*ISI Research Report*
*ISI/RR-89-225*
*July 1989*

(4)


Susan Coatney
David Mizell

# ISI's SDI Architecture Simulator: Initial Prototype User Interface






*INFORMATION SCIENCES INSTITUTE*

*213/822-1511*
*4676 Admiralty Way/Marina del Rey/Californ. 90292-6695*

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION / AVAILABILITY OF REPORT<br>This document is approved for public release; distribution is unlimited. |
| 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>ISI/RR-89-225 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>------- |
| 6a. NAME OF PERFORMING ORGANIZATION<br>USC/Information Sciences Institute<br>6b. OFFICE SYMBOL (*If applicable*) | 7a. NAME OF MONITORING ORGANIZATION<br>Office of Naval Research |
| 6c. ADDRESS (*City, State, and ZIP Code*)<br>4676 Admiralty Way<br>Marina del Rey, CA 90292-6695 | 7b. ADDRESS (*City, State, and ZIP Code*)<br>800 N. Quincy Street<br>Arlington, VA 22217 |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION<br>SDIO<br>8b. OFFICE SYMBOL (*If applicable*) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>N00014-87-K-0022 |
| 8c. ADDRESS (*City, State, and ZIP Code*)<br>Strategic Defense Initiative Organization<br>Office of the Secretary of Defense<br>The Pentagon, Washington, DC 20301 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. ------- / PROJECT NO. ------- / TASK NO. ------- / WORK UNIT ACCESSION NO. ------- |
| 11. TITLE (*Include Security Classification*)<br>ISI's SDI Architecture Simulator: Initial Prototype User Interface (Unclassified) | |
| 12. PERSONAL AUTHOR(S)<br>Coatney, Susan; Mizell, David | |
| 13a. TYPE OF REPORT<br>Research Report | 13b. TIME COVERED<br>FROM _____ TO _____ |
| 14. DATE OF REPORT (*Year, Month, Day*)<br>1989, July | 15. PAGE COUNT<br>20 |
| 16. SUPPLEMENTARY NOTATION | |

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| 09 | 02 | |
| | | |

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
discrete-event simulation, modular software design, object-oriented programming, SDI architecture, software engineering

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This report describes the software design of the initial prototype user interface for the strategic defense architecture simulation system at ISI. The user interface was designed to satisfy several requirements. First, it should provide a means for users to easily specify new simulation parameters, including candidate defense system architectures and threat scenarios. Second, it should not be necessary for users to be familiar with the internal workings of the simulator in order to execute the simulator and examine the simulation results. The report documents the design and implementation of a user interface that meets these requirements. It includes a general description of the user interface and its software components, an example interactive session to demonstrate use of the system, and some suggestions for future improvements to the software.

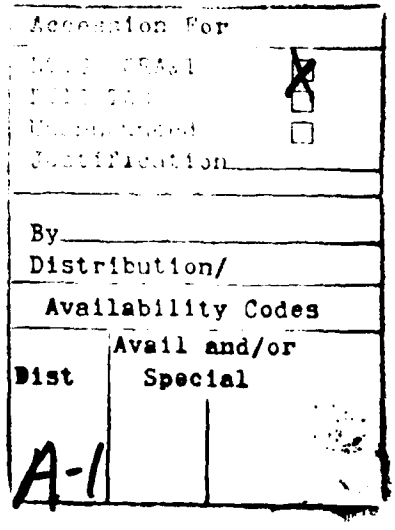
| | |
|---|---|
| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT<br>☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Victor Brown Sheila Coyazo | 22b. TELEPHONE (*Include Area Code*)<br>213/822-1511<br>22c. OFFICE SYMBOL |

*ISI Research Report*
*ISI/RR-89-225*
*July 1989*

*University of Southern California*

Susan Coatney
David Mizell

# ISI's SDI Architecture Simulator: Initial Prototype User Interface




*INFORMATION SCIENCES INSTITUTE*

*213/822-1511*
*4676 Admiralty Way/Marina del Rey/California 90292-6695*

This research is supported by the Strategic Defense Initiative Office under Office of Naval Research Contract No. N00014-87-K-0022. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of the Strategic Defense Initiative Office, the Office of Naval Research, or the U. S. Government. No official Government endorsement should be inferred.

# 1. Introduction

In 1987, the ISI parallel and distributed computing research group implemented a prototype sequential, discrete-event simulator of SDI architectures. An important design goal of this prototype simulation system software development effort was to achieve a clean separation between the candidate being modeled and the "outside world" -- that is, between the simulated defense architecture and the simulated characteristics and effects of the physical environment and the threat. Additionally, our simulator design was intended to incorporate an abstract yet executable representation of the battle management system of the candidate architecture being modeled. These two central design goals followed directly from our perception that SDIO needs a simulation system that (a) is independent of any particular candidate architecture (i.e., it is capable of simulating many different candidates under the same set of assumptions about environment and threat) and (b) includes a nontrivial representation of the computation and decision-making performed by the battle management system before and during an attack. A discussion of our design approach to the simulation software and a detailed description of the resulting system are provided in [2].

It is implicit in these assumptions about the requirements of the simulation system that we expect the simulator to be used by many different people and for simulating many different candidate architectures. With these future simulator users in mind, we have developed a prototype interactive user interface. The purpose of this document is to report on the results of this prototype development, as well as to serve as a rudimentary manual for simulator users. Section 2 provides a general description of the user interface and its software components, Section 3 walks the reader through a example interactive session in which a user sets up and runs a simulation, and Section 4 contains our suggestions of future improvements to the user interface software.

# 2. Overview of the User Interface

We used the same development philosophy in implementing the prototype user interface that we used when implementing the simulator itself: keep the design as simple as possible without avoiding any difficult design issues. Our assumption throughout this project has been that we were *not* developing a fully functional, fully debugged, complete software product, but instead were constructing a design framework in such a way that the difficult, fundamental design problems were addressed, but the details of a full, "productized" implementation could be added later without additional fundamental design problems being encountered. Thus, our user interface is simple and primitive.

The user interface software consists of several components: the high-level BMA language translator, an interactive session manager, an output filtering mechanism, a database for

storage and retrieval of simulation output, and a facility for graphic display of simulation results (see Fig. 1). The database system and graphics package are described in [2], and are therefore not discussed here.

## 2.1 High-level BMA Language Translator

"BMA" is an abbreviation of "battle manager abstraction." This is our term for the high-level but executable programs that model the computations of the defense architecture's battle management/C3 system. A BMA can be specified by the user for any platform or set of platforms in the defense architecture being simulated. A companion report [3] provides a detailed description of "kmac," the high-level language we developed for specifying a BMA. The translator for this language converts a high-level BMA specification to C++ code, which must then be compiled and linked with the main body of the simulator. The session manager simplifies this process by automatically invoking the translator and recompiling the simulator when necessary.

## 2.2 Session Manager

The session manager is a menu-driven, interactive interface to the simulator. It is implemented in C++ and runs on a VAX 8650 under 4.3 BSD UNIX. Its design attempts to meet the following criteria: (1) the user should be able to easily specify new simulation parameters, including new candidate defense system architectures and threat scenarios; and (2) the user should not be required to be familiar with the internal workings of the simulator.

To allow new simulation parameters to be easily defined, menus are provided that prompt the user to select or specify the following:

1. the platforms and configuration of the defense architecture, including the platform types, the platform capabilities, and the platform deployment sequence.
2. the high-level language representations of the BMAs.
3. the nature, magnitude and timing of the threat.
4. the textual or graphic output desired from the simulation run.
5. miscellaneous parameters, such as the starting seed for the random number generator, and the screen update rate.

The session manager also maintains a database of the current definitions of the simulation parameters for each user. This database is consulted on each invocation of the program and is updated on exit. Thus, a session can be interrupted and easily restarted.

In order to mask the internal structure of the simulator from the user, the session manager transparently coordinates the interaction between the user and the other components of the simulation system (see Figure 2). It automates the process of translating the high-level BMA specifications and incorporating the resulting code into the simulator; any necessary auxiliary files needed by the simulator are also automatically generated from user input.

The structure of the session manager is very simple: it consists of a set of C++ functions that manage the menus and process user requests, passing control to the other units in the user interface when necessary. For instance, when given a new BMA specification, the program automatically invokes the translator, generates several BMA-specific simulation system files, and invokes a shell script that recompiles the simulator, producing a new executable module. When the object file is executed, control is passed to the simulator, which causes some output files to be generated by the output filter mechanism. When the simulation terminates, control is returned to the session manager. Data is passed between the units primarily in the form of arguments specifying the names of input files that are created by the session manager and contain the information specified by the user.

During implementation of the session manager, we found that one additional feature was needed: a locking mechanism that restricts the concurrent use of the front-end to a single user. This is to prevent multiple users from simultaneously overwriting the system files created by the program when new platform types are defined. The implementation is based on the use of a file as a semaphore. When the program is invoked, it attempts to create a file in a predesignated directory. If the file already exists, the creation attempt will fail and the program will exit; otherwise execution will proceed normally and the file will be removed upon exit, allowing the next user to access the program.

Section 2 provides a detailed description of a sample session illustrating the specification of both a defense system architecture and a threat configuration.

## 2.3 Output Filtering Mechanism

In order to obtain meaningful results from a simulation run, there must be some means of filtering the voluminous output from the simulator. One method of doing this is to conceive of the simulator output as a single stream of data that can be passed through a series of filters. Each filter collects some pertinent information from the data stream and, possibly, removes some data from it before passing it on to the next filter. By associating a data transformation function with each filter, the collected data can be summarized into statistical output as specified by the user. Implementing the filters as separate processes executing in parallel with the simulator would allow the computation to proceed in a pipelined fashion, resulting in greater effective utilization of computer system resources.

We have implemented a rudimentary form of this concept. The user specifies which simulated events are of interest; the selected events are collected and saved in output files when the simulator is executed.

## 3. Example User Scenario

The user interface to the simulator is an interactive program designed to allow the user to (1) specify a new execution environment for the simulator; (2) recompile the simulator if necessary; and (3) run the simulator with the newly defined parameters.

In the following example of an interactive session, three types of text appear: user input, explanatory text, and program output. User input is printed in **boldface** and may be accompanied by explanatory text; the resulting program output is printed in `typewriter font`. The symbol ↵ represents a point at which the user has struck the carriage return key.

**frontend** ↵ Start up the program; the root menu will be displayed.

```
================================================================================
                    *** OPTIONS ***

                    P    Platform type definition
                    I    Input/output file definition
                    M    Miscellaneous parameters
                    S    Show the current bindings
                    C    Compile
                    E    Execute the simulator
                    H    Help message
                    Q    Quit


                    ENTER YOUR CHOICE:  S↵
================================================================================
```

## 3.1 Current Context Display

The program is structured as a hierarchy of menus; the user selects a choice by entering the appropriate character. The current simulation parameters are stored in a file named "$USER.context" (with the actual user's name substituted for $USER). This file is read upon each invocation of the program; it contains the current bindings for the platform type definitions, the names of the simulator input files, and other simulation parameters. The context file is updated with the new bindings upon exit from the program. Selecting option "S" causes the current context to be displayed.

```
================================================================================
******* PLATFORM TYPES:
          No platform types have been defined.

******* INPUT FILES:
          Platform deployment:  UNDEFINED
          Threat:               /u3/mizell/fysicium/src/frontend/threat
          Capabilities:         UNDEFINED
          Filters:              /u3/mizell/fysicium/src/frontend/flags
          Output:               coatney
          BMA header:           UNDEFINED

******* OTHER PARAMETERS:
          Screen Update Rate:        5
          Seed:                      1


     .... Type <CR> to continue: ↵
================================================================================
```

Initially, the user is assigned a default context. Items that are marked UNDEFINED must be defined before the simulator can be executed. Each of the items in the context will be described more fully as the session proceeds. Typing a carriage return (CR) returns us to the root menu, where we select option "P" to define some platform types.

## 3.2 Platform Type Definition

```
P↵
================================================================================
          *** CURRENT PLATFORM TYPES ***
          No platform types have been defined.
          BMA header:              UNDEFINED

          *** OPTIONS ***
          B   Define a BMA
          H   Specify the BMA Header File
          A   Abort this session
          Q   Quit this menu


          ENTER YOUR CHOICE: B↵
================================================================================
```

This display contains two sections -- a description of the currently defined platform types, and a menu of options. A platform type is characterized by its name, which is chosen by the user and must be unique, and the name of its BMA. Selecting option "B" allows the user to define a new platform type. The user is then asked to enter the name of the platform type and the name of the file containing the high-level source code for the

BMA (see [3]). The program then invokes the kmac parser to translate the BMA specification to C++ code. If successful, the resulting code is saved and later compiled with the simulator system code.

```
================================================================================

                    Name of the new platform type?          dummy1↵

                    Name of the BMA source file?            dummy1.k↵

                    Source file successfully parsed.

                    .... Type <CR> to continue:↵
================================================================================
```

After defining platform type dummy2 (not shown), we select option "H" to define the name of the BMA header file. This header file should contain all definitions of user-defined types that appear in the definition of BMA persistent variables. If no header file is needed (i.e., only base types or simulator system defined types are used for persistent variable definitions) this parameter should be set to UNDEFINED.

```
================================================================================

                    Set the include file to UNDEFINED? (Y/N)    n↵

                    Name of the include file:               dummybma.h↵

                    .... Type <CR> to continue:↵
================================================================================
```

## 3.3 Input/Output File Definition

We now return to the root menu by quitting the current menu, and select the input/output file definition menu. Information is passed between the session manager and the simulator in the form of arguments that specify the names of input files created by the session manager and that contain the information supplied by the user. This menu allows the user to define the contents of those files, which include descriptions of the platform deployment, the platform capabilities, the threat scenario, the output filters, the basename for the output files created by the simulator, and the BMA header file. The current input file names are shown in the upper portion of the display.

```
Q↵
I↵
==========================================================================
        *** CURRENT INPUT/OUTPUT FILES ***
        Platform Deployment:        UNDEFINED
        Threat:                     /u3/mizell/fysicium/src/frontend/threat
        Capabilities:               UNDEFINED
        Filters:                    /u3/mizell/fysicium/src/frontend/flags
        Output:                     coatney
        BMA Header:                 /u3/mizell/fysicium/src/frontend/dummybma.h

        *** OPTIONS ***
        D   Platform Deployment
        C   Platform Capabilities
        T   Threat Scenario
        F   Output Filters
        O   Output file basename
        A   Abort this session
        Q   Quit this menu

        ENTER YOUR CHOICE: D↵
==========================================================================
```

## 3.3.1 Platform Deployment

We will first define the platform deployment. The user can either specify the name of an existing file (previously defined using this program) or define a new one. For each platform type, the user must designate the number of platforms of that type and the trajectories of those platforms. Currently, two types of trajectories have been implemented -- geostationary and walker (see [2] for details on trajectories).

```
==========================================================================
      Use an existing deployment file? (Y/N)                    n↵
      Enter the name of the deployment file:                    mydeploy↵

      *** How many platforms of type dummy1?                    1↵
      Trajectory type: <G> geostationary, <W> Walker orbits:    g↵
      Longitude of dummy1 platform 0:                           90↵

      *** How many platforms of type dummy2?                    3↵
      Trajectory type: <G> geostationary, <W> Walker orbits:    w↵
      Number of rings in Walker orbits:                         1↵
```

```
Latitude of the maximum coverage point:                    45↵
Longitude of the maximum coverage point:                   90↵

.... Type <CR> to continue: ↵
================================================================================
```

### 3.3.2 Platform Capabilities

We return to the input/output file menu, and select "C" to define the platform capabilities. Note that the user must know what capabilities to assign to each platform type. If the capabilities are not correctly assigned, the simulation may not execute successfully (see the suggestions for future improvements of the simulation system in Section 5).

The program maintains an internal listing of the currently implemented technology modules (see [1, 2]). For each such module, the user is asked to specify how many devices of that type to assign to the capabilities list of each currently defined platform type.

```
C↵
================================================================================
        Use an existing capabilities file? (Y/N)                   n↵
        Enter the capabilities filename:                           mycap↵
        **** Defining capabilities for platform type  [dummy1]:
        How many sensors of type STARING_SENSOR?                   1↵
        How many channels of type CHANNEL0?                        1↵
        How many weapons of type LASER?                            0↵
        **** Defining capabilities for platform type  [dummy2]:
        How many sensors of type STARING_SENSOR?                   0↵
        How many channels of type CHANNEL0?                        1↵
        How many weapons of type LASER?                            0↵

        .... Type <CR> to continue: ↵
================================================================================
```

### 3.3.3 Threat Scenario

Next we select option "T" to define the threat scenario. The current version of the user interface allows only a very limited specification of the threat scenario. The deployment sequence of the missiles consists of the number of missiles to launch, the launch time of

the first missile, and the interval between launches. The program arbitrarily assigns values for the initial position of the missiles and their targets.

```
T↵
================================================================================

                    Use an existing threat file? (Y/N)          n↵
                    Enter the threat filename:                  mythreat↵
                    Number of missiles:                         3↵
                    Time to begin launching missiles (seconds): 5↵
                    Interval between launches (seconds):        10↵

                    .... Type <CR> to continue: ↵
================================================================================
```

### 3.3.4 Output Filters

In order to obtain meaningful results from a simulation run, there must be some means of filtering the voluminous output from the simulator. One method of doing this is to conceive of the simulator output as a single stream of data that can be passed through a series of sieves. Each sieve collects some pertinent information and, possibly, removes some data from the stream before passing the stream on to the next sieve. The sieves could be implemented as separate processes executing in parallel with the simulator.

We have implemented a simple version of this concept. The user specifies which events are of interest; the specified events will be collected and saved in output files when the simulator is executed.

```
F↵
================================================================================

        Use an existing filters file?       (Y/N)    n↵
        Enter the filters file name:                 myfilters↵
        Print out event type SHOOT?         (Y/N)    n↵
        Print out event type DESTROY?       (Y/N)    n↵
        Print out event type BMA SENSE?     (Y/N)    n↵
        Print out event type SAMPLE?        (Y/N)    n↵
```

```
Print out event type SENSE APPEAR?     (Y/N)     y↵
          Print out relevant data ?    (Y/N)     y↵
Print out event type SENSE DISAPPEAR?  (Y/N)     y↵
          Print out relevant data ?    (Y/N)     n↵
Print out event type SENSE BLOWNUP?    (Y/N)     n↵
Print out event type LAUNCH?           (Y/N)     y↵
Print out event type NUKE?             (Y/N)     y↵
Print out event type BMA DELAY?        (Y/N)     y↵
Print out event type BMA COMM?         (Y/N)     y↵
Print out event type BMA INIT?         (Y/N)     y↵
Print out event type PRINT POSITION?   (Y/N)     y↵
          start time = 0↵
          end time   = 30↵
          delta time = 5↵

     .... Type <CR> to continue: ↵
================================================================================
```

The filtered output is written to files created by the simulator. The names of the files are created by concatenating a base name (by default the name of the current user) with the following set of strings:

*.err* (error file)

*.evnt* (filtered events)

*.in* (input parameters and file names)

*.msgs* (inter-platform messages)

*.pos* (platform and missile positions)

*.trg* (trajectory data)

*.db.evnt* (events in database format)

*.db.pos* (platform and missile positions in database format).

The basename of the files can be changed by selecting option "O".

```
O↵
================================================================================
        Enter the base name for the simulator output files: sim1

        .... Type <CR> to continue: ↵
================================================================================
```

## 3.4 Miscellaneous Parameters

Since all the input/output parameters have now been set, we return to the root menu and select option "M" to inspect the remaining miscellaneous simulation parameters -- the random number generator seed and the screen update rate.

The seed is passed to the simulator to serve as the seed for a random number stream. The default value of "1" should be changed by either directly specifying a new value for the seed or by requesting that the program assign a random value (at present, the value of the current time as returned by the UNIX system call *gettimeofday* is used as the random value).

The screen update rate is the rate (real time) at which the current simulation time is reported to the user while the simulation is proceeding; this is useful only for very long simulation runs, and is specified in units of seconds.

```
Q↵
M↵
================================================================================
        *** CURRENT SIMULATION PARAMETER VALUES ***
        Screen Update Rate:          5
                Seed:                1

                *** OPTIONS ***
                R   Random number generator seed
                S   Screen update rate
                A   Abort this session
                Q   Quit this menu

                ENTER YOUR CHOICE: Q↵
================================================================================
```

Setting the miscellaneous parameters is straightforward, and therefore will not be demonstrated.

## 3.5 Compiling the Simulator

We are ready to return to the root menu and compile the simulator. The program creates several system files, then attempts to compile and link the simulator by invoking the UNIX utility *make*. Diagnostic messages are printed.

C↵

```
================================================================================

     IS A FULL COMPILATION NEEDED? (if you don't know, type y): (Y/N) y↵
     Creating file [/u3/mizell/fysicium/include/init_plats.h]
     Creating file [/u3/mizell/fysicium/src/objects/init_plats.c]
     Creating file [/u3/mizell/fysicium/src/misc/bma.c]
     Creating file [/u3/mizell/fysicium/include/bma_pvars.h]
     set umask 000
     cd  /u3/mizell/fysicium/src

     ***** BEGIN COMPILING *****
     make -s NEWBMAH=/u3/mizell/fysicium/src/frontend/dummybma.h
     CC  objects.c:
     CC  init_plats.c:
     CC  bma.c:
     cc -p  simulator.o ../../lib/libS.a ../../lib/libS2.a -lm -o
            newsim -lC
     *** FINISHED COMPILING ***


     .... Type <CR> to continue: ↵
================================================================================
```

Usually a full compilation is necessary. In that case, the system generates four C++ source files that contain (1) declarations of BMA-related global variables; (2) definitions of functions used for initializing platform capabilities; (3) the translated BMA code; and (4) the declarations of BMA persistent variables. A full compilation does not imply that all of the simulator system code must be recompiled. On the contrary, only three source files require compilation. The resulting object code is then linked with the remaining system code to produce a new executable image. A partial compilation causes only the translated BMA code to be compiled.

There are several possible reasons for the compilation phase to fail. The BMA code may be incorrect, in which case the user should follow these steps:

1. edit the offending BMA source code
2. redefine the platform type (this will cause the source code to be translated into C++ code)
3. compile the simulator (partial compilation is sufficient).

If the source code is correct, the user should verify that all user-defined types used in the definition of persistent variables appear in the BMA header file, and make sure that the appropriate header file was specified. If any changes were necessary, recompile the simulator (full compilation).

## 3.6 Executing the Simulator

From the root menu, we select option "E" to execute the simulator. First the input parameters are printed, then any output from the BMAs. The references to New Zealand are a side effect of the enemy missiles successfully destroying their targets.

```
E↵
================================================================================
          USER = coatney
          DATE = Mon May  9 17:32:19 1988
          PLATFORM FILE = mydeploy
          THREAT FILE = mythreat
          CAPABILITY FILE = mycap

          *** BEGIN SIMULATION
          platform [0] entered BMA_SENSE at time [8.000000]
          platform [0] entered BMA_SENSE at time [18.000000]
          platform [0] entered BMA_SENSE at time [28.000000]

                    ... verbose output deleted ...

          WE SHOULD HAVE GONE TO NEW ZEALAND
          WE SHOULD HAVE GONE TO NEW ZEALAND
          WE SHOULD HAVE GONE TO NEW ZEALAND
          *** END SIMULATION


          .... Type <CR> to continue: ↵
================================================================================
```

We are now back at the root menu. Any or all of the input/output files and miscellaneous parameters can be changed and a new simulation run can be made without requiring recompilation. However, if a new BMA or platform type is added, the simulator must be recompiled before execution. This concludes the demonstration of the interactive user interface.

## 4. Suggested Improvements

We offer the following suggestions to future maintainers of this system:

1. Currently, the capabilities must be separately defined through the session manager, thus the user of the program must know what capabilities to assign to each platform type.

A better approach would be to specify the platform capabilities as part of the high-level EMA definition. The kmac parser could then automatically generate entries for each platform type in the capabilities file.

2. The program should allow more detailed specification of the threat scenario, in which the user could specify the initial position, target, and launch time of each missile.

3. There should be an option to display the contents of the current simulator input files in some intelligible format.

4. The program should be ported to Xwindows [4]. This would allow remote execution and a more sophisticated user interface.

5. A facility should be provided for removing previously defined platform types from the context. The only way to do this now is to manually edit the context file.

## References


[1] Carter, S. and Mizell, D., "ISI's SDI Architecture Simulator: An Experiment in Adding New Software to the System," ISI Research Report, in preparation, 1988.

[2] Mizell, D., Tung, Y., Coatney, S., Carter, S., Sherman, R., and Najjar, W., "On the Design of ISI's Initial Prototype SDI Architecture Simulator," ISI Research Report, ISI/RR-88-205, March 1988.

[3] Mizell, D. and Carter, S., "ISI's SDI Architecture Simulator: The 'Kmac' Battle Manager Specification Language," ISI Research Report, in preparation.

[4] Scheifler, R. and Gettys, J., "The X Window System," *ACM Transactions on Graphics*, Vol. 5 No. 2, April 1987, pp. 79-109.

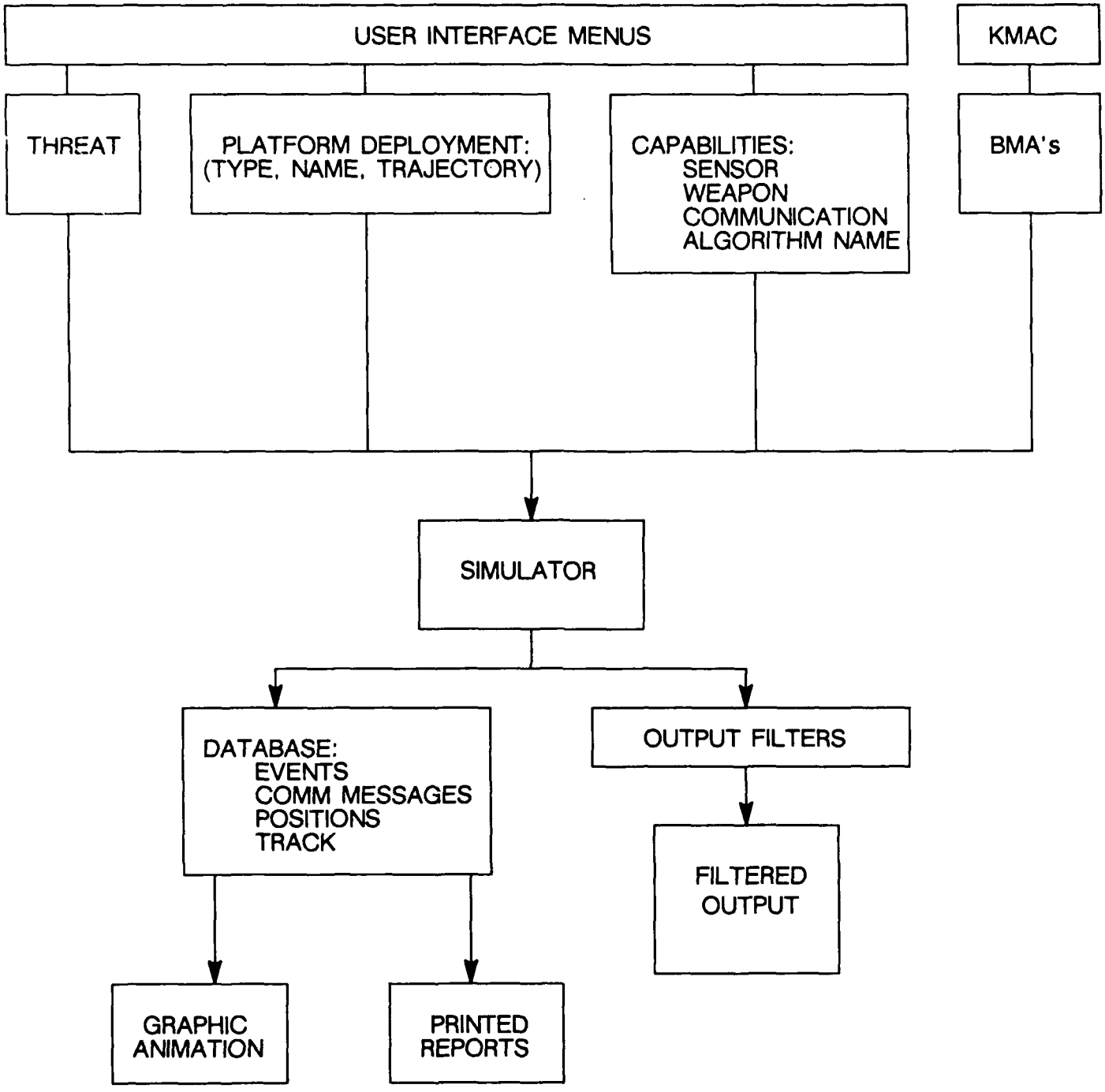



Figure 1. User interface software components

The session manager allows new simulation parameters to be easily and rapidly specified and coordinates user interaction with other simulation system components.

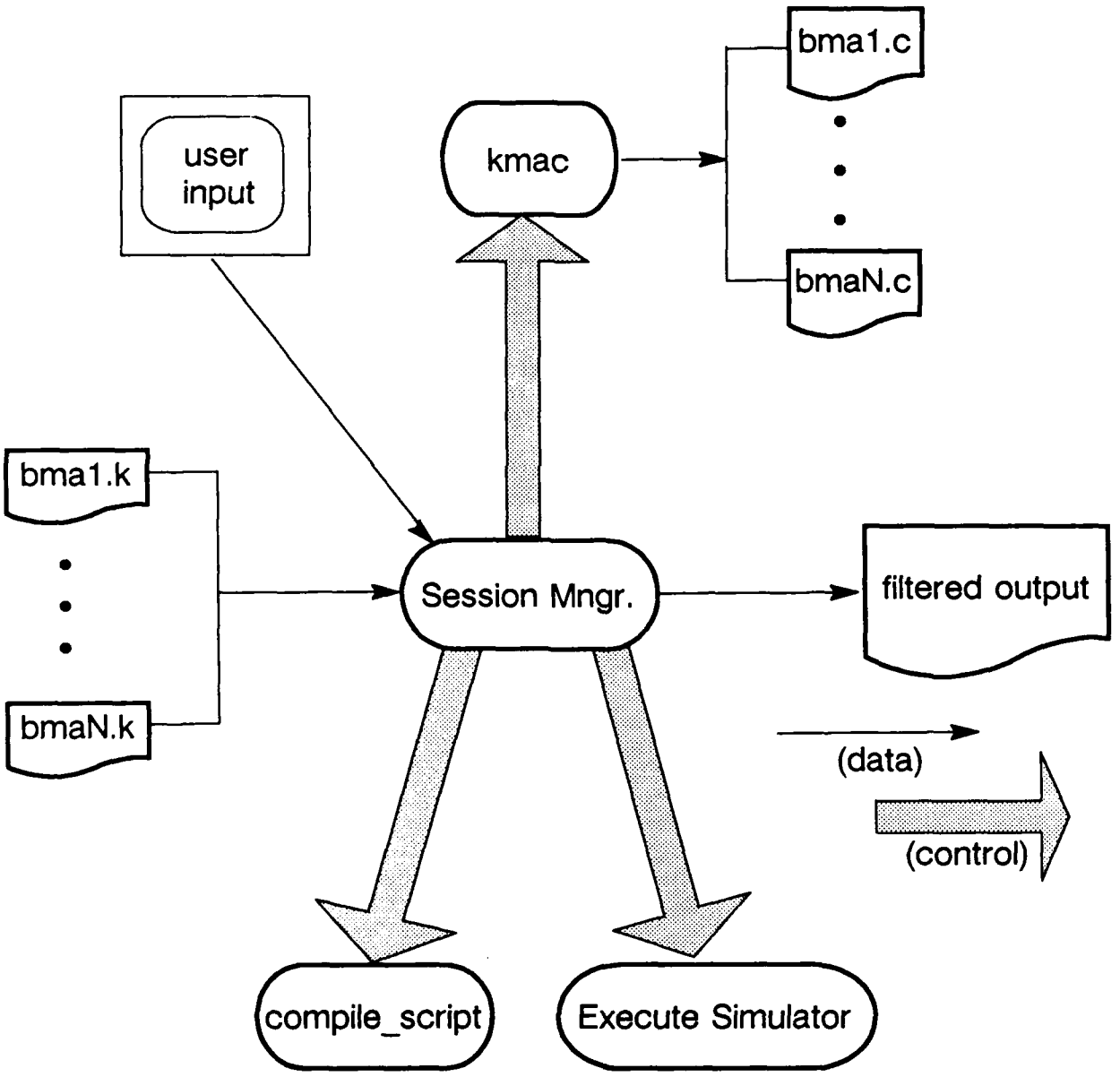



Figure 2. Interactions between user interface components