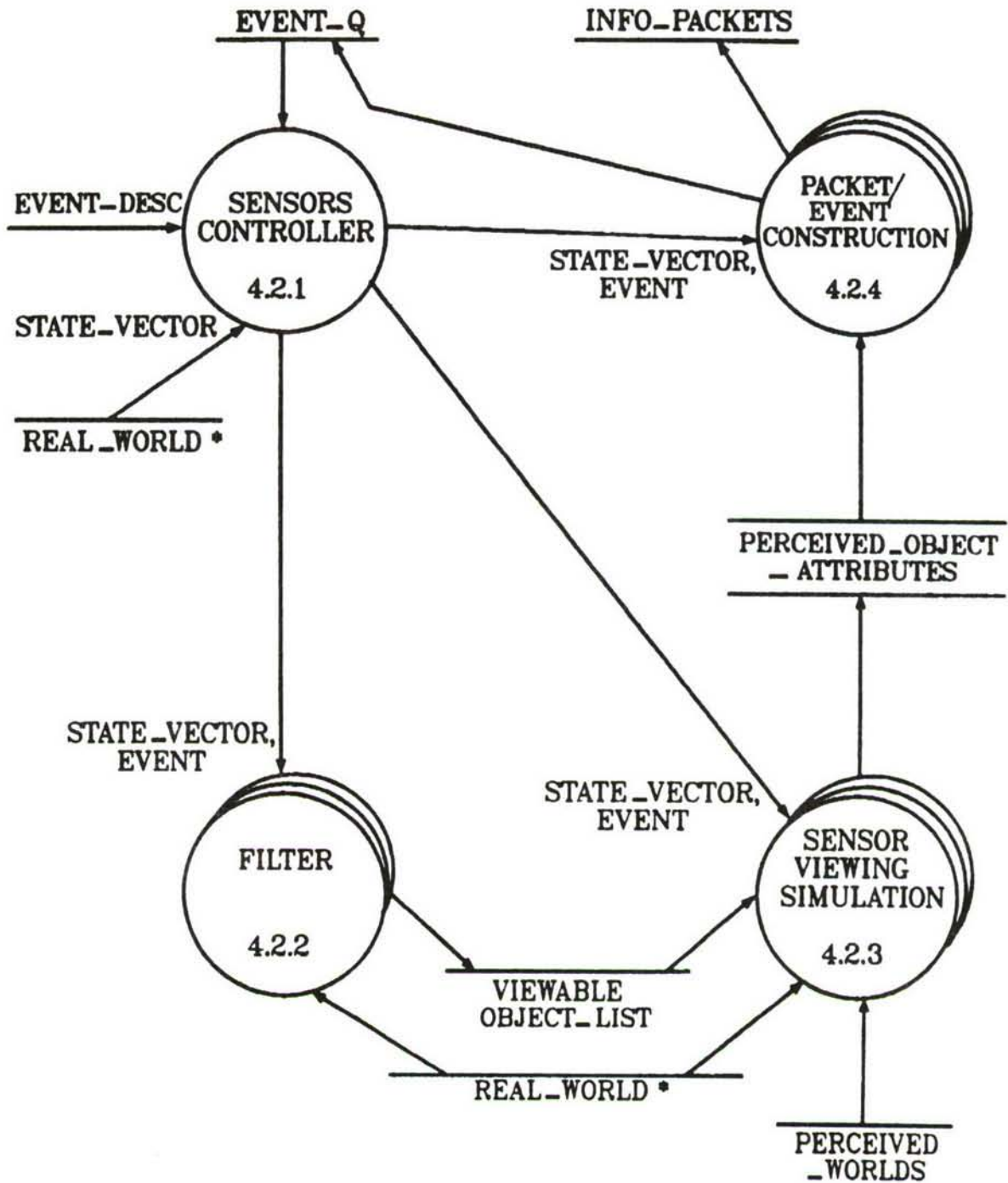
```
1
2
3
4 -----
5     bm - battle manager for generic function
6     4.1.1
7     -----
8
9     function - This module is supposed to give a rough
10    idea of how a real battle manager will use the data
11    and the type of control required.
12
13    input
14    event_desc - pointer to the event driving the battle manager
15
16    output - none
17
18    -----
19    bm (event_desc)
20
21    get event from event_q (event_desc, event_data)
22    get battle manager state_vector from real_world
23    extract damage from bm state_vector
24    if damage = dead, then
25        log receipt of event for dead asset
26    else /*asset may be functioning*/
27        if event_data indicates a packet accompanies the event, then
28            get packet from info_packets (event_data, packet)
29        end if
30        extract specific module number from event_data
31        case (mode)
32            when (first_step)
33                if time of next step < time of final event
34                    build event for first step (event_data)
35                    put into event_q (event_data)
36                end if
37            when (partial | step | last_step | instant)
38                if module is used, then
39                    determine battle mgr capability (event_data,
40                                                         <packet, state_vector, ok) /*4.1.1.n*/
41                    /* <xxx> indicates an optional parameter*/
42                end if
43                if ok = true /*if bm has capability to continue*/
44                    & module is used, then
45                        initial review of data (event_data, <packet>, state_vector, ok)
46                                                         /*4.1.2.n*/
47                end if
```

Level 2 & 3

## LEVEL 2 & 3 BATTLE MANAGERS

```
48     if ok = true /*if data satisfied the initial review*/
49         & module is used, then
50         analyze/evaluate operations (event_data, <packet>,
51             state_vector, recorded_decision, ok) /*4.1.3.n*/
52     end if
53     if ok = true /*if bm needs to continue this operation*/
54         & module is used, then
55         determine next operation (state_vector,
56             recorded_decision) /*4.1.4.n*/
57         if (mode = step & calculation will continue &
58             time of next step < time of final_event)
59             build event for next step (event_data)
60             put into event_q (event_data)
61         end if
62     else
63         error
64     end if
65     otherwise
66         error
67     end case
68 end if
69
70 end /*bm*/
```

## 4.2 SENSORS Level 2 Data Flow



## **LEVEL 2 & 3 SENSORS**

### **4.2 SENSORS**

#### **4.2.1 SENSORS CONTROLLER**

This module does the following: gets the **EVENT\_DATA** appropriate to the **EVENT\_DESC**, and analyzes the **EVENT\_DATA** to determine the **SENSOR\_OPERATING\_PARAMETERS** and a specific sensor ID. It performs the control functions to call the other modules within Sensors.

#### **4.2.2 FILTER**

This is one of a set of modules that uses the **SENSOR\_OPERATING\_PARAMETERS** and object **STATE\_VECTORS** to determine which objects are viewable by a particular sensor. The viewable **STATE\_VECTORS** are written to the **VIEWABLE\_OBJECTS\_LIST**.

#### **4.2.3 SENSOR VIEWING SIMULATION**

This is one of a set of modules, each of which simulates the functioning of a specific sensor type. It uses condition parameters from the appropriate installation **STATE\_VECTOR** along with **ENVIRONMENT** and possibly **GEOGRAPHY** to determine the appropriate perception of each object in the viewable objects list.

#### **4.2.4 EVENT / PACKET CONSTRUCTION**

This is a set of modules, one for each sensor type, which does the following:

- 1) performs routine analysis (analysis not requiring a Battle Manager) of the perceived data, e.g., sorts the objects according to whether or not they match a predetermined signature,
- 2) constructs the **PACKET(s)** to be transmitted to Battle Manager(s),
- 3) constructs an **EVENT** for the transmission to be accomplished, and
- 4) writes the **PACKET** and **EVENT** to the appropriate files.

## LEVEL 2 & 3 SENSORS

```
1
2
3 -----
4     SENSOR - SIMULATE SENSORS
5         4.2.1
6     -----
7
8     FUNCTION - A SENSOR FUNCTIONS AS A FILTER ON REAL-
9               AND PERCEIVED-WORLD DATA. SENSORS MAY OPERATE PERIODICALLY,
10              CONTINUOUSLY, OR UPON COMMAND.
11
12     INPUT
13         EVENT_DESC - THE EVENT DRIVING THIS ACTIVITY
14
15     OUTPUT
16         NONE
17
18     -----
19     SENSORS (EVENT_DESC)
20
21     get event from event_q (event_desc, event_data)
22     get sensor state_vector from real_world
23     extract damage from sensor state_vector
24     if damage = dead, then
25         log receipt of event for dead asset
26     else /*asset may be functioning*/
27         extract specific module number from event_data
28         case (mode)
29             when (first_step)
30                 if time of next step < time of final event, then
31                     build event for first step
32                     put into event_q (event_data)
33                 end if
34             when (partial | step | last step | instant)
35
36                 /*4.2.2 - determine if object is viewable*/
37                 call filter (state_vector, event)
38                 /*4.2.3 - do sensor viewing simulation*/
39                 call svx (state_vector, event)
40                 /*4.2.4 - construct packet and event*/
41                 call pec (state_vector, event)
42                 if mode = step & calculation is to continue &
43                     time of next step < time of final event
44                     build event for next step (event_data)
45                     put into event_q (event_data)
46                 end if
47
```

Level 2 & 3

## LEVEL 2 & 3 SENSORS

```
48         otherwise
49         error
50     end case
51 end if
52
53 END /*SENSOR*/
54
55 -----
56
57 FILTER - DETERMINE IF OBJECT IS VIEWABLE
58     4.2.2
59 -----
60
61 FUNCTION - THIS MODULE USES THE SENSOR OPERATING PARAMETERS
62 AND OBJECT STATE VECTORS TO DETERMINE WHICH OBJECTS
63 ARE VIEWABLE BY THE SENSOR. THE VIEWABLE OBJECTS'
64 STATE VECTORS ARE WRITTEN TO THE VIEWABLE.OBJECT.LIST FILE.
65
66 INPUT
67     STATE VECTOR - SENSOR STATE VECTOR
68     EVENT - THE EVENT DRIVING THIS ACTIVITY
69
70 OUTPUT
71     NONE
72
73 -----
74 FILTER (STATE VECTOR, EVENT)
75
76     get sensor_operating_parameters from event_data
77     find viewable sectors for sensor_operating_parameters
78     do for each viewable sector
79         do for each object in sector for class of interest
80             get state_vector for object from real_world
81             calculate if viewable
82             if viewable, then
83                 write object's state_vector to viewable_object_list
84             end if
85         end do
86     end do
87     do background for filter
88     write background sources to viewable_object_list
89
90 END /*FILTER*/
91
92 -----
93
94 SVS - SENSOR VIEWING SIMULATION
95     4.2.3
96 -----
```

## LEVEL 2 & 3 SENSORS

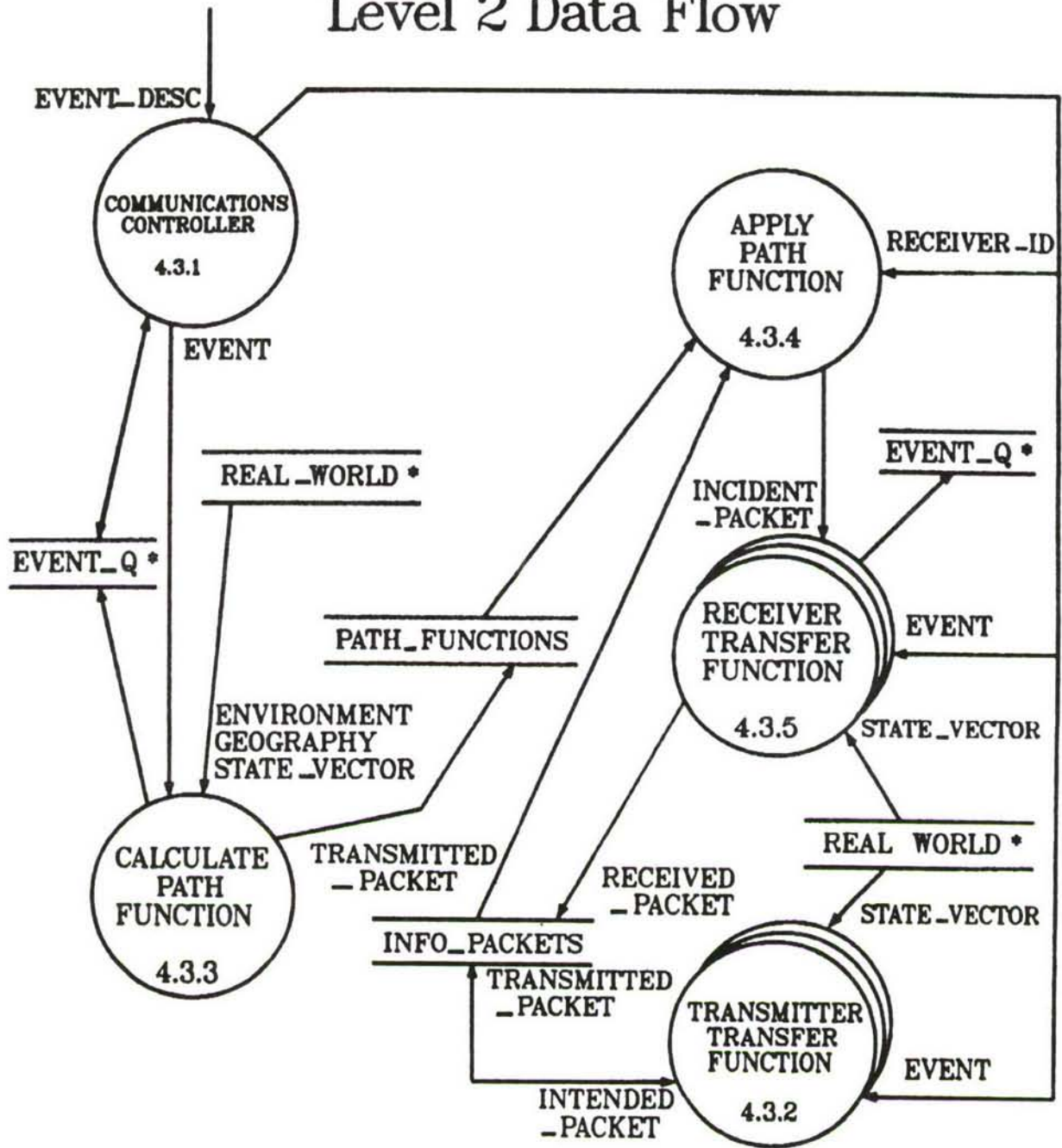
```
97
98 FUNCTION - THIS MODULE USES THE SENSOR STATE VECTOR ALONG
99 WITH ENVIRONMENT AND PERHAPS GEOGRAPHY TO DETERMINE
100 THE APPROPRIATE PERCEPTION OF AN OBJECT.
101
102 INPUT
103 STATE_VECTOR - SENSOR STATE VECTOR
104 EVENT - THE EVENT FOR THIS ACTIVITY
105
106 OUTPUT
107 NONE
108
109 -----
110 SVS (STATE_VECTOR, EVENT)
111
112 get sensor_operating_parameters from event_data
113 if other parameters are needed
114     get additional sensor_operating_parameters from state_vector
115 end if
116 do for each object in viewable_object_list
117     do viewing simulation to produce perceived_attributes
118     write perceived_attributes to perceived_object_attributes
119 end do
120 do sensor background simulation
121 write background to perceived_object_attributes
122
123 END /*SVS*/
124
125 -----
126 PEC - CONSTRUCT PACKET AND EVENT
127     4.2.4
128 -----
129
130 FUNCTION
131 THIS MODULE CONSTRUCTS ANY PACKET APPROPRIATE FOR THIS
132 EVENT, AND CONSTRUCTS AND STORES AN EVENT TO SEND THE PACKET.
133
134 INPUT
135 STATE_VECTOR - THE STATE VECTOR OF A PARTICULAR SENSOR
136 EVENT - THE EVENT DRIVING THIS ACTIVITY
137
138 OUTPUT
139 NONE
140
141 -----
142 PEC (STATE_VECTOR, EVENT)
143
144 determine if a packet is to be sent
145
```

Level 2 & 3

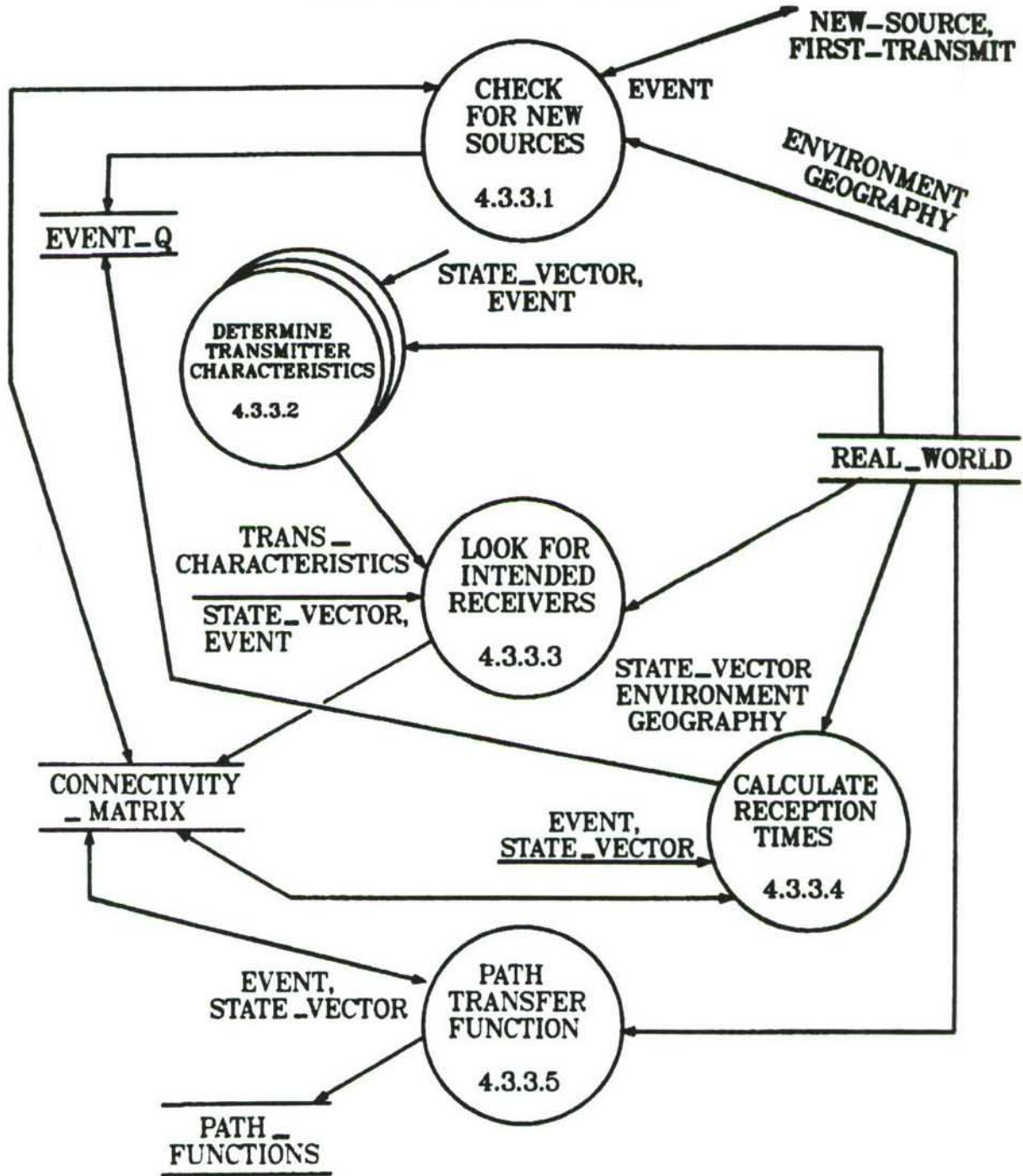
### LEVEL 2 & 3 SENSORS

```
146     if a packet is to be sent
147         construct the packet
148         store the packet in INFO_PACKETS
149         construct the EVENT to send the PACKET
150         put the EVENT in the EVENT_Q
151     end if
152     if sensor needs to run again
153         build event to restart sensor (event_data)
154         put into event_q (event_data)
155     end if
156
157 END /*PEC*/
```

## 4.3 COMMUNICATIONS Level 2 Data Flow



# 4.3.3 COMMUNICATIONS CALCULATE PATH FUNCTIONS Level 3 Data Flow



## **LEVEL 2 & 3 COMMUNICATIONS**

### **4.3.1 COMMUNICATIONS CONTROLLER**

This process does the overall control of the communication function.

### **4.3.2 TRANSMITTER TRANSFER FUNCTION**

This is a series of modules, each of which represents a type of transmitter. Each module uses the STATE\_VECTOR, including damage parameters for a particular transmitter installation, to transform the INTENDED\_PACKET into the TRANSMITTED\_PACKET. The TRANSMITTED\_PACKET may include parametric transmission characteristics such as frequency, antenna pattern type, power level, signal-to-noise ratio, and transmitter location. At some fidelity levels, the transmitter module functions may depend on ENVIRONMENT data from REAL\_WORLD.

### **4.3.3 CALCULATE PATH FUNCTION**

This module computes the transform for the communication path. For multiple paths to the same receiver, a composite transform is completed.

### **4.3.4 APPLY PATH FUNCTION**

The module does the actual convolution of the path transfer function with a TRANSMITTED\_PACKET to give an INCIDENT\_PACKET.

### **4.3.5 RECEIVER TRANSFER FUNCTION**

This is a series of modules, each representing a type of receiver. Each module uses the STATE\_VECTOR, including damage parameters for a particular receiver installation, to transform the INCIDENT\_PACKET into the RECEIVED\_PACKET. At some fidelity levels, the receiver module function may depend on ENVIRONMENT data from REAL\_WORLD.

#### **4.3.3.1 CHECK FOR NEW SOURCES**

This module determines if some new sources have become active that will affect the communications simulation. If this is true, it will be necessary to recompute the path transfer function.

#### **4.3.3.2 DETERMINE TRANSMITTER CHARACTERISTICS**

This is a series of modules, one for each type of transmitter. A module determines the characteristics for a particular transmitter. This is then used to calculate routing and receive times.

#### **4.3.3.3 LOOK FOR INTENDED RECEIVERS**

This module sets up the matrix which defines the receivers and the paths to those receivers for this particular transmission.

#### **4.3.3.4 CALCULATE RECEPTION TIMES**

This module calculates a vacuum reception time for each receiver identified by the CONNECTIVITY\_MATRIX. An event is generated for each receiver to simulate the reception of the message, and the event is put into EVENT\_Q.

## **LEVEL 2 & 3 COMMUNICATIONS**

### **4.3.3.5 PATH TRANSFER FUNCTION**

This module computes the transform for the communication path, including composite transforms, when required.

## LEVEL 2 & 3 COMMUNICATIONS

```
1
2
3 -----
4     COMM - COMMUNICATIONS
5         4.3.1
6     -----
7
8     FUNCTION - THIS FUNCTION SIMULATES COMMUNICATIONS
9               BETWEEN VARIOUS ASSETS USED IN THE SIMULATION.
10              MULTIPATH EFFECTS WILL BE SIMULATED.
11              ENVIRONMENTAL EFFECTS ARE ALSO
12              ACCOUNTED FOR, E.G., WEATHER, NUCLEAR.
13
14     INPUT
15         EVENT_DESC - THE EVENT DRIVING THIS ACTIVITY
16
17     OUTPUT
18         NONE
19
20     -----
21     COMM (EVENT_DESC)
22
23     GET EVENT FROM EVENT_Q (EVENT_DESC, EVENT_DATA)
24     if this is a transmit event, then
25         /*4.3.2 - transmitter transfer function*/
26         call ttf (event)
27         /*4.3.3 - calculate path functions*/
28         call cpf (event)
29     else /*receive event*/
30         /*4.3.3 - calculate path functions*/
31         call cpf (event)
32         /*4.3.4 - apply path function*/
33         call apf (event, incident_packet)
34         /*4.3.5 - receiver transfer function*/
35         call rtf (event, incident_packet)
36     end if
37
38     END /*COMM*/
39
40
41 -----
42     TTF - TRANSMITTER TRANSFER FUNCTION
43         4.3.2
44     -----
45
46     FUNCTION - THIS MODULE USES THE TRANSMITTER STATE VECTOR
47               INCLUDING DAMAGE PARAMETERS TO TRANSFORM AN
```

## LEVEL 2 & 3 COMMUNICATIONS

```
48     INTENDED_PACKET INTO THE TRANSMITTED PACKET.
49
50     INPUT
51     EVENT - THE EVENT DRIVING THIS ACTIVITY
52
53     OUTPUT
54     NONE
55
56     -----
57     TTF (EVENT)
58
59     CASE (MODE)
60     WHEN (FIRST_STEP)
61     IF TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
62     BUILD EVENT FOR FIRST STEP
63     PUT INTO EVENT_Q (EVENT_DATA)
64     END IF
65
66     WHEN (PARTIAL ! STEP ! LAST_STEP ! INSTANT)
67
68     get transmitter state_vector from real_world
69     get intended packet from info_packet
70     get damage parameters from state_vector
71     calculate transmitter transfer function (damage)
72     transmitted_packet = intended_packet * transmitter transfer
73     function
74     put transmitted_packet into info_packets
75
76     IF MODE = STEP & calculation will continue &
77     TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
78     BUILD EVENT FOR NEXT STEP (EVENT_DATA)
79     PUT INTO EVENT_Q (EVENT_DATA)
80     END IF
81
82     OTHERWISE
83     ERROR
84
85     END CASE
86     END /*TTF*/
87
88
89     -----
90     CPF - CALCULATE PATH FUNCTION
91     4.3.3
92     -----
93
94     FUNCTION - THIS MODULE COMPUTES THE TRANSFORM FOR THE
95     COMMUNICATION PATH.
96
```

## LEVEL 2 & 3 COMMUNICATIONS

```
97     INPUT
98     EVENT - THE EVENT DRIVING THIS ACTIVITY
99
100    OUTPUT
101    NONE
102
103    -----
104    CPF (EVENT)
105
106    get connectivity matrices [old] & [new]
107    get transmitter state_vector from real.world
108    if this is a receive event, then
109        do for each sector in [new]
110            search sector for sources
111            write sources in [new]
112        end do
113    end if
114        /*4.3.3.1 - check for new sources*/
115    call cns (event; new_source, first.transmit)
116    if new_source = true, then
117        if first.transmit = true, then
118            /*4.3.3.2 - determine transmitter characteristics*/
119            call dtc (event, state_vector, trans.characteristics)
120            /*4.3.3.3 - look for intended receivers*/
121            call lfir (event, trans.characteristics, state_vector)
122        end if
123        if this is a transmit event, then
124            do for each sector in [old]
125                search sector for sources
126                write sources in [old]
127            end do
128            /*4.3.3.4 - calculate reception times*/
129            call crt (event, state_vector)
130        else
131            /*4.3.3.5 - path transfer function*/
132            call ptf (event, state_vector)
133        end if
134    end if
135
136    END /*CPF*/
137
138    -----
139
140    APF - APPLY PATH FUNCTION
141    4.3.4
142    -----
143
144    FUNCTION
145    THIS MODULE DOES THE ACTUAL CONVOLUTION OF THE
```

## LEVEL 2 & 3 COMMUNICATIONS

```
146     PATH TRANSFER FUNCTION WITH THE TRANSMITTED_PACKET
147     TO GIVE THE INCIDENT_PACKET.
148
149     INPUT
150     EVENT - THE EVENT DRIVING THIS ACTIVITY
151
152     OUTPUT
153     INCIDENT_PACKET - THE PACKET ACTUALLY DELIVERED BY THE
154     COMMUNICATION PATH
155
156     -----
157     APF (EVENT, INCIDENT_PACKET)
158
159     get transmitted_packet from info_packets
160     get path transfer function from path_functions
161     incident_packet = transmitted_packet * path transfer function
162
163     END /*APF*/
164     -----
165     RTF - RECEIVER TRANSFER FUNCTION
166     4.3.5
167     -----
168
169     FUNCTION - THIS MODULE USES THE RECEIVER STATE VECTOR
170     INCLUDING DAMAGE PARAMETERS TO TRANSFORM THE
171     INCIDENT_PACKET TO THE RECEIVED_PACKET
172
173     INPUT
174     EVENT - THE EVENT DRIVING THIS ACTIVITY
175     INCIDENT_PACKET - THE PACKET RECEIVED AT THE RECEIVER
176
177     OUTPUT
178     NONE
179
180     -----
181     RTF (EVENT, INCIDENT_PACKET)
182
183     CASE (MODE)
184     WHEN (FIRST_STEP)
185     IF TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
186     BUILD EVENT FOR FIRST STEP
187     PUT INTO EVENT_Q (EVENT_DATA)
188     END IF
189
190     WHEN (PARTIAL | STEP | LAST_STEP | INSTANT)
191
192     get receiver state_vector from real world
193     get damage from state_vector
194     calculate receiver transfer function (damage)
```

## LEVEL 2 & 3 COMMUNICATIONS

```
195     received_packet = incident_packet *
196                       receiver transfer function
197     add received_packet to info_packets
198     build event to send received_packet
199       to next destination (event_data)
200     put into event_q (event_data)
201
202     IF MODE = STEP & calculation will continue &
203       TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
204       BUILD EVENT FOR NEXT STEP (EVENT.DATA)
205       PUT INTO EVENT_Q (EVENT.DATA)
206     END IF
207
208     OTHERWISE
209       ERROR
210
211     END CASE
212
213     END /*RTF*/
214
215
216 -----
217     CNS - CHECK FOR NEW SOURCES
218       4.3.3.1
219     -----
220
221     FUNCTION - THIS MODULE DETERMINES IF SOME NEW SOURCE
222       HAS BECOME ACTIVE THAT MIGHT AFFECT COMMUNICATIONS.
223
224     INPUT
225       EVENT - THE EVENT DRIVING THIS ACTIVITY
226
227     OUTPUT
228       NEW_SOURCE - LOGICAL VARIABLE DENOTING THE PRESENCE
229         OF NEW INTERFERENCE SOURCES
230       FIRST_TRANSMIT - LOGICAL VARIABLE DENOTING FIRST TRANSMISSION
231
232     DEFINITIONS:
233       [...] DENOTES A MATRIX
234       |[...]| DENOTES THE NORM OF A MATRIX
235
236     -----
237     CNS (EVENT; NEW_SOURCE, FIRST_TRANSMIT)
238
239     check connectivity submatrix [new] for emptiness
240     if empty, then
241       new_source = true
242       first_transmit = true
243     else
```

Level 2 & 3

## LEVEL 2 & 3 COMMUNICATIONS

```
244     if |[old]-[new]| > 0 , then
245         new_source = true
246         flag active sources in [new]
247         flag active sectors in [new]
248     else
249         new_source = false
250     end if
251     first.transmit = false
252 end if
253
254 END /*CNS*/
255
256 -----
257
258 DTC - DETERMINE TRANSMITTER CHARACTERISTICS
259     4.3.3.2
260 -----
261
262 FUNCTION - THIS MODULE DETERMINES THE CHARACTERISTICS
263 FOR SOME PARTICULAR COMMUNICATION TRANSMITTER
264
265 INPUT
266     EVENT - THE EVENT DRIVING THIS ACTIVITY
267     STATE.VECTOR - THE TRANSMITTER STATE.VECTOR
268
269 OUTPUT
270     TRANS.CHARACTERISTICS - CHARACTERISTICS FOR A TRANSMITTER
271
272 -----
273 DTC (EVENT, STATE.VECTOR, TRANS.CHARACTERISTICS)
274
275 END /*DTC*/
276
277 -----
278
279 LFIR - LOOK FOR INTENDED RECEIVERS
280     4.3.3.3
281 -----
282
283 FUNCTION - THIS MODULE ADDS TO THE CONNECTIVITY MATRIX
284 THAT DEFINES THE PATHS AND RECEIVERS FOR A PARTICULAR
285 TRANSMITTER. "OLD" AND "NEW" VERSIONS OF THE MATRIX ARE
286 WRITTEN TO A FILE. BOTH "OLD" AND "NEW" MATRICES MUST
287 BE INITIALIZED AT THE BEGINNING OF THE SIMULATION.
288
289 DEFINITIONS -
290     tbnd = transmitter frequency band
291     rbnd = receiver frequency band
292
```

## LEVEL 2 & 3 COMMUNICATIONS

```
293 INPUT
294 EVENT - THE EVENT DRIVING THIS ACTIVITY
295 TRANS_CHARACTERISTICS - CHARACTERISTICS FOR TRANSMITTER
296 STATE_VECTOR - THE TRANSMITTER STATE VECTOR
297
298 OUTPUT
299 NONE
300
301 -----
302 LFIR (EVENT, TRANS_CHARACTERISTICS, STATE_VECTOR)
303
304 get trans_sv from real.world
305 parse trans_sv for frequency characteristics (trans_sv; tbnd)
306 get intended receiver list from event_desc
307 do for all intended receivers
308     get rec_sv from real.world
309     parse rec_sv for frequency characteristics (rec_sv; rbnd)
310     if tbnd within rbnd, then
311         set path = no
312         check transrec pair for los (trans_sv, rec_sv)
313         if los = yes, then
314             write receiver_id in connectivity matrices [old] & [new]
315             find sectors along los
316             write sectors in connectivity matrices [old] & [new]
317             set path = direct
318             write path in connectivity matrices [old] & [new]
319         end if
320         check for additional paths /*do waveguides exist?*/
321         do for each alternate path
322             write receiver_ID in connectivity matrices [old] & [new]
323             find sectors along alternate path
324             write sectors in connectivity matrices [old] & [new]
325             set path = indirect
326             write path in connectivity matrices [old] & [new]
327         end do
328         if path = no, then
329             write "no path" message
330         end if
331         else
332             write "ineligible receiver" message
333         end if
334     end do
335
336 END /*LFIR*/
337
338 -----
339
340 CRY - CALCULATE RECEPTION TIMES
341 4.3.3.4
```

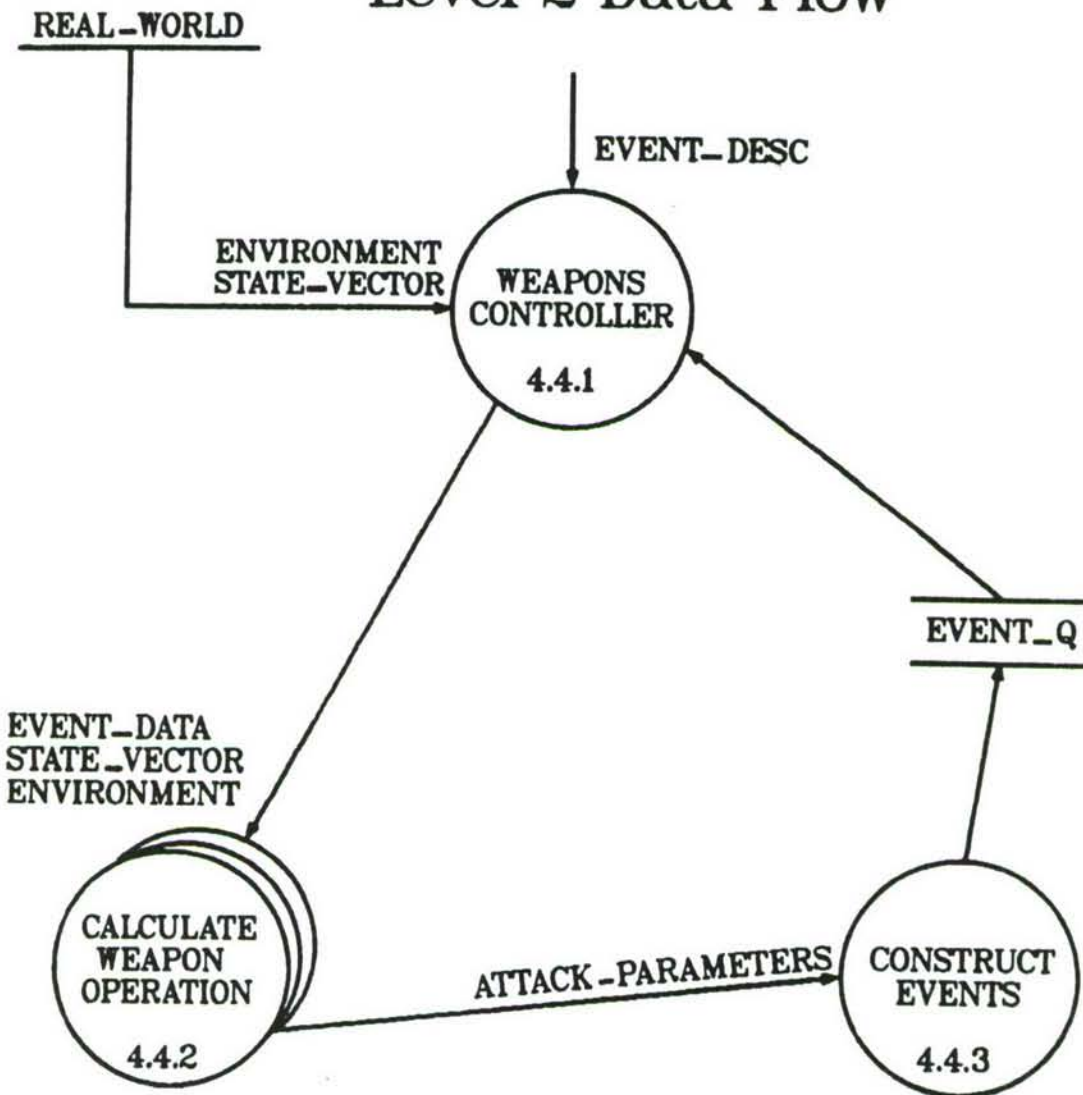
## LEVEL 2 & 3 COMMUNICATIONS

```
342 -----
343
344 FUNCTION - THIS MODULE CALCULATES A RECEPTION TIME
345 FOR EACH RECEIVER IDENTIFIED BY THE
346 CONNECTIVITY_MATRIX. EVENTS ARE BUILT AND PUT INTO
347 THE EVENT_Q TO SIMULATE THE RECEPTION OF THE MESSAGES.
348
349 INPUT
350     EVENT - THE EVENT DRIVING THIS ACTIVITY
351     STATE_VECTOR - THE TRANSMITTER STATE_VECTOR
352
353 OUTPUT
354     NONE
355
356 -----
357 CRI (EVENT, STATE_VECTOR)
358
359     ttrn = time from event data
360     get transmitter_sv and geography from real.world
361     get conn_matrix
362     do for each receiver in connectivity matrix
363         get receiver_sv from real.world
364         do for each path
365             calculate vacuum transit time
366             build event to simulate reception (event_data)
367             put into event_q (event_data)
368         end do
369     end do
370
371
372 END /*CRI*/
373
374
375 -----
376 PIF - PATH TRANSFER FUNCTION
377     4.3.3.5
378     -----
379
380 FUNCTION - THIS MODULE COMPUTES THE TRANSFORM FOR THE
381 COMMUNICATION PATH.
382
383 INPUT
384     EVENT - THE EVENT DRIVING THIS ACTIVITY
385     STATE_VECTOR - THE TRANSMITTER STATE_VECTOR
386
387 OUTPUT
388     NONE
389
390 -----
```

## LEVEL 2 & 3 COMMUNICATIONS

```
391      PTF (EVENT, STATE-VECTOR)
392
393      get path lengths and directions from connectivity_matrix [new]
394      calculate hard link parameters
395      calculate soft link parameters /*primary and secondary paths,
396                                     over or in land and water
397                                     or ice*/
398      get weather sources from [new]
399      calculate weather perturbations /*solar and terrestrial*/
400      get nuclear effects sources from connectivity_matrix [new]
401      calculate nuclear effects perturbations
402      calculate path transfer function
403      put path transfer function into path_functions
404
405      END /*PTF*/
406
```

# 4.4 WEAPONS Level 2 Data Flow



## **LEVEL 2 & 3 WEAPONS**

### **4.4.1 WEAPONS CONTROLLER**

There is a weapon's module or process for each type of weapon. The controller does the standard functions of getting data from the REAL\_WORLD, getting and building EVENTS, and determining which weapon modules to call.

### **4.4.2 CALCULATE WEAPON OPERATION**

This series of modules uses the EVENT\_DATA to calculate ATTACK\_PARAMETERS that completely and parametrically describe the operation of the weapon. They may also identify future weapon operation that must be calculated.

### **4.4.3 CONSTRUCT EVENTS**

This module constructs EVENTS using ATTACK\_PARAMETERS. The events are used to call Event Physics immediately or to recall the required weapon's module.

## LEVEL 2 & 3 WEAPONS

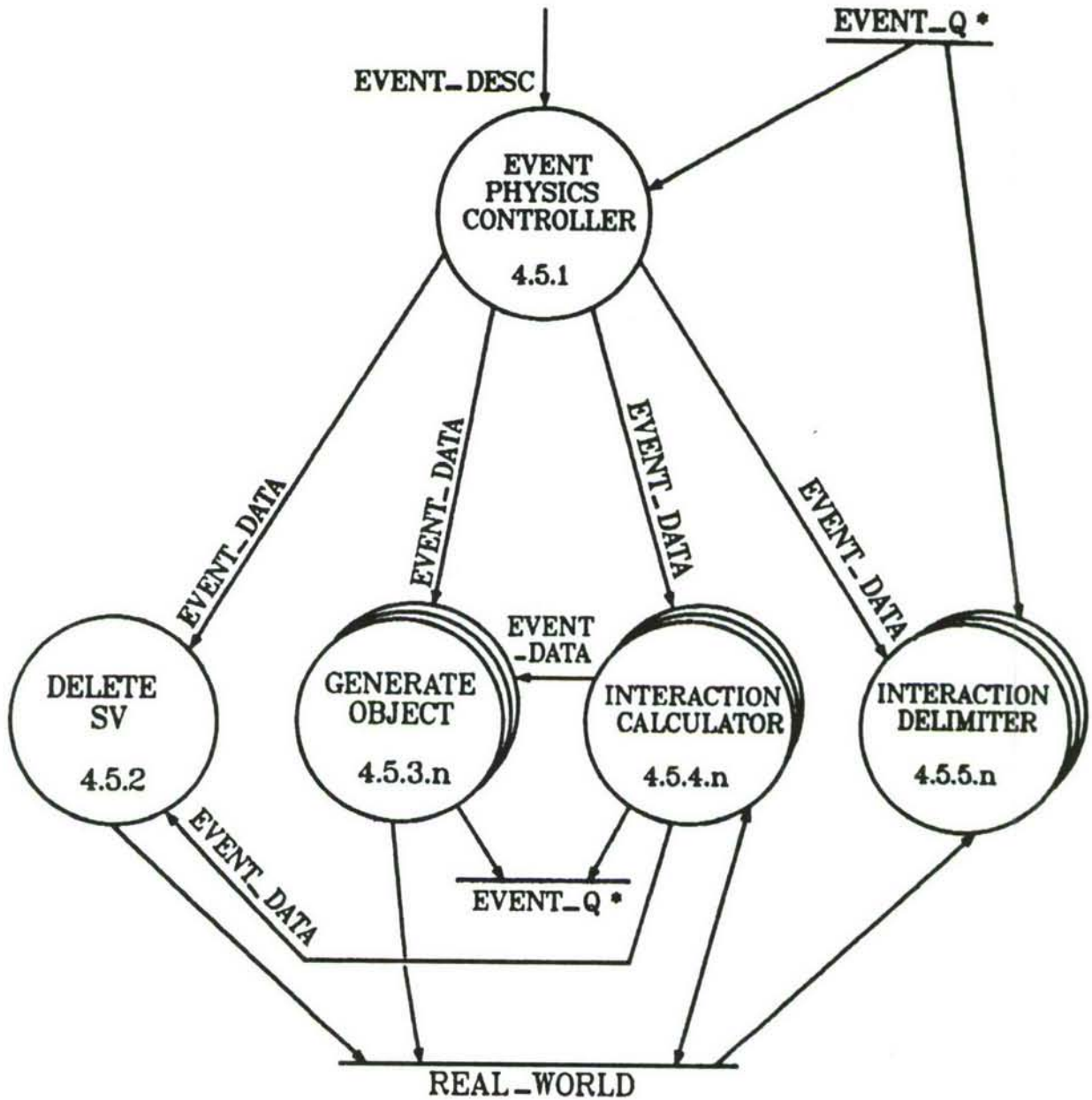
```
1
2
3 -----
4 WEAPONS - SIMULATE WEAPONS
5   4.4.1
6 -----
7
8 FUNCTION
9   THE WEAPON'S MODULE CALCULATES THE ACTUAL FUNCTION
10  FOR A WEAPON SUCH AS YIELD FOR A BOMB
11  OR BEAM DURATION AND INTENSITY FOR A LASER.
12
13 INPUT
14   EVENT_DESC - THE EVENT DRIVING THIS ACTIVITY
15
16 OUTPUT
17   NONE
18
19 -----
20 WEAPONS (EVENT_DESC)
21
22 get event from event_q (event_desc, event_data)
23
24 CASE (MODE)
25   WHEN (FIRST_STEP)
26     IF TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
27       BUILD EVENT FOR FIRST STEP
28       PUT INTO EVENT_Q (EVENT_DATA)
29     END IF
30
31   WHEN (PARTIAL | STEP | LAST_STEP | INSTANT)
32
33     get state_vector for this weapon from real.world
34     get this weapon's environment from real.world
35     /*4.4.2*/
36     calculate operation for this weapon (state_vector,
37                                         event_data, environment,
38                                         attack_parameters)
39
40     /*4.4.3 - construct events*/
41     if weapon must function at a later time
42       build event to continue function (attack_parameters,
43                                         event_data)
44     add to event_q (event_data)
45     end if
46     build event to execute event physics immediately (event_data)
```

## LEVEL 2 & 3 WEAPONS

```
47      add to event_q (event_data)
48      /*end 4.4.3*/
49
50      IF MODE = STEP & calculation is to continue &
51          TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
52          BUILD EVENT FOR NEXT STEP (EVENT_DATA)
53          PUT INTO EVENT_Q (EVENT_DATA)
54      END IF
55
56      OTHERWISE
57          ERROR
58      END CASE
59
60      END /*WEAPONS*/
```

Level 2 & 3

# 4.5 PHYSICS Level 2 Data Flow



## **LEVEL 2 & 3 EVENT PHYSICS**

### **4.5 EVENT PHYSICS**

#### **4.5.1 EVENT PHYSICS CONTROLLER**

This module uses the `EVENT_DESC` given to it by the Executive to get the appropriate `EVENT_DATA`. It then uses information extracted from the `EVENT_DATA` to call the appropriate module to perform the event.

#### **4.5.2 DELETE SV**

This module extracts a `SV_ID` from the `EVENT_DATA` and deletes the `STATE_VECTOR` with this ID from the `REAL_WORLD` file.

#### **4.5.3.n GENERATE OBJECT**

This is a series of modules, each of which uses parameters from `EVENT_DATA` to create a complete parameter set for a new object. It converts this parameter set into `STATE_VECTOR` format and stores the `STATE_VECTOR` in the `REAL_WORLD` file.

#### **4.5.4.n INTERACTION CALCULATOR**

This is a series of modules, each of which calculates a type of force-on-asset interaction. Each module is conceptually of the "zeroth-order convolution" optionally followed by "interaction loops" that refine the zeroth-order calculations.

#### **4.5.5.n INTERACTION DELIMITER**

This is a set of modules, each of which accomplishes the following for a particular effect type:

- 1) identifies affected assets
- 2) creates `EVENTS` to trigger the interaction calculations.

In the process of identifying the affected assets, it may update object positions and/or conditions beyond the last Mother Nature call, but these updates will not be stored in the `REAL_WORLD` file.

## LEVEL 2 & 3 EVENT PHYSICS

```
1
2
3 -----
4 EP - EVENT PHYSICS
5   4.5.1
6   -----
7
8 FUNCTION
9   THIS MODULE GETS THE EVENT INFORMATION AND THEN SELECTS
10  THE MODULE REQUIRED TO EXECUTE THE EVENT.
11
12 INPUT
13   EVENT_DESC - POINTER TO THE EVENT DRIVING THIS ACTIVITY
14
15 OUTPUT
16   NONE
17
18 -----
19 EP (EVENT_DESC)
20
21 get event from event_q (event_desc, event_data)
22 extract ep.type from event_data
23 case (ep.type)
24   when (delete_sv)
25     /*4.5.2 - delete state vector*/
26     call delsv (event_data)
27   when (generate_object)
28     extract object.type from event_data
29     case object.type
30       when (object.type n)
31         /*4.5.3.n - generate object*/
32         call genobj (event_data)
33     .
34     .
35     .
36   end case
37   when (interaction)
38     extract interaction.type from event_data
39     case (interaction.type)
40       when (interaction.type n)
41         /*4.5.4.n - interaction calculator*/
42         call incalul (event)
43     .
44     .
45     .
46   end case
47   when (interaction_delimiter)
```

## LEVEL 2 & 3 EVENT PHYSICS

```
48      extract interaction_delimiter_type from event_data
49      case (interaction_delimiter_type)
50          when (interaction_delimiter_type n)
51              /*4.5.5.n - interaction delimiter*/
52              call indelim (event_data)
53          .
54          .
55          .
56      end case
57  end case
58
59  END /*EP*/
60
61 -----
62  DELSV - DELETE STATE VECTOR
63      4.5.2
64 -----
65
66  FUNCTION
67      THIS MODULE FINDS THE IDENTIFICATION AND
68      LOCATION OF THE STATE VECTOR AND DELETES IT FROM
69      THE REAL_WORLD.
70
71  INPUT
72      EVENT_DATA - THE EVENT DEFINING THE STATE VECTOR TO DELETE
73
74  OUTPUT
75      NONE
76
77 -----
78  DELSV (EVENT_DATA)
79
80  extract sv_id from event_data
81  find state_vector in real_world (sv_id)
82  delete state_vector from real_world (sv_id)
83
84  END /*DELSV*/
85
86 -----
87
88  GENOBJ - GENERATE OBJECT
89      4.5.3.n
90 -----
91
92  FUNCTION
93      THIS MODULE USES THE PARAMETERS FROM EVENT_DATA TO
94      CREATE THE PARAMETER SET FOR A NEWLY CREATED OBJECT.
95      THESE PARAMETERS ARE STORED IN STATE VECTOR FORMAT
96
```

Level 2 & 3

## LEVEL 2 & 3 EVENT PHYSICS

```
97      IN REAL_WORLD.
98
99      INPUT
100     EVENT_DATA - THE EVENT CAUSING THE OBJECT TO BE CREATED
101
102     OUTPUT
103     NONE
104
105     -----
106     GENOBJ (EVENT_DATA)
107
108     generate object
109     construct state_vector
110     put state_vector into real_world
111     if object requires mother nature update, then
112         build event to run mother nature (event_data)
113         put into event_q (event_data)
114     end if
115
116     END /*GENOBJ*/
117
118
119     -----
120     INCALUL - INTERACTION CALCULATOR
121     4.5.4.n
122     -----
123
124     FUNCTION
125     THIS MODULE CALCULATES A TYPE OF FORCE ON ASSET
126     INTERACCOCTION. CONCEPTUALLY THIS IS A ZEROth-ORDER
127     CONVOLUTION OPTIONALLY FOLLOWED BY INTERACTION
128     LOOPS THAT REFINE THE ZEROth.ORDER CALCULATION.
129
130     INPUT
131     EVENT - THE EVENT DEFINING THE CALCULATION
132
133     OUTPUT
134     NONE
135
136     -----
137     INCALCU (EVENT)
138
139     CASE (MODE)
140     WHEN (FIRST_STEP)
141         IF TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
142             CALCULATE TIME OF FIRST STEP
143             BUILD EVENT FOR FIRST STEP
144             PUT INTO EVENT_Q (EVENT_DATA)
145         END IF
```

## LEVEL 2 & 3 EVENT PHYSICS

```
146
147     WHEN (PARTIAL ! STEP ! LAST_STEP ! INSTANT)
148
149         get state_vectors, geography, environment from real world
150         update fast state_vectors, fast environment components
151         do zeroth order convolution
152         do interaction loops to refine zeroth order
153         modify state_vector to reflect new damage parameters
154         replace state_vector in real_world with modified state_vector
155         if interaction calculation spawned new objects
156             do for each new object
157                 construct event_data to do generation
158                 /*4.5.3 - generate object*/
159                 call genobj (event_data)
160             end do
161         end if
162         if interaction calculation required deletion of objects
163             do for each object to be deleted
164                 construct event_data for deletion
165                 /*4.5.2 - delete state vector*/
166                 call delsv (event_data)
167             end do
168         end if
169
170         IF MODE = STEP & calculation will continue &
171             TIME OF NEXT STEP < TIME OF FINAL EVENT, THEN
172             CALCULATE TIME OF NEXT STEP
173             BUILD EVENT FOR NEXT STEP (EVENT_DATA)
174             PUT INTO EVENT_Q (EVENT_DATA)
175         END IF
176
177     OTHERWISE
178         ERROR
179
180     END CASE
181
182     END /*INCALCU*/
183
184
185 -----
186     INDELIM - INTERACTION DELIMITER
187         4.5.5.n
188     -----
189
190     FUNCTION
191         FOR A PARTICULAR EFFECT TYPE, THIS MODULE IDENTIFIES
192         AFFECTED ASSETS AND CREATES EVENTS TO TRIGGER THE
193         INTERACTION CALCULATIONS. THE IDENTIFICATION PROCESS
194         MAY REQUIRE THAT OBJECT POSITIONS OR
```

## LEVEL 2 & 3 EVENT PHYSICS

```
195     CONDITIONS BE UPDATED BEYOND THE POINT OF THE LAST MOTHER
196     NATURE CALL.  THESE UPDATES HOWEVER ARE NOT SAVED IN
197     THE REAL WORLD.
198
199     INPUT
200     EVENT_DATA - THE DATA DEFINING THE INTERACTION
201
202     OUTPUT
203     NONE
204
205     -----
206     INDELIM (EVENT_DATA)
207
208     get state_vectors, geography, environment from real_world
209     update fast state_vectors, fast environment components
210     do for each state_vector in question
211         decide if it will interact
212         if it will interact
213             find out when it will interact
214             construct event to do the interaction
215             put event in the event_q
216         end if
217     end do
218
219     END /*INDELIM*/
```

## 8. DATA DEFINITIONS

DATA DEFINITIONS	PAGE
ATTACK_PARAMETERS	8-4
ATTACK_PLAN	8-4
BATTLE_MGR_DIRECTIVE	8-4
CMD_ATTACK_PLAN	8-4
CMD_ENVIRONMENT	8-5
CMD_GEOGRAPHY	8-5
CMD_INTERVENTIONS	8-5
CMD_MODULES	8-6
CMD_ORDER_OF_BATTLE	8-6
CMD_OUTPUT_SPECS	8-6
CMD_PLANNED_RESPONSES	8-7
CMD_PROMPT	8-7
CMDS	8-7
CODE_DATA	8-8
CODE_STATUS	8-8
CONFLICT	8-8
CONFLICT_LIST	8-9
CONNECTIVITY_MATRIX	8-9
CONSTANTS	8-10
DAMAGE	8-10
DATE_TIME	8-10
DESTINATION	8-11
DISPLAY	8-11
DO_EVENT	8-11
DO_NOW	8-12
ENVIRONMENT	8-12
EP_TYPE	8-12
ERROR_MSG	8-13
EV_DATA	8-13
EVENT	8-14
EVENT_DATA (generic)	8-14
EVENT_DATA_PTR	8-14
EVENT_DESC	8-15
EVENT_Q	8-15
FIRST_TRANSMIT	8-16
GEOGRAPHY	8-16
GLOBAL_SIMULATION_DATA	8-16
GRID_QUANTITY	8-16
INCIDENT_PACKET	8-17
INFO_DATA	8-17
INFO_PACKETS	8-17
INFO_PTR	8-18
IN_PROGRESS	8-18
INTENDED_PACKET	8-18
INTERACTION_DELIMITER_TYPE	8-19
INTERACTION_TYPE	8-19
INTERRUPT	8-19

## DATA DEFINITIONS

## DATA DEFINITIONS

INTERVENTIONS	8-20
MENU	8-20
MN_AGAIN	8-21
MN_EVENT_PARAMETERS	8-21
MN_GEOGRAPHY	8-21
MN_GEOGRAPHY_REQUEST	8-22
MN_SOURCES	8-22
MODE	8-22
MODULE_LIBRARY	8-23
MODULE_NAME	8-23
MSG_CODE	8-24
MSG_TEXT	8-24
NEXT_TIME	8-24
NEW_SOURCE	8-25
OBJECT_TYPE	8-25
ORDER_OF_BATTLE	8-25
ORIENTATION	8-25
OUTPUT_LOG	8-26
PACKET (generic)	8-26
PATH_FUNCTIONS	8-27
PERCEIVED_ATTRIBUTES	8-27
PERCEIVED_OBJECT_ATTRIBUTES	8-27
PERCEIVED_WORLDS	8-28
PHYSICAL_OUTPUT	8-28
PLANNED_RESPONSE	8-28
POSITION	8-29
POST_PROCESSING	8-29
PREV_TIME	8-29
REAL_WORLD	8-30
RECEIVED_PACKET	8-30
RECEIVED_ID	8-30
RECORDED_DECISION	8-31
RECORDED_OBSERVATION	8-31
REF_LIBRARY	8-31
RESTART_DUMPS	8-32
SAVE_SV	8-32
SCENARIO	8-32
SCENARIO_LIBRARY	8-33
SENSOR_OPERATING_PARAMETERS	8-33
SIM_DATA	8-34
SIM_STATUS	8-34
SOURCE	8-34
STATE_VECTOR (generic)	8-35
SV_CLASS_ID	8-35
SV_ID	8-35
SYSTEM_TABLES	8-36
TEXT_MSG	8-36
TIME	8-36
TRANS_CHARACTERISTICS	8-37

## DATA DEFINITIONS

## DATA DEFINITIONS

TRANSMITTED_PACKET	8-37
TRANSMITTER_ID	8-37
VIEWABLE_OBJECT_LIST	8-38
WARNING_MSG	8-38

## DATA DEFINITIONS

NAME: ATTACK\_PARAMETERS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: This is information describing how a weapon functions (e.g. yield, duration) for inclusion in EVENT\_DATA.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: ATTACK\_PLAN

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: Offensive plans for the battle managers.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: BATTLE\_MGR\_DIRECTIVE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: A recorded order sent by a battle manager to some device under its control.

COMPOSITION: EVENT

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_ATTACK\_PLAN

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

DESCRIPTION: The user commands to specify the attack plan.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_ENVIRONMENT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the user commands to specify the environment for the simulation.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_GEOGRAPHY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the user commands to specify the geography.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_INTERVENTIONS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the user commands to specify events or actions that interrupt, change, override, or report on activities.

COMPOSITION:

## DATA DEFINITIONS

## DATA DEFINITIONS

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_MODULES

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the user commands used to select the particular code modules to be used for the simulation. The code modules selected will define the fidelity of the simulation.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_ORDER\_OF\_BATTLE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: The user commands to specify the initial order of battle including equipment and forces.

COMPOSITION:

ORGANIZATION ( if file ):

NOTES:

---

NAME: CMD\_OUTPUT\_SPECS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the user commands used to define the output that the simulation is to collect and present.

COMPOSITION:

ORGANIZATION ( if file ):

## DATA DEFINITIONS

## DATA DEFINITIONS

### NOTES:

---

NAME: CMD\_PLANNED\_RESPONSES

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: These are the user commands that specify and select the defensive plans for the battle manager.

### COMPOSITION:

ORGANIZATION ( if file ):

### NOTES:

---

NAME: CMD\_PROMPT

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: These are the prompts that assist the user in deciding what to do next.

### COMPOSITION:

ORGANIZATION ( if file ):

### NOTES:

---

NAME: CMDS

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW| FILE

DESCRIPTION: These are the commands that allow the user to setup, run, and observe the code and simulation.

### COMPOSITION:

{CMD\_ORDER\_OF\_BATTLE /\* the initial order of  
battle - this includes equipment and forces \*/  
+ CMD\_ATTACK\_PLAN + CMD\_PLANNED\_RESPONSES}  
+ {CMD\_INTERVENTION} /\* events to trigger natural

## DATA DEFINITIONS

## DATA DEFINITIONS

and man-imposed actions \*/  
+ CMD\_MODULES + CMD\_GEOGRAPHY + CMD\_ENVIRONMENT  
+ CMD\_OUTPUT\_SPECS

ORGANIZATION (if file):

NOTES: Model fidelity is determined by the selection of modules for the simulation. For most cases fidelity will be mixed.

---

NAME: CODE\_DATA

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is information that defines the state or status of the simulation program.

COMPOSITION: binary representation of code system status as to written the POST\_PROCESSING and RESTART\_DUMPS files.

ORGANIZATION (if file):

NOTES: The CODE\_DATA definition applies to binary machine representation of data that the user has requested.

---

NAME: CODE\_STATUS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the status or state of the actual simulation program -not the simulation -as displayed for the user.

COMPOSITION: Text representation of the CODE\_DATA data definition.

ORGANIZATION (if file):

NOTES: The CODE\_STATUS definition applies to text representation of data that the user has requested.

---

NAME: CONFLICT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

DESCRIPTION: This is a flag that defines whether a conflict exists or not.

COMPOSITION: TRUE | FALSE

TRUE - a conflict exists

FALSE - no conflict

ORGANIZATION (if file):

NOTES:

---

NAME: CONFLICT\_LIST

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a list of entries from the IN\_PROGRESS file that are in conflict with one of the assets from a particular event.

COMPOSITION: {SV.ID + FINAL\_EVENT\_PTR}

ORGANIZATION (if file):

NOTES:

---

NAME: CONNECTIVITY\_MATRIX

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This matrix which actually is two submatrices contains the possible communication paths and receivers for a packet transmission from a particular transmitter. The most general (high-fidelity) form of the matrix is five-dimensional, with indices (I.J.K.L.M).

I identifies the transmitter,

J identifies the receiver(s),

K identifies the transmission path(s),

L identifies the sector(s) that contain the K-th path, and

M identifies the communications interference sources.

COMPOSITION: old + new

## DATA DEFINITIONS

## DATA DEFINITIONS

new - submatrix for new transmissions  
old - submatrix for existing transmissions.

ORGANIZATION (if file):

NOTES:

---

NAME: CONSTANTS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: The standard constants used throughout the simulation program:  
e.g. pi, the speed of light, etc.

COMPOSITION:

ORGANIZATION (if file):

NOTES: Constants will not be hardcoded into the program. They will be  
defined once in this file and loaded into the code at  
initialization time.

---

NAME: DAMAGE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This specifies the fraction of damage to an asset.

COMPOSITION:

DEAD = 1 ALIVE = 0 ALIVE  $\leq$  DAMAGE  $\leq$  DEAD

ORGANIZATION: (if file):

NOTES:

---

NAME: DATE.TIME

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the standard format for text date and time.

## DATA DEFINITIONS

## DATA DEFINITIONS

COMPOSITION:

ORGANIZATION (if file):

NOTES: We may wish to have this information in many different formats.

---

NAME: DESTINATION

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This identifies the module to be executed.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: DISPLAYS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the terminal displays for the user of the simulation.

COMPOSITION:

[MENU | CMD\_PROMPT] /\* prompting via menus or text \*/  
+ CODE\_STATUS /\* status of the simulation program \*/  
+ SIM\_STATUS /\* status of the simulated battle-maps,  
bar charts, trajectories, etc. \*/

ORGANIZATION (if file):

NOTES:

---

NAME: DO\_EVENT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a flag returned by the conflict handler to the executive controller indicating whether the new event should be

## DATA DEFINITIONS

## DATA DEFINITIONS

executed or not.

COMPOSITION: TRUE | FALSE

TRUE - execute event  
FALSE - do not execute the event

ORGANIZATION (if file):

NOTES:

---

NAME: DO\_NOW

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a flag in an event that indicates whether an event can be executed now or if it must go to the conflict handler first.

COMPOSITION: TRUE | FALSE

ORGANIZATION (if file):

NOTES: TRUE - EVENT can be executed without going to conflict handler  
FALSE - EVENT must go to conflict handler.

---

NAME: ENVIRONMENT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: The physical world's environment.

COMPOSITION: weather + space + eme\* + etc

ORGANIZATION (if file):

NOTES: \* eme is defined as ElectroMagnetic Environment.

---

NAME: EP\_TYPE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

DESCRIPTION: Part of EVENT\_DATA, this parameter denotes which of several types of calculations is to be performed by EVENT\_PHYSICS. It may legally acquire the following values:

DELETE\_SV  
GENERATE\_OBJECT  
INTERACTION\_DELIMITATION  
INTERACTION.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: ERROR\_MSG

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: The standard format text message put out when the simulation program detects an error.

COMPOSITION: TEXT\_MSG

ORGANIZATION (if file):

NOTES:

---

NAME: EV\_DATA

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are event specific data that are part of the EVENT\_DATA definition. Each event type will most likely have a different format for this structure.

COMPOSITION: [PREV.TIME] + event dependent data

ORGANIZATION (if file):

NOTES:

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: EVENT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is an activity that is to occur at some specific time.

COMPOSITION: EVENT\_DESC + EVENT\_DATA

ORGANIZATION (if file):

NOTES:

---

NAME: EVENT\_DATA (generic)

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the (generic) structure of the data part of the EVENT\_Q file. Each event type will most likely have a different EV\_DATA format.

COMPOSITION:

TIME /\* when the event is to take place \*/  
+ SOURCE /\* who originated the event \*/  
+ DESTINATION /\* module to be executed \*/  
+ EV\_DATA /\* event specific information \*/

ORGANIZATION (if file):

NOTES:

---

NAME: EVENT\_DATA\_PTR

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the address of EVENT\_DATA in the EVENT\_Q file.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

## DATA DEFINITIONS

## DATA DEFINITIONS

---

NAME: EVENT\_DESC

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the structure of the pointer part of the EVENT\_Q file.

COMPOSITION:

TIME /\* when the event is to take place\*/  
+ EVENT\_DATA\_PTR /\* pointer to the EVENT\_DATA entry\*/  
+ DESTINATION /\* to whom the event is destined \*/  
+ {SV\_ID} /\* the assets involved in the event\*/  
+ FINAL\_EVENT\_PTR /\* a pointer to the EVENT\_DESC of the  
final event of an extended event\*/  
+ DO\_NOW /\* flag indicating if conflict is  
already done and the event can be  
executed without going through the  
conflict handler\*/  
+ MODE /\* the mode in which the event is to  
be calculated \*/

ORGANIZATION (if file):

NOTES: The maximum number of SV\_IDS is 2 (two).

---

NAME: EVENT\_Q

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a time-ordered series of events used to trigger activities in the simulation.

COMPOSITION: EVENT

ORGANIZATION (if file):

ORGANIZATION (if file): The file consists of a data section in arbitrary order and a compact time-ordered list pointing to entries in the data section.

NOTES:

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: FIRST\_TRANSMIT

ALIASES

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a logical variable denoting first communication transmission.

COMPOSITION: TRUE | FALSE

ORGANIZATION: (if file):

NOTES:

---

NAME: GEOGRAPHY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a map of the world defined to the extent needed for the simulation.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: GLOBAL\_SIMULATION\_DATA

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION:

COMPOSITION:

ORGANIZATION (if file):

NOTES: Is this identical or related to SIM\_STATUS or SIM\_DATA?

---

NAME: GRID\_QUANTITY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

**DESCRIPTION:** This is a quantity such as electromagnetic environment or nonlocal weather that is stored in DETEC in a discretized fashion, the GRID being the basis for discretization.

**COMPOSITION:**

**ORGANIZATION (if file):**

**NOTES:**

---

**NAME:** INCIDENT\_PACKET

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** This is the communication packet that arrived at the receiver. It is the TRANSMITTED\_PACKET modified by the path transfer function. For a perfect communication path, it is identical to the TRANSMITTED\_PACKET.

**COMPOSITION:** PACKET

**ORGANIZATION (if file):**

**NOTES:**

---

**NAME:** INFO\_DATA

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** These are the data accompanying an information packet. There probably will be many different formats for these data.

**COMPOSITION:**

**ORGANIZATION (if file):**

**NOTES:** This will include routing information if necessary.

---

**NAME:** INFO\_PACKETS

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** INFO\_PACKETS is a file containing messages that are transmitted

## DATA DEFINITIONS

## DATA DEFINITIONS

from asset to asset.

COMPOSITION: {PACKET}

ORGANIZATION (if file):

NOTES:

---

NAME: INFO\_PTR

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the address of INFO.DATA that is part of a packet.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: IN\_PROGRESS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a list of asset identifications for which an extended length simulation is currently being done. It is possible for a single asset to be listed several times.

COMPOSITION:

{SV\_ID + FINAL\_EVENT\_PTR} /\* location of final  
EVENT\_PTR in EVENT\_Q for an extended event \*/

ORGANIZATION (if file):

NOTES:

---

NAME: INTENDED\_PACKET

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the correct communication packet that the sender

## DATA DEFINITIONS

## DATA DEFINITIONS

intended to transmit.

COMPOSITION: PACKET

ORGANIZATION (if file):

NOTES:

---

NAME: INTERACTION\_DELIMITER\_TYPE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: Part of EVENT\_DATA, this parameter indicates which type of effect an interaction delimitation is to be performed.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: INTERACTION\_TYPE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: Part of EVENT\_DATA, this parameter indicates which type (effect-on-asset) of interaction is to be calculated.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: INTERRUPT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a control flag, returned by the executive to the

## DATA DEFINITIONS

## DATA DEFINITIONS

manager, that indicates the reason the executive is stopping or suspending.

COMPOSITION: INTERVENE | COMPLETE | FATAL\_ERROR | NO\_INTR

INTERVENE - the user has requested an intervention so that a change to the course of the simulation can be made.

COMPLETE - the simulation has completed.

FATAL\_ERROR - the executive has detected an error and cannot continue.

NO\_INTR - No interrupt has occurred.

ORGANIZATION (if file):

NOTES:

---

NAME: INTERVENTIONS

ALIASES:

TYPE: DATA\_ELEMENT | DATA\_FLOW | FILE

DESCRIPTION: These are events or actions that interrupt, change, and override SCENARIO conditions only.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: MENU

ALIASES:

TYPE: DATA\_ELEMENT | DATA\_FLOW | FILE

DESCRIPTION: This is the set of selectable options that are displayed for the user.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: MN\_AGAIN

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a flag indicating whether MOTHER\_NATURE modules are to be called again to continue their calculation.

COMPOSITION: TRUE | FALSE

ORGANIZATION (if file):

NOTES:

---

NAME: MN\_EVENT\_PARAMETERS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a file that contains parameters describing all EVENT requests from MOTHER\_NATURE submodules. It is used to construct one or more EVENTS to exercise some or all of the MOTHER\_NATURE submodules.

COMPOSITION: {SV\_CLASS\_ID + NEXT.TIME}

ORGANIZATION (if file):

NOTES:

---

NAME: MN\_GEOGRAPHY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a temporary file that stores the GEOGRAPHY information needed by the Grid Quantity Calculation modules.

COMPOSITION:

ORGANIZATION (if file):

## DATA DEFINITIONS

## DATA DEFINITIONS

NOTES:

---

NAME: MN\_GEOGRAPHY\_REQUEST

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This identifies the type and extent of GEOGRAPHY information needed by a particular GRID\_QUANTITY calculator, all done in MOTHER\_NATURE.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: MN\_SOURCES

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a temporary file that stores the source STATE\_VECTORS needed by the GRID\_QUANTITY calculation modules.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: MODE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: Mode defines the particular conflict resolution in which a module is supposed to execute in.

COMPOSITION: INSTANT | FIRST\_STEP | STEP | LAST\_STEP | PARTIAL

INSTANT - execute the instantaneous module

FIRST\_STEP - put an event in the EVENT\_Q to start the first step

## DATA DEFINITIONS

## DATA DEFINITIONS

of a conflict resolution. A step event is generated but no calculations are done.

STEP - execute the module from the last time run to the present time and put an event in the EVENT\_Q to initiate the next step of the conflict resolution.

LAST\_STEP - complete execution of the extended event and do not put a step event in the queue.

PARTIAL - execute the module from the last time run to the present time. No step event is generated.

ORGANIZATION (if file):

NOTES:

---

NAME: MODULE\_LIBRARY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the various versions of the code modules as maintained by the SOURCE CODE CONTROL SYSTEM. The user may select the modules appropriate to a given simulation.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: MODULE\_NAME

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the standard format text module name. This is the FORTRAN name.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

## DATA DEFINITIONS

## DATA DEFINITIONS

---

NAME: MSG.CODE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a message code in text that can be used to further identify the message or warning or error.

COMPOSITION:

ORGANIZATION (if file):

NOTES: This can be used as an index by a program or as a reference to find a more detailed explanation of an error or message.

---

NAME: MSG.TEXT

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a standard format text string with human-readable message.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: NEXT.TIME

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the next time at which some specified function is to be done.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: NEW\_SOURCE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a logical variable denoting presence of new interference sources that can affect a communication.

COMPOSITION: TRUE | FALSE

ORGANIZATION: (if file):

NOTES:

---

NAME: OBJECT\_TYPE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: Part of EVENT\_DATA, this parameter denotes which type of object is to be generated by GENERATE\_OBJECT.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: ORDER\_OF\_BATTLE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the initial order of battle including equipment and forces.

COMPOSITION:

ORGANIZATION (if file):

---

NAME: ORIENTATION

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

DESCRIPTION: The orientation of an object deformed by, for example, Euler angle.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: OUTPUT\_LOG

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a text file containing periodic code and simulation information and all error and warning messages. This will have all user commands and messages that appear on the DISPLAYS data definitions. All errors, warnings, and actions taken will be included in a time-ordered manner. Also included in this description might be a set of files - one for each side in a battle simulation and one for the overall simulation run.

COMPOSITION: {ERROR\_MSG} + {WARNING\_MSG} + {DISPLAYS}

ORGANIZATION (if file):

NOTES: All processes write into this file as required. It in general is not shown on the data flow diagrams because it makes them too complicated.

---

NAME: PACKET (generic)

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the generic form of the messages that are transmitted from asset to asset.

COMPOSITION: INFO\_PTR + XMIT\_ID + RCVR\_ID + INFO\_DATA

ORGANIZATION (if file):

NOTES: There will be many different formats of the data fields for the various assets.  
XMIT\_ID and RCVR\_ID are the SV\_ID's for the transmitter and receiver, respectively. See SV\_ID for more information.

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: PATH\_FUNCTIONS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: Path functions are time- and frequency-dependent functions that describe the transformations of TRANSMITTED\_PACKET(s) into INCIDENT\_PACKET(s).

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: PERCEIVED\_ATTRIBUTES

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are parameters "measured" by a sensor for each object detected.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: PERCEIVED\_OBJECT\_ATTRIBUTES

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the local file containing the results of a SENSOR's observations. The information contained in it is processed to construct an INFO\_PACKET.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

## DATA DEFINITIONS

## DATA DEFINITIONS

**NAME:** PERCEIVED\_WORLDS

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** This is the perception of the world as seen by the combatants and the assets belonging to the combatants.

**COMPOSITION:** BATTLE\_ORDER +  
#combatants #assets  
{ { ( { STATE\_VECTOR } )  
+ ( { RECORDED\_OBSERVATION } )  
+ ( { RECORDED\_DECISION } )  
+ ( { ATTACK\_PLAN } )  
+ ( { PLANNED\_RESPONSE } )  
+ ( { BATTLE\_MGR\_DIRECTIVE } ) } } }  
+ ENVIRONMENT

**ORGANIZATION (if file):** The boundaries in this file are absolute. For example, no asset can look directly at another asset's PERCEIVED\_WORLDS. Spies are an exception to this.

**NOTES:**

-----

**NAME:** PHYSICAL\_OUTPUT

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** This is the hardcopy (e.g., film, paper, etc.) output available to the user.

**COMPOSITION:** DISPLAYS

**ORGANIZATION (if file):**

**NOTES:**

-----

**NAME:** PLANNED\_RESPONSE

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** This is the pre-planned response of a combattant to an attack.

## DATA DEFINITIONS

## DATA DEFINITIONS

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: POSITION

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the location of an object.

COMPOSITION:

ORGANIZATION (if file):

NOTES: It may be necessary to carry position in multiple coordinate systems to allow more efficient calculations.

---

NAME: POST\_PROCESSING

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the raw data collected during a simulation run that may be reduced and analyzed following the run.

COMPOSITION:

{REAL\_WORLD} + {PERCEIVED\_WORLDS} + {SCENARIO}  
+ {EVENT\_Q} + {CODE\_DATA} + {SIM\_DATA}  
+ {INFO\_PACKETS}

ORGANIZATION (if file):

---

NAME: PREV\_TIME

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the time at which the previous simulation calculation or activity took place for an event.

COMPOSITION:

## DATA DEFINITIONS

## DATA DEFINITIONS

ORGANIZATION (if file):

NOTES:

---

NAME: REAL\_WORLD

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the world as it actually and precisely exists. There are no errors in the real world.

COMPOSITION:

{STATE\_VECTOR} /\* for all assets \*/  
+ ENVIRONMENT + CONSTANTS /\* e.g., pi, speed of light, etc. \*/  
+ GEOGRAPHY

ORGANIZATION (if file):

NOTES:

---

NAME: RECEIVED\_PACKET

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a communication packet that is delivered by the receiver. It is the INCIDENT\_PACKET modified by the receiver transfer function. For a perfect receiver, it is identical to the INCIDENT\_PACKET.

COMPOSITION: PACKET

ORGANIZATION (if file):

NOTES:

---

NAME: RECEIVER\_ID

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This data element identifies the particular communications receiver to which a packet is sent.

## DATA DEFINITIONS

## DATA DEFINITIONS

COMPOSITION:

ORGANIZATION (if file):

NOTES: This is a SV\_ID

---

NAME: RECORDED\_DECISION

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a recorded decision made by some particular battle manager.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: RECORDED\_OBSERVATION

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a recorded perception of some particular sensor.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: REF\_LIBRARY

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: These are the various versions of the DETEC data base as maintained by the SOURCE CODE CONTROL SYSTEM. The data base consists of physical constants, geography, asset descriptions, etc...

COMPOSITION:

## DATA DEFINITIONS

## DATA DEFINITIONS

ORGANIZATION (if file):

NOTES:

---

NAME: RESTART\_DUMPS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a file containing all data necessary to resume a simulation run as if it had not been interrupted.

COMPOSITION:

{ {SCENARIO} + {REAL\_WORLD} + {PERCEIVED\_WORLDS}  
+ {EVENT\_Q} + {INFO\_PACKETS} + {CODE\_DATA}  
+ {IN\_PROGRESS} + {PATH\_FUNCTIONS} }

ORGANIZATION (if file):

NOTES:

---

NAME: SAVE\_SV

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a flag which indicates whether a state vector must be saved if changed.

COMPOSITION: TRUE | FALSE

TRUE - save state vector  
FALSE - do not save state vector

ORGANIZATION (if file):

NOTES:

---

NAME: SCENARIO

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

## DATA DEFINITIONS

## DATA DEFINITIONS

**DESCRIPTION:** This is the predefined series of actions, events, and constraints that drive and control the simulation.

**COMPOSITION:**

```
#combatants
{ (ATTACK_PLAN) + (PLANNED_RESPONSE) }
+ (INTERVENTIONS) /* Events and changes which are used to
trigger natural and omnipotent actions */
+ (ORDER_OF_BATTLE)
```

**ORGANIZATION (if file):**

**NOTES:** This defines intent, of the plans, combatant assets and status, times of launch, and battle-manager instructions and constraints. Interventions are sometimes referred to as acts of God.

---

**NAME:** SCENARIO\_LIBRARY

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** These are the various scenarios that are saved and maintained by the SOURCE CODE CONTROL SYSTEM.

**COMPOSITION:**

**ORGANIZATION (if file):**

**NOTES:**

---

**NAME:** SENSOR\_OPERATING\_PARAMETERS

**ALIASES:**

**TYPE:** DATA ELEMENT | DATA FLOW | FILE

**DESCRIPTION:** This is information describing parameters such as threshold and frequency range to be used for the SENSOR operations. It is extracted from EVENT\_DATA.

**COMPOSITION:**

**ORGANIZATION (if file):**

**NOTES:**

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: SIM\_DATA

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This information gives the state or status of the simulated battle(s).

COMPOSITION: This is the binary machine representation of data the user had requested for processing into the POST\_PROCESSING and RESTART\_DUMPS files.

ORGANIZATION (if file):

NOTES: This is very closely related to SIM\_STATUS in that the user has requested to log simulation data here in a machine form where SIM\_STATUS is in a text user-readable form.

---

NAME: SIM\_STATUS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the status of the simulated battle(s) as displayed for the user.

COMPOSITION: This is text representation of simulation data that the user has requested to view either in DISPLAYS and/or OUTPUT\_LOG

ORGANIZATION (if file):

NOTES: See SIM\_DATA for clarification of the differences between SIM\_STATUS and SIM\_DATA.

---

NAME: SOURCE

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This identifies the module that originated the message, data, or function.

COMPOSITION:

ORGANIZATION (if file):

## DATA DEFINITIONS

## DATA DEFINITIONS

### NOTES:

---

NAME: STATE\_VECTOR (generic)

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the standard definition of an object's position, velocity, status, and all other existing parameters.

### COMPOSITION:

SV\_ID + POSITION + ORIENTATION + SV\_CLASS\_ID  
+ DAMAGE + SV.... + SAVE\_SV

ORGANIZATION (if file):

### NOTES:

---

NAME: SV\_CLASS\_ID

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is an alphanumeric identifier for each class of STATE\_VECTORS. A class may be defined as all STATE\_VECTORS that follow the same algorithm for normal (undisturbed) time evolution but may have different values for parameter used in the algorithm. These parameters are part of each STATE\_VECTOR.

### COMPOSITION:

ORGANIZATION (if file):

### NOTES:

---

NAME: SV\_ID

### ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a unique identification for a state vector.

### COMPOSITION:

## DATA DEFINITIONS

## DATA DEFINITIONS

ORGANIZATION (if file):

NOTES:

---

NAME: SYSTEM\_TABLES

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a file accessed to change or retrieve the status of the DETEC code within the CTSS system. This will be implemented by library calls to the appropriate system function.

COMPOSITION: System parameters (to be defined as required)

ORGANIZATION (if file):

NOTES:

---

NAME: TEXT\_MSG

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the standard format text message designed for human understanding.

COMPOSITION: DATE.TIME + MODULE\_NAME + MSG\_CODE + MSG\_TEXT

ORGANIZATION (if file):

NOTES:

---

NAME: TIME

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the binary time as maintained by the simulation program.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

## DATA DEFINITIONS

## DATA DEFINITIONS

---

NAME: TRANS\_CHARACTERISTICS

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This data flow describes the characteristics of a communications transmitter, such as frequency band, power distribution function, power level, etc.

COMPOSITION:

ORGANIZATION (if file):

NOTES:

---

NAME: TRANSMITTED\_PACKET

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is the communications packet that was actually transmitted. It is the INTENDED\_PACKET modified by transmitter transfer function. For a perfect transmitter, it is identical to the INTENDED\_PACKET.

COMPOSITION: PACKET

ORGANIZATION (if file):

NOTES:

---

NAME: TRANSMITTER\_ID

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This data element identifies the particular communications transmitter that sends a packet.

COMPOSITION:

ORGANIZATION (if file):

NOTES: This is a SV\_ID

---

## DATA DEFINITIONS

## DATA DEFINITIONS

NAME: VIEWABLE\_OBJECT\_LIST

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is a file containing IDs of the STATE\_VECTOR of all objects determined to be viewable by the Filter.

COMPOSITION: {SV.ID}

ORGANIZATION (if file):

NOTES:

---

NAME: WARNING\_MSG

ALIASES:

TYPE: DATA ELEMENT | DATA FLOW | FILE

DESCRIPTION: This is an information message describing a possibly abnormal condition.

COMPOSITION: TEXT\_MSG

ORGANIZATION (if file):

NOTES:

## DATA DEFINITIONS

## **9. APPENDIX**

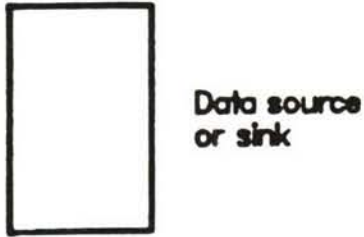
- A. STRUCTURED ANALYSIS CONVENTIONS**
- B. NAMING CONVENTIONS**
- C. ALTERNATIVE PROPOSALS**

# APPENDIX A. STRUCTURED ANALYSIS CONVENTIONS

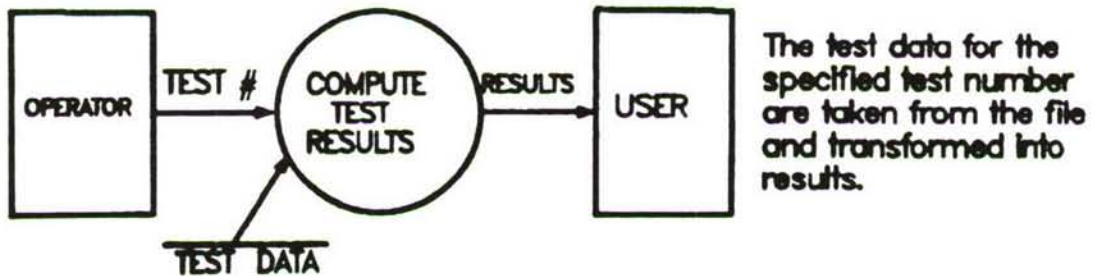
Reference: Structured Analysis and System Specification

by Tom De Marco

## Data Flow Notation:



## Data Flow Example:



## APPENDIX A

### Data Dictionary Notation:

- = means "is equivalent to"
- + means "and"
- [ ] means "either-or" i.e., select one of the options
- { } means "iterations of" the component enclosed
- ( ) means the enclosed component is "optional"
- | means "or"

### Data Dictionary Examples:

COMMAND = [START | STOP | ABORT]

- select one of the options

DATA-BASE = {SHOT-DATA + SHOT-NUMBER +  
(SUBSYSTEM-ID)}

-the data base consists of multiple sets of shot data and shot number and an optional subsystem identification.

## APPENDIX B. NAMING CONVENTIONS

BM - battle-manager process  
CM - communications process  
CMD - command  
EG - engagement  
ENV - environment  
EP - event physics process  
EQ - event queue file  
EV - event  
EX - executive process  
IN - in progress file  
INTEL - intelligence  
IP - information packet  
LG - logger process  
MN - mother nature process  
MR - manager process  
PF - path functions file  
PP - postprocessing file  
PTR - pointer  
PW - perceived world file  
Q - queue  
RD - restart dumps file  
RW - real-world file  
SC - scenario file  
SN - sensor process  
SV - state vector  
WP - weapons process

## APPENDIX B

## APPENDIX C. ALTERNATIVE PROPOSALS

This is an alternative proposal for the Executive conflict handler.

```
1 -----
2 CONHAN - CONFLICT HANDLER
3 -----
4 DEFINITIONS -
5   TI = Initial time for CONHAN search
6   TH = Time horizon for CONHAN search
7   TB = Begin time for extended event
8   TE = End time for extended event
9   TET = Temporary end time for extended event
10  IC = Check time
11  ICP = Previous check time
12  TM = First mismatch time in EVENT_Q comparison
13  CHK = Check event
14  SCH = Search event
15  ITP = Interval type flag
16  Interval types are:
17     AF = Asset Function (ITP=AF)
18     EP = Event Physics (ITP=EP)
19  LBL = Interval label
20  Ai = Set of labels for EP intervals that overlap AF
21     intervals, for asset i.
22  Bi = Set of labels for responsive EP intervals,
23     for asset i.
24  Responsiveness of an EP interval implies that EP acting
25  upon asset i has the potential of creating new events
26  within the EP interval.
27
28  FUNCTION - This module searches the interval [TI,TH] in the
29  EVENT_Q for conflict events and establishes conflict
30  process control by setting state vector flags and spawning
31  new events. This is accomplished by using set theory.
32  -----
33  CONHAN (EVENT_PTR)
34  /* Compare EVENT_Q with EVENT_Q_OLD over [ICP,IC] */
35  IF (CHK AND EVENT_Q = EVENT_Q_OLD), THEN
36    ICP = IC
37  ELSE
38    IF (SCH), THEN
39      TI = TH
40      ICP = TI
41      CALCULATE NEXT TIME HORIZON (TH)
42      INSERT SCH IN EVENT_Q
43      INSERT CHK(S) IN EVENT_Q
44    ELSE
45      FIND TM
```

## APPENDIX C

## APPENDIX C. ALTERNATIVE PROPOSALS

```
46     TI = TM
47     END IF
48     WRITE EVENT.Q FOR [TI,TH] IN EVENT.Q.OLD
49     SEARCH [TI,TH] FOR EVENTS THAT HAVE ASSETS IN COMMON
50     /* Scan to horizon */
51     DO FOR EACH ASSET (i)
52         TET = TE
53         IF (TH < TE), THEN
54             TET = TH
55         END IF
56         COMPUTE TIME INTERVAL [TB,TET]
57         ASSIGN INTERVAL TYPE AND LABEL (ITP,LBL)
58         CHECK FOR OVERLAP OF TIME INTERVALS AND LOAD A1
59         CHECK COUPLING MATRIX FOR RESPONSIVENESS AND LOAD B1
60         IF (A1 OR B1), THEN
61             FURTHER INTERVAL TESTS /* May not be necessary! */
62             SET CONFLICT FLAGS IN EVENT.DATA (ITP)
63         END IF
64     END DO
65     END IF
66     RETURN
67
```

## APPENDIX C

Printed in the United States of America  
Available from  
National Technical Information Service  
US Department of Commerce  
5285 Port Royal Road  
Springfield, VA 22161

Microfiche (A01)

NTIS		NTIS		NTIS		NTIS	
Page Range	Price Code	Page Range	Price Code	Page Range	Price Code	Page Range	Price Code
001-025	A02	151-175	A08	301-325	A14	451-475	A20
026-050	A03	176-200	A09	326-350	A15	476-500	A21
051-075	A04	201-225	A10	351-375	A16	501-525	A22
076-100	A05	226-250	A11	376-400	A17	526-550	A23
101-125	A06	251-275	A12	401-425	A18	551-575	A24
126-150	A07	276-300	A13	426-450	A19	576-600	A25
						601-up*	A99

\*Contact NTIS for a price quote.

Los Alamos