SAGEN USER'S GUIDE
Version 1.5

Michael R. Kappel
Cy D. Ardoin
Cathy Jo Linn
Joseph L. Linn
John Salasin

April 1988

Prepared for
Strategic Defense Initiative Organization (SDIO)

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311
DEFINITIONS
IDA publishes the following documents to report the results of its work:

Reports
Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers
Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports
IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their context and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 94 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This paper has been reviewed by IDA to ensure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology.

Approved for public release; distribution unlimited
SAGEN USER'S GUIDE Version 1.5 (UNCLASSIFIED)

Michael R. Kappel, Cy D. Ardoin, Cathy Jo Linn, Joseph L. Linn, John Salasin

IDA Paper P-2028 documents a tool that can facilitate the description of processes for the Strategic Defense System (SDS) and Battle Management/Command, Control and Communications (BM/C3) architectures. The process descriptions generated by this tool conform to the Strategic Defense Initiative (SDI) Architecture Dataflow Modeling Technique (SADMT)
SAGEN USER’S GUIDE
Version 1.5

Michael R. Kappel
Cy D. Ardoim
Cathy Jo Linn
Joseph L. Linn
John Salasin

April 1988
## CONTENTS

1. INTRODUCTION ........................................... 2  
2. PROCESS MODEL ........................................... 2  
3. CONVENTIONS ........................................... 4  
   3.1 Notational Conventions ......................... 4  
   3.2 Syntactic Conventions ......................... 6  
   3.3 Semantic Conventions ......................... 6  
   3.4 Programmatic Conventions .................... 6  
   3.5 Keyword Order ................................ 7  
4. SYNTAX ................................................ 7  
   4.1 Process Hierarchy, Ports and Data Types ..... 7  
   4.2 Port Linkages ................................ 7  
   4.3 Process Semantics ............................. 8  
5. SEMANTICS ........................................... 8  
   5.1 Process Hierarchy, Ports and Data Types ..... 8  
   5.2 Port Linkages ................................ 10  
   5.3 Process Semantics ............................. 10  
6. EXECUTION ............................................ 11  
7. REFERENCES .......................................... 12  
APPENDIX A - SAGEN EXAMPLE .......................... 14  
APPENDIX B - GENERATED SADMT ..................... 20  
APPENDIX C - SIMULATION OUTPUT .................... 32
LIST OF FIGURES

Figure 1. Depiction of a BM/C3 Process .......................... 3
Figure 2. Exploded View of BM/C3 .......................... 4
Figure 3. Platforms, Technology Modules and Processes .......... 5
Figure 4. Top Level Platform .......................... 14
Figure 5. Exploded View of Parent Process .......................... 15
1. INTRODUCTION

This paper documents the use of a tool that can facilitate the description of processes for Strategic Defense System (SDS) and Battle Management/Command, Control and Communications (BM/C3) architectures. The generated process description conforms to the Strategic Defense Initiative (SDI) Architecture Dataflow Modeling Technique (SADMT). SADMT descriptions use a complex Ada template to simulate SDS and BM/C3 architectures. This tool, called SAGEN (for SADMT Generator), accepts a simpler specification of the architecture and automatically generates the required SADMT template.

SAGEN eliminates much of the drudgery of specifying the SADMT template for each module. However, Ada must still be used to specify port linkages and process semantics. SADMT modules generated by SAGEN may then be compiled by an Ada compiler, linked with the SADMT Simulation Framework [Linn 88] which is also written in Ada, and executed to simulate the performance of the system.

The SAGEN specification provides constructs to support the SADMT process model. A system is modeled as a hierarchy of processes which communicate via ports. Familiarity with this abstract model as described in [Linn 88] is assumed, but familiarity with the syntax and implementation details of SADMT is not required.

This paper is organized as follows:

Section 2 SADMT process model
Section 3 notational, syntactic, and semantic conventions
Section 4 SAGEN syntax
Section 5 SAGEN semantics
Section 6 execution of SAGEN
Section 7 references
Appendix A SAGEN example
Appendix B generated SADMT code
Appendix C Simulation output

2. PROCESS MODEL

To capture SDS and BM/C3 architectural specifications in early design stages, SADMT defines an abstract entity called a “process” and a mechanism for specifying a process as a set of communicating subprocesses. SADMT processes are defined in a specific format and interprocess communications are performed according to a specific model. SADMT architectural descriptions use the standard syntax and semantics of Ada to capture the information needed to simulate the system.

In SADMT, a system is viewed as a hierarchy of processes. The system itself is the zero-th level process and is specified as a network of level-one processes; similarly, a level-n process may be specified as an interconnected set of level-(n+1) processes. Eventually, some processes will be leaf nodes since they are not decomposed further. The leaf processes contain the semantics of
the system. Leaf processes need not all be at the same level.

SADMT processes are defined to have "ports", i.e., windows for passing data into or out of a process. All interprocess communication is accomplished via these ports. A port is either an input port or an output port; SADMT makes no provision for bi-directional ports. Furthermore, ports in SADMT are typed to restrict the data which may flow into or out of a port.

SADMT provides a facility to specify the interconnections of communicating processes. There are three types of interprocess communications links: (1) internal, (2) input-inherited, and (3) output-inherited. In all cases, the data type of the connected ports must be the same. The first type of link, internal, connects an output port of a subprocess to an input port of another subprocess of the same parent process. Inherited links capture the concept that data flowing through a port on a higher level process is actually the input or output of lower level processes. An input-inherited link connects an input port on a parent process to an input port one of its child subprocesses. An output-inherited link connects an output port on a child subprocess to an output port on its parent process.

Various types of links are depicted in Figure 1. The highest level of a BM/C3 process is shown with a single input port and a single output port. (Note that port names and types are not indicated.) Figure 2 is an exploded view of the BM/C3 process, seen as an interconnected set of three subprocesses: (1) threat assessment, (2) weapon assignment, and (3) view of world (a process to manage retained state data.) The solid lines represent internal links, while the dashed lines represent inherited links.

![Figure 1. Depiction of a BM/C3 Process](image-url)

SADMT processes are simulated within the SADMT Simulation Framework [Linn 88]. The Simulation Framework simulates the physical environment in which the SDS operates. The Simulation Framework employs two primitives - platforms and cones. Platforms represent all physical entities including the sensors, weapons, and carrier vehicles of the SDS and the weapons and debris of the threat. Cones represent entities such as communication waves and laser beams.

Platforms are composed of logical processes and technology modules (see Figu 3). Technology modules represent hardware technology such as sensors, weapons, communications and boosters. Technology modules provide the interface between SADMT processes and the Simulation Framework.
3. CONVENTIONS

3.1 Notational Conventions

The following notational conventions are used in the syntactic specification of SAGEN:

- `<item>` a variable item
- `[item]` an optional item
- `{item1 | item2}` item1 or item2
- `{items}*` items repeated zero or more times

The variables in the syntactic specification are defined as follows:

- `<alias>` a string of characters
- `<data_type>` a valid Ada data type
- `<declarations>` valid Ada declarations
- `<default>` a valid Ada expression
- `<discriminant>` a valid Ada discriminant
- `<name>` a valid Ada identifier
- `<param_list>` a list of parameters separated by commas
Figure 3. Platforms, Technology Modules and Processes
3.2 Syntactic Conventions

SAGEN syntactic conventions are Ada-like and non-restrictive:

1. A SAGEN statement may extend over more than one physical line.
2. More than one SAGEN statement may appear on one physical line.
3. SAGEN keywords and variables may appear in any column.
4. Ada comments may be embedded within a SAGEN statement.

3.3 Semantic Conventions

SAGEN semantic conventions are as follows:

1. Non-SAGEN code (i.e. Ada source and comments) should be valid Ada. If the code is not legal Ada, SAGEN will function properly, but the generated SADMT code will not compile.
2. The Ada source code appearing within SAGEN blocks should not violate the SADMT process model. The user-supplied Ada code should not include direct calls of other process tasks nor the abort statement.
3. The data type of a port is assumed to be declared in a user-supplied package having the same name as the data type followed by "_pkg". For example, the package Order_pkg will be made visible to the process that has a port of type Order.
4. A temporary file called temporary.sagen is created and deleted by SAGEN. This file may appear if SAGEN terminates abnormally.
5. Comments placed within the main SAGEN block will appear before the package specification in the generated SADMT code. Comments placed within Ada source code in any SAGEN block will appear in the generated SADMT code as entered.
6. Any non-SAGEN code placed between SAGEN blocks will appear at the top of the next generated SADMT file.

3.4 Programmatic Conventions

1. The maximum number of subprocesses per parent process is set at 1000. This limit can be raised by changing constant max_array_bound in the SAGEN source file.
2. Variable names are limited to 80 characters. This limit can be raised by changing constant max_line_length in the SAGEN source file.
3. The generated SADMT files will be 80 characters wide. This convention can be altered by changing constant max_line_length in the SAGEN source file.
4. One physical input line is limited to 256 characters. This limit can be raised by changing variable line in the SAGEN source file.
3.5 Keyword Order

Process, platform or technology module must appear as the first SAGEN statement of each specification. Subprocess, import, export, parameters, subdata and cone may follow in any order.

Process, platform or technology module and end must appear once for each specification. Subprocess, import, export, parameters, subdata and cone may each appear zero or more times.

One link or one task block should then follow the main specification. A link block should appear for a platform or a non-leaf process or technology module. A task block should appear for a leaf process or technology module.

4. SYNTAX

SDS and BM/C3 architectures are described in three parts:

1. Process hierarchy, ports and data types
2. Port linkages
3. Process semantics

4.1 Process Hierarchy, Ports and Data Types

The syntax of SAGEN statements for specifying process hierarchy, ports and data types is:

```plaintext
[<with_or_use>]
{ $process <name> | $platform <name> := <alias> | $tech[nology]_module <name> | $dynamic-tech[nology]_module <name> := <alias> } is
$subprocess[es] <name> [(<range>)] [:= (<param_list>)]
    , <name> [(<range>)] [:= (<param_list>)]*;
$[selectable_] [{ data_ | control_ | mech_ } { inport[s] | outport[s] }
    <name> [(<range>)] : <data_type> [:= (<param_list>)]
    , <name> [(<range>)] : <data_type> [:= (<param_list>)]*;
$[cone | event | platform}_inport <name>;
$parameter[s] <name> : <data_type> [(<discriminant>)] := (<default>)
    , <name> : <data_type> [(<discriminant>)] := (<default>)
    , <name> : <data_type> [(<discriminant>)] := (<default>)]*;
$subdata <data_type> {, <data_type> }*;
$cone[s] <data_type> {, <data_type> }*;
$begin
  [Ada source lines (body)]
$end;
```

4.2 Port Linkages

The syntax of SAGEN statements for specifying port linkages is:

```plaintext
[<with_or_use>]
$link[s] <name> is
```

UNCLASSIFIED
4.3 Process Semantics

The syntax of SAGEN statements for specifying process semantics is:

```
[<declarations>]
$begin
[Ada source lines (link)]
$end;
```

5. SEMANTICS

5.1 Process Hierarchy, Ports and Data Types

The semantics of each SAGEN statement for specifying process hierarchy, ports and data types are described below.

```
<with_or_use>
$task <name> is
[Ada source lines (task)]
$end;
```

These Ada context clauses (with) or visibility clauses (use) are placed at the head of the procedure specification.

```
{ $process <name> | $platform <name> := <alias> | $tech[ology]_module <name> | $dynamic.tech[ology]_module <name> := <alias> } is
```

Process, platform or technology module specifies the name of an SDI process, platform or technology module, respectively. In addition, a designator string name must be specified for a platform or a dynamic technology module. A dynamic technology module is a special module to facilitate the specification of companion modules such as sensor returns (see [Linn 88]). The specification and body of the creator task (which is what makes a platform process different from a regular SADMT process) is automatically inserted into the task body of the named platform.

```
$subprocess[es] <name> [(<range>)] := (<param_list>)
{, <name> [(<range>)] := (<param_list>) }*
```

Subprocess specifies the names of the SDI subprocesses associated with the named process, platform or technology module. The range parameter is used to specify an array of subprocesses. The param_list parameter may be used to specify parameters for initializing the subprocess (see [Linn 88]). These parameters will be included in the call to procedure initialize for the subprocess. If the named process, platform or technology module has no subprocesses (i.e. it is a leaf), the subprocess statement must not appear.
Inport specifies the names of ports that input data to the named process or technology module and the type of data flowing into each port. Outport specifies the names of ports that output data from the named process or technology module and the type of data flowing out of each port. (Platforms do not have ports. Dynamic technology modules have only special input ports (see below).) Ports may be further designated as selectable. A selectable port is one of a special subset of ports of the same type to which or from which data may be specifically directed (see [Linn 88]). Ports may also be designated as data, control or mechanism ports (a la the SDI System Design Language [SRS 87]). Such a designation is for documentation purposes only and does not influence the generated SADMT. The range parameter is used to specify an array of ports. The param_list parameter may be used to specify parameters for initializing the port (see [Linn 88]). These parameters will be included in the call to procedure initialize for the port.

Parameter specifies a set of parameters that are supplied to the initialize procedure in the named process, platform or technology module. Data type specifies the data type of the named parameter and default specifies its default value. A discriminant may be supplied for the data type.

Subdata specifies the types of data that flow within and between the subprocesses of the named process, platform or technology module. These data types are made visible to the package specification of the named process or technology module. A data type that is already specified for a port at this level need not be respecified as subdata for its subprocesses. If the named process, platform or technology module has no subprocesses, the subdata statement must not appear.

Cone specifies data types for cones. Cones only make sense for leaf technology modules. A warning will be generated by SAGEN if cones are specified for processes or platforms or nonleaf technology modules. The cone data types are made visible to the package specification of the named technology module. Furthermore, procedure create_cone is renamed in the task body.
Begin specifies the beginning of a block of Ada source code.

[Ada source lines (body)]

These Ada source lines are procedures which are placed in the package body to be referenced by the initialize procedure or the task associated with the named process, platform or technology module.

$end;

End specifies the end of the process, platform or technology module block.

5.2 Port Linkages

The semantics of each SAGEN statement for specifying port linkages is described below.

[<with_or_use>]

These Ada context clauses (with) or visibility clauses (use) are placed at the head of the initialize procedure.

$link[s] <name> is

Link indicates the start of a block of Ada source lines that perform the necessary linkages among the subprocesses of the named process, platform or technology module. The name parameter must correspond to the name of the previous process, platform or technology module block. The link block must not be specified for a leaf process or technology module.

[<declarations>]

These Ada declarations are placed in the declarative region of the initialize procedure.

$begin

Begin specifies the beginning of a block of Ada source code.

[Ada source lines (link)]

These Ada source lines specify the linkages among the subprocesses of the named process, platform or technology module.

$end;

End specifies the end of the linkage block.

5.3 Process Semantics

The semantics of each SAGEN statement for specifying process semantics is described below.

[<with_or_use>]
These Ada context clauses (with) or visibility clauses (use) are placed at the head of the task body.

$task <name> is

Task indicates the start of a block of Ada source lines that supply the semantics of the named process or technology module. The name parameter must correspond to the name of the previous process or technology module block. The task block must not be specified for a platform or a non-leaf process or technology module.

[<declarations>]

These Ada declarations are placed in the declarative region of the task body.

$begin

Begin specifies the beginning of a block of Ada source code.

[Ada source lines (task)]

These Ada source lines specify the semantics of the named process or technology module.

$end;

End specifies the end of the task block.

6. EXECUTION

SAGEN is invoked at the user's terminal by typing:

sagen

The following prompt will then appear on the screen:

Please enter the name of the SAGEN file

The user should then enter the name of the file containing the code to be translated.

If any lexical or syntactic errors are detected by SAGEN, an error message will appear on the screen. The message will indicate the type of error and display the line on which the error occurred.

After translation is complete, several new files are created that contain the generated SADMT code. The names of the new files will be:

<name>.a for package specifications
<name>_body.a for package bodies
<name>_link.a for port linkages
<name>_task.a for process semantics
where <name> is the name of the associated process, platform, or technology module.

7. REFERENCES


APPENDIX A - SAGEN EXAMPLE

An example system specification in SAGEN is given in this Appendix, followed by the SADMT generated by the SAGEN processor in Appendix B, and the output generated by the Simulation Framework in Appendix C. An example of a simple SDI architecture specified in SAGEN, along with the automatically generated SADMT and the simulation output is given in [Cohen 88].

The example network of processes is shown in Figures 4 and 5.

Figure 4. Top Level Platform
Figure 5. Exploded View of Parent Process

The following SAGEN represents the system of processes shown in these figures:

```
platform TopLevel_Platfor:n=TOPLEVEL is
  $subprocesses Parent_Process(1..3), RW_Process;
  $subdata Simple_Msg;
$end;

$links TopLevel_Platfor:n is
  $begin
    internal_link (Z.SUB.Parent_Process(1).message_out,
                   Z.SUB.Parent_Process(2).message_in);
    internal_link (Z.SUB.Parent_Process(1).message_out,
                   Z.SUB.Parent_Process(3).message_in);
    internal_link (Z.SUB.Parent_Process(2).message_out,
                   Z.SUB.Parent_Process(3).message_in);
$end;
```
Z.SUB.RW_Process.message_in);
internal_link (Z.SUB.Paren_Process(3).message_out,
Z.SUB.RW_Process.message_in);
internal_link (Z.SUB.RW_Process.message_out,
Z.SUB.Paren_Process(1).message_in);
$end;

Parent Process

$process Parent_Process is
$subprocesses Simple_Process(1..3):=(wt(i));
$import message_in:Simple_Msg;
$outport message_out:Simple_Msg;
$end;

$links Parent_Process is
wt: constant array(1..3) of PDL_time_type := (20,30,50);
$begin
internal_link (Z.SUB.Simple_Process(2).message_out,
Z.SUB.Simple_Process(1).message_in);
internal_link (Z.SUB.Simple_Process(2).message_out,
Z.SUB.Simple_Process(3).message_in);
inherited_link (Z.message_in,
Z.SUB.Simple_Process(2).message_in);
inherited_link (Z.SUB.Simple_Process(3).message_out,
Z.message_out);
$end;

Simple Subprocess

$process Simple_Process is
$import message_in:Simple_Msg;
$outport message_out:Simple_Msg;
$parameter waittime:PDL_time_type :=(20);
$end;

$task Simple_Process is
buffer: Simple_msg;
$begin
loop
  wait_for_activity(Z.PDL);
  buffer:= port_data(Z.message_in);
  consume(Z.message_in);
  wait(Z.PDL,Z.PRM.waittime);
  buffer.last_slot:= buffer.last_slot + 1;
  buffer.route(buffer.last_slot):= integer(Z.PDL.process_id);

UNCLASSIFIED

15

UNCLASSIFIED

15
emit(Z.message_out,buffer);
end loop;
exception
when others =>
  write_process_id(Z.PDL,"AND THEN SOME EXCEPTION in simple_proc, ");
$end;

--------------- RW Subprocess ---------------

$process RW_Process is
  $import message_in : Simple_Msg;
  $export message_out : Simple_Msg;
$end;

$task RW_Process is
  buffer : Simple_Msg;
  start_up_time, last_time : PDL_time_type := 1000;
  which_port : integer := -50;
$begin
  start_up_time := Current_PDL_time;
  loop
    while not port_empty(Z.message_in) loop
      write_process_id(Z.PDL," DEQUEUING=",||,false);
      put_msg(port_data(Z.message_in),0);
      consume(Z.message_in);
    end loop;
    if last_time /= Current_PDL_time then
      if (Current_PDL_time - start_up_time) mod 40 = 0
        or (Current_PDL_time - start_up_time) mod 40 = 30 then
        buffer.time_created := Current_PDL_time;
        buffer.last_slot := 0;
        emit(Z.message_out,buffer);
        last_time := Current_PDL_time;
      end if;
    end if;
    wait_for_activity(Z.PDL,(Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_out.PORT
                         ,Z.message_in.PORT)
                         ,which_port
                         ,Time_out => 10);
  end loop;

UNCLASSIFIED

16
UNCLASSIFIED

exception
  when others =>
    put_line("AND THEN SOME EXCEPTION in rw_proc");
$end;
APPENDIX B - GENERATED SADMT

The following SADMT output files are created by the SAGEN tool:

******************** TopLevel_Platform.a ********************

with Cones_n_Platforms, Parent_Process_pkg, RW_Process_pkg, Simple_Msg_pkg;
package TopLevel_Platform_pkg is
use Cones_n_Platforms;
use PDL_pkg, Simple_Msg_pkg;
TopLevel_Platform_designator : constant platform_designator_type
:= new string"TOPLEVEL";
TopLevel_Platform_name : PDL_string_ptr := new string"TopLevel_Platform";
TopLevel_Platform_discr_name : PDL_string_ptr := empty_string;
TopLevel_Platform_characteristic : PDL_string_ptr := new string"typename=TopLevel_Platform";
type TopLevel_Platform_subprocesses is private;
package TopLevel_Platform_PARAM_pkg is
type TopLevel_Platform_parameterization is record
  null;
end record;

type TopLevel_Platform_parameterization_ptr is access
  TopLevel_Platform_parameterization;
end TopLevel_Platform_PARAM_pkg;
use TopLevel_Platform_PARAM_pkg;

package TopLevel_Platform_CP_pkg is new interface_procs.PlatformDefiner_pkg
(T=>TopLevel_Platform_parameterization,
  T_ptr=>TopLevel_Platform_parameterization_ptr);
private
use PIG_pkg, Parent_Process_pkg, RW_Process_pkg:
type Parent_Process_vector is array(integer range <>) of
  Parent_Process_type;
type TopLevel_Platform_subprocesses is record
  PIG: PIG_type;
  Parent_Process: Parent_Process_vector(1..3);
  RW_Process: RW_Process_type;
end record;
end TopLevel_Platform_pkg;
UNCLASSIFIED

*************************************************************** TopLevel_Platform_body.a **************************************************************************

package body TopLevel_Platform_pkg is
use Simple_Msg_pkg.PD.Procedures;
use PDL_IO; use TXT_IO, INT_IO, TIME_IO, DURATION_IO;
use interface_procs;

type TopLevel_Platform_block;
type TopLevel_Platform_type is access TopLevel_Platform_block;

type TopLevel_Platform_block is record
  PDL: PDL_ptr := new_PDL_block(platform);
  SUB: TopLevel_Platform_subprocesses;
end record;

procedure initialize (ZZ: out PIG_type; param:
  TopLevel_Platform_parameterization_ptr; my_name: PDL_string_ptr :=
  TopLevel_Platform_name; descr_name: PDL_string_ptr :=
  TopLevel_Platform_discr_name; characteristic: PDL_string_ptr :=
  TopLevel_Platform_characteristic) is separate;

package Creator is new PlatformCreator_pkg (*
  TopLevel_Platform_parameterization,
  TopLevel_Platform_parameterization_ptr, lookup_platform_designator
  (TopLevel_Platform_designator), initialize);*
end TopLevel_Platform_pkg;

*************************************************************** TopLevel_Platform_link.a **************************************************************************

separate (TopLevel_Platform_pkg)
procedure initialize (ZZ: out PIG_type; param:
  TopLevel_Platform_parameterization_ptr; my_name: PDL_string_ptr :=
  TopLevel_Platform_name; descr_name: PDL_string_ptr :=
  TopLevel_Platform_discr_name; characteristic: PDL_string_ptr :=
  TopLevel_Platform_characteristic) is

Z: TopLevel_Platform_type;

begin
Z := new TopLevel_Platform_block;
declare
  PIG : PIG_type renames Z.SU.B.PIG;
  MYSELF : PDL_ptr renames Z.PDL;
begin
  set_process_parent(MYSELF,null,my_name,descr_name,characteristic);
  if init_debug_level > 100 then
write_process_full(MYSELF,"*init> "," before start_up");
end if;
initialize(PIG,MYSELF);
for i in 1..3 loop
initialize(Parent_Process(i),MYSELF
, discr_name => new string(integer'IMAGE(i)));
end loop;
initialize(RW_Process,MYSELF);
ZZ := PIG;

internal_link (Z.SUB.Parent_Process(1).message_out,
  Z.SUB.Parent_Process(2).message_in);
internal_link (Z.SUB.Parent_Process(1).message_out,
  Z.SUB.Parent_Process(3).message_in);
internal_link (Z.SUB.Parent_Process(2).message_out,
  Z.SUB.RW_Process.message_in);
internal_link (Z.SUB.Parent_Process(3).message_out,
  Z.SUB.RW_Process.message_in);
internal_link (Z.SUB.RW_Process.message_out,
  Z.SUB.Parent_Process(1).message_in);
end;

if init_debug_level > 100 then
  write_process_full(Z.PDL,"*init> "," after start_up");
end if;
exception
  when others => write_process_full(Z.PDL,"***Some error in ","**");
end initialize;

*************** Parent_Process.a **************

--- Parent Process ---

with PDL_pkg,Simple_Msg_pkg,Simple_Process_pkg;
package Parent_Process_pkg is
  use PDL_pkg,Simple_Msg_pkg;

type Parent_Process_block;
type Parent_Process_type is access Parent_Process_block;

type Parent_Process_subprocesses is private;

type Parent_Process_block is record
  PDL: PDL_ptr := new_PDL_block(nonleaf);

UNCLASSIFIED 21
SUB: Parent_Process_subprocesses;
message_in: Simple_Msg_ipptr := new Simple_Msg_port(inport);
message_out: Simple_Msg_opptr := new Simple_Msg_port(outport);
end record;

Parent_Process_name : PDL_string_ptr := new string’
("Parent_Process");
Parent_Process_type_name : PDL_string_ptr := new string’
("Parent_Process");
Parent_Process_discr_name : PDL_string_ptr := empty_string;
Parent_Process_characteristic : PDL_string_ptr := new string’
("typename=Parent_Process");

procedure initialize (Z: in out Parent_Process_type; Parent: PDL_ptr;
my_name: PDL_string_ptr := Parent_Process_name;
discr_name: PDL_string_ptr := Parent_Process_discr_name;
type_name: PDL_string_ptr := Parent_Process_type_name;
characteristic: PDL_string_ptr := Parent_Process_characteristic);

private
use Simple_Process_pkg;
type Simple_Process_vector is array(integer range <>) of
Simple_Process_type;
type Parent_Process_subprocesses is record
   Simple_Process: Simple_Process_vector(1..3);
end record;
end Parent_Process_pkg;

package body Parent_Process_pkg is
use PDL_IO; use TXT_IO, INT_IO, TIME_IO, DURATION_IO;
use Simple_Msg_pkg.PD.Procedures;
procedure initialize (Z: in out Parent_Process_type; Parent: PDL_ptr;
my_name: PDL_string_ptr := Parent_Process_name;
discr_name: PDL_string_ptr := Parent_Process_discr_name;
type_name: PDL_string_ptr := Parent_Process_type_name;
characteristic: PDL_string_ptr := Parent_Process_characteristic
) is separate;
end Parent_Process_pkg;

package Parent_Process_link.a is
procedure initialize (Z: in out Parent_Process_type; Parent: PDL_ptr;
my_name: PDL_string_ptr := Parent_Process_name;
discr_name: PDL_string_ptr := Parent_Process_discr_name;

UNCLASSIFIED
UNCLASSIFIED

```
type_name: PDL_string_ptr := Parent_Process_type_name;
characteristic: PDL_string_ptr := Parent_Process_characteristic) is

wt: constant array(1..3) of PDL_time_type := (20,30,50);
begin
    Z := new Parent_Process_block;
declare
        message_in : Simple_Msg_ipptr renames Z.message_in;
        message_out : Simple_Msg_opptr renames Z.message_out;
        MYSELF : PDL_ptr renames Z.PDL;
    begin
        set_process_parent(MYSELF,Parent,my_name,discr_name,characteristic);
        if init_debug_level > 130 then
            write_process_full(MYSELF,"*init> "," before start-up");
        end if;
        initialize(message_in,MYSELF,"portname=message_in","message_in");
        initialize(message_out,MYSELF,"portname=message_out","message_out");
        for i in 1..3 loop
            initialize(Simple_Process(i),MYSELF,wt(i)
                ,discr_name => new string'(integer'IMAGE(i)));
        end loop;
        internal_link (Z.SUB.Simple_Process(2).message_out,
            Z.SUB.Simple_Process(1).message_in);
        internal_link (Z.SUB.Simple_Process(2).message_out,
            Z.SUB.Simple_Process(3).message_in);
        inherited_link (Z.message_in,
            Z.SUB.Simple_Process(2).message_in);
        inherited_link (Z.SUB.Simple_Process(3).message_out,
            Z.message_out);
    end;
    make_known(Z.PDL);
    if init_debug_level > 130 then
        write_process_full(Z.PDL,"*init> "," after start-up");
    end if;
    exception
        when others => write_process_full(Z.PDL,"***Some error in ","**");
    end initialize;

**************************************************************************
```

UNCLASSIFIED
with PDL_pkg, Simple_Msg_pkg;
package Simple_Process_pkg is
  use PDL_pkg, Simple_Msg_pkg;

package Simple_Process_PARAM_pkg is
  type Simple_Process_parameterization is record
    waittime: PDL_time_type := 20;
    end record;
end Simple_Process_PARAM_pkg;
use Simple_Process_PARAM_pkg;

type Simple_Process_block; type Simple_Process_type is access Simple_Process_block;
task type Simple_Process_task is
  entry start_up(Z: Simple_Process_type);
end Simple_Process_task;
type Simple_Process_task_ptr is access Simple_Process_task;

type Simple_Process_block is record
  PDL: PDL_ptr := new PDL_block(leaf);
  SEM: Simple_Process_task_ptr;
  PRM: Simple_Process_parameterization;
  message_in: Simple_Msg_ipptr := new Simple_Msg_port(inport);
  message_out: Simple_Msg_opptr := new Simple_Msg_port(outport);
end record;

Simple_Process_name : PDL_string_ptr := new string'("Simple_Process");
Simple_Process_type_name : PDL_string_ptr := new string'("Simple_Process");
Simple_Process_discr_name : PDL_string_ptr := empty_string;
Simple_Process_characteristic : PDL_string_ptr := new string'("typename=Simple_Process");

procedure initialize (Z: in out Simple_Process_type; Parent: PDL_ptr;
  waittime_param : PDL_time_type := 20; my_name: PDL_string_ptr :=
  Simple_Process_name; descr_name: PDL_string_ptr :=
  Simple_Process_discr_name; type_name: PDL_string_ptr :=
  Simple_Process_type_name; characteristic: PDL_string_ptr :=
  Simple_Process_characteristic);
end Simple_Process_pkg;

************************************************** Simple_Process_body.a **************************************************
package body Simple_Process_pkg is
  use PDL_IO; use TXT_IO, INT_IO, TIME_IO, DURATION_IO;

UNCLASSIFIED 24
use Simple_Msg_pkg.PDL.Procedures;
task body Simple_Process_task is separate;
procedure initialize (Z: in out Simple_Process_type; Parent: PDL_ptr;
  waittime_param : PDL_time_type := 20; my_name: PDL_string_ptr :=
  Simple_Process_name; descr_name: PDL_string_ptr :=
  Simple_Process_discr_name; type_name: PDL_string_ptr :=
  Simple_Process_type_name; characteristic: PDL_string_ptr :=
  Simple_Process_characteristic) is separate;
end Simple_Process_pkg;

********************************** Simple_Process_link.a ***************************
separate (Simple_Process_pkg)
procedure initialize (Z: in out Simple_Process_type; Parent: PDL_ptr;
  waittime_param : PDL_time_type := 20; my_name: PDL_string_ptr :=
  Simple_Process_name; descr_name: PDL_string_ptr :=
  Simple_Process_discr_name; type_name: PDL_string_ptr :=
  Simple_Process_type_name; characteristic: PDL_string_ptr :=
  Simple_Process_characteristic) is
begin
  Z := new Simple_Process_block;
declare
  message_in : Simple_Msg_ipptr renames Z.message_in;
  message_out : Simple_Msg_opptr renames Z.message_out;
  waittime : PDL_time_type renames Z.PRM.waittime;
  MYSELF : PDL_ptr renames Z.PDL;
begin
  set_process_parent(MYSELF,Parent,my_name,descr_name,characteristic);
  if init_debug_level > 130 then
    write_process_full(MYSELF,"*init> "," before start_up");
  end if;
  waittime := waittime_param;
  initialize(message_in,MYSELF,"portname=message_in","message_in");
  initialize(message_out,MYSELF,"portname=message_out","message_out");
end;
Z.SEM := new Simple_Process_task;
Z.SEM.start_up(Z);

if init_debug_level > 130 then
  write_process_full(Z.PDL,"*init> "," after start_up");
end if;
exception
  when others => write_process_full(Z.PDL,"***Some error in ","***");
end initialize;

********************************** Simple_Process_task.a ***************************
separate (Simple_Process_pkg)
task body Simple_Process_task is
  use timing_ops;
  Z : Simple_Process_type := null;

  buffer: Simple_msg;
begin
  accept start_up(Z:Simple_Process_type) do
    Simple_Process_task.Z := Z;
    make_known(Z.PDL);
  end start_up;

  declare
    message_in : Simple_Msg_ipptr renames Z.message_in;
    message_out : Simple_Msg_opptr renames Z.message_out;
    waittime : PDL_time_type renames Z.PRM.waittime;
    MYSELF : PDL_ptr renames Z.PDL;
  package WAITING_pkg is new Wait_pkg(MYSELF);
  use WAITING_pkg;
begin
  wait_for_initialization;
  loop
    wait_for_activity(Z.PDL);
    buffer:= port_data(Z.message_in);
    consume(Z.message_in);
    wait(Z.PDL,Z.PRM.waittime);
    buffer.last_slot:= buffer.last_slot + 1;
    buffer.route(buffer.last_slot):= integer(Z.PDL.process_id);
    emit(Z.message_out,buffer);
  end loop;
exception
  when others =>
    write_process_id(Z.PDL,"AND THEN SOME EXCEPTION in simple_proc, ");
  end;
  write_process_full(Z.PDL,"*********************** Invalid SADMT Process");
  kill_process(Z.PDL);
end Simple_Process_task;

RW_Process.pkg

RW Subprocess

with PDL_pkg,Simple_Msg_pkg;
package RW_Process_pkg is

UNCLASSIFIED 26
use PDL_pkg, Simple_Msg_pkg;

type RW_Process_block;
type RW_Process_type is access RW_Process_block;

task type RW_Process_task is
  entry startup(Z:RW_Process_type);
end RW_Process_task;
type RW_Process_task_ptr is access RW_Process_task;

type RW_Process_block is record
  PDL: PDL_ptr := new_PDL_block(leaf);
  SEM: RW_Process_task_ptr;
  message_in: Simple_Msg_ipptr := new Simple_Msg_port(inport);
  message_out: Simple_Msg_opptr := new Simple_Msg_port(outport);
end record;

RW_Process_name : PDL_string_ptr := new_string('RW_Process');
RW_Process_type_name : PDL_string_ptr := new_string('RW_Process');
RW_Process_discr_name : PDL_string_ptr := new_string('RW_Process');
RW_Process_characteristic : PDL_string_ptr := new_string('RW_Process');

procedure initialize (Z: in out RW_Process_type; Parent: PDL_ptr;
  my_name: PDL_string_ptr := RW_Process_name;
  discr_name: PDL_string_ptr := RW_Process_discr_name;
  type_name: PDL_string_ptr := RW_Process_type_name;
  characteristic: PDL_string_ptr := RW_Process_characteristic);
end RW_Process_pkg;

package body RW_Process_pkg is
  use PDL_IO; use TXT_IO, INT_IO, TIME_IO, DURATION_IO;
  use Simple_Msg_pkg.PD. Procedures;
  task body RW_Process_task is separate;
  procedure initialize (Z: in out RW_Process_type; Parent: PDL_ptr;
    my_name: PDL_string_ptr := RW_Process_name;
    discr_name: PDL_string_ptr := RW_Process_discr_name;
    type_name: PDL_string_ptr := RW_Process_type_name;
    characteristic: PDL_string_ptr := RW_Process_characteristic
  ) is separate;
end RW_Process_pkg;

separate (RW_Process_pkg)
procedure initialize (Z: in out RW_Process_type; Parent: PDL_ptr;
my_name: PDL_string_ptr := RW_Process_name; discr_name: PDL_string_ptr := RW_Process_discr_name; type_name: PDL_string_ptr := RW_Process_type_name; characteristic: PDL_string_ptr := RW_Process_characteristic) is

begin
    Z := new RW_Process_block;
declare
        message_in : Simple_Msg_ipptr renames Z.message_in;
        message_out : Simple_Msg_opptr renames Z.message_out;
        MYSELF : PDL_ptr renames Z.PDL;
begin
    set_process_parent(MYSELF,Parent,my_name,discr_name,characteristic);
    if init_debug_level > 130 then
        write_process_full(MYSELF,"*init"," before start-up");
    end if;
    initialize(message_in,MYSELF,"portname=message_in","message_in");
    initialize(message_out,MYSELF,"portname=message_out","message_out");
end;
Z.SEM := new RW_Process_task;
Z.SEM.start_up(Z);
if init_debug_level > 130 then
    write_process_full(Z.PDL,"*init"," after start-up");
end if;
exception
    when others => write_process_full(Z.PDL,"***Some error in","**");
end initialize;

********************** RW_Process_task.a **********************

** RW_Process_task.a **

separate (RW_Process_pkg)
task body RW_Process_task is
    use timing_ops;
    Z : RW_Process_type := null;

    buffer : Simple_Msg;
    start_up_time, last_time : PDL_time_type := 1000;
    which_port : integer := -50;
begin
    accept start_up(Z:RW_Process_type) do
        RW_Process_task.Z := Z;
        make_known(Z.PDL);
    end start_up;

    declare
        message_in : Simple_Msg_ipptr renames Z.message_in;
        message_out : Simple_Msg_opptr renames Z.message_out;
MYSELF : PDL_ptr renames Z.PDL;
package WAITING_pkg is new Wait_pkg(MYSELF);
use WAITING_pkg;
begin
wait_for_initialization;
start_up_time := Current_PDL_time;
loop
while not port_empty(Z.message_in) loop
    write_process_id(Z.PDL," DEQUEUING-",",",false);
    put_msg(port_data(Z.message_in),0);
    consume(Z.message_in);
end loop;
if last_time /= Current_PDL_time then
    if (Current_PDL_time - start_up_time) mod 40 = 0
        or (Current_PDL_time - start_up_time) mod 40 = 30 then
        buffer.time_created := Current_PDL_time;
        buffer.last_slot := 0;
        emit(Z.message_out,buffer);
        last_time := Current_PDL_time;
    end if;
end if;
end if;
wait_for_activity(Z.PDL,(Z.message_out.PORT
    ,Z.message_out.PORT
    ,Z.message_out.PORT
    ,Z.message_out.PORT
    ,Z.message_out.PORT
    ,Z.message_out.PORT
    ,Z.message_out.PORT
    ,which_port,Time_out => 10);
end loop;
exception
    when others =>
        put_line("AND THEN SOME EXCEPTION in rw_proc");
end;
write_process_full(Z.PDL,"*************** Invalid SADMT Process");
kill_process(Z.PDL);
end RW_Process_task;
APPENDIX C - SIMULATION OUTPUT

The following is the output generated by running the SADMT description of the system of processes depicted in Figures 4 and 5 on the prototype SSF simulation facility:

hello
simulation begins...........

DEQUEUEING-RW_Process/[PL]PR=[1]10<t=160>|the message ts,r=0:2:3:8:9:


DEQUEUEING-RW_Process/[PL]PR=[1]10<t=390>|the message ts,r=70:2:3:8:9:
DEQUEUEING-RW_Process/[PL]PR=[1]10<t=470>|the message ts,r=80:2:3:8:9:
Distribution List for P-2028

Sponsor

LTC Jon Rindt
SDIO
Pentagon 
BM/C3
1E149
Washington, DC 20301-7100

CAPT David Hart
SDIO
Pentagon 
BM/C3
1E149
Washington, DC 20301-7100

LTC Chuck Lillie
SDIO
Pentagon 
BM/C3
1E149
Washington, DC 20301-7100

LTC Pete Sowa
SDIO
Pentagon 
BM/C3
1E149
Washington, DC 20301-7100

Other

Defense Technical Information Center 2 copies
Cameron Station
Alexandria, VA 22314

(Each should receive 1 copy.)

F.R. Albrow
MoD (PE) RSRE
St. Andrews Road
Great Malvern
Worcester WR14 3PS
England

John Anderson
11569 Hicks Court
Manassas, VA 22111
Jim Armitage  
GTE SSD  
1 Research Dr.  
Westborough, MA 01581

David Audley, Associate  
Financial Strategies  
Prudential Bache Securities  
26th Floor  
199 Water  
New York, NY 10292

Dr. Algirdas Avizienis  
Computer Science Department  
4731 Boelter Hall  
University of California, Los Angeles  
Los Angeles, CA 90024

Dan Baker  
TASC  
55 Walkers Brook Drive  
Redding, MA 01867

Gary H. Barber  
Program Manager  
Intermetrics, Inc.  
1100 Hercules, Suite 300  
Houston, TX 77058

John Barry  
SJ-72  
Rockwell  
P.O. Box 3644  
Seal Beach, CA 90740-7644

Elizabeth Bently  
Marketing Coordinator  
Research Triangle Institute  
P.O. Box 12184  
Research Triangle Park, N.C. 27709

Cheryl Bittner  
General Electric  
Box 8555, Bldg. 19, Suite 200  
Philadelphia, PA 19101

Grady Booch  
Director, Software Engineering Program  
Rational  
1501 Salado Dr.  
Mountain View, CA 94043

Gina Bowden  
MS 202  
Teledyne Brown Engineering  
Cummings Research Park  
Huntsville, AL 35807
Bijoy G. Chatterjee  
Deputy Director  
Advanced Technology Systems  
ESL/TRW  
495 Java Dr.  
P.O. Box 3510  
Sunnyvale, CA 3510

John L. Chinn  
Manager, Advanced Technology program  
General Electric Space Systems Division  
4041 North First St.  
San Jose, CA 95134

Starla Christakos  
AFATL/SAI  
Eglin Air Force Base, FL 320542

Dr. Joe Clema  
ECAC/IITRI  
185 Admiral Cochrane Drive  
Annapolis, MD 21401

Karen Coates  
Program Manager, Advanced Technology  
ESL/TRW  
495 Java Dr.  
P.O. Box 3510  
Sunnyvale, CA 94088-3510

Susan Coatney  
Information Science Institute  
University of Southern California  
4676 Admiralty Way  
Marina del Ray, CA 90292-5950

Mr. Danny Cohen  
Director, System Division  
Information Science Institute  
University of Southern California  
4676 Admiralty Way  
Marina del Rey, CA 90292-6695

Christopher F. Cole  
Marketing Representative  
Technology Programs  
General Electric Company  
Space Systems Division  
Valley Forge Space Center  
P.O. Box 8555  
Philadelphia, PA 19101

Carol Combs  
National Security Agency  
9800 Savage Road  
Ft. Meade, MD 20755-6000
Edward R. Comer  
Software Productivity Solutions, Inc.  
122 North 4th Av.  
Indialantic, FL 32903

Lawrence L. Cone  
Cone Software Laboratory  
312 East Summit Av.  
Haddonfield, N.J. 08033

Robert P. Cook  
Department of Computer Science  
Thornton Hall  
University of Virginia  
Charlottesville, VA 22903

Chuck Cooper  
Control Data Corporation  
901 E. 78th Street  
MS BMW 03M  
Minneapolis, MN 55420

Lee Cooper  
Advanced Technology  
2121 Crystal Drive, Suite 200  
Arlington, VA 22202

Mark Cosby  
Science Applications International Corp.  
1710 Goodridge Drive  
McLean, VA 22012

L. Cristina  
USASD  
ATTN: DASD-H-SBY  
P.O. 1500  
106 Wynn Dr.  
Huntsville, AL 35807-3801

Vincent Dambrauskas  
Technical Director  
Washington Technical Center  
Strategic Systems Division  
GTE Government Systems Corporation  
6850 Versar Center, Suite 354  
Springfield, VA 22151-4196

Samuel A. DeNitto  
Romse Air Development Center  
RADC/COE, Bldg. 3  
Griffis AFB, NY 13441-5700

Cameron M. G. Donaldson  
Software Productivity Solutions, Inc.  
122 North 4th Av.  
Indialantic, FL 32903
Ralph Duncan  
Control Data Government Systems  
300 Embassy Row  
Atlanta, GA 30328

Stephen Edwards  
1-55 Caltech  
Pasadena, CA 91126

Jim Egolf  
Ford Aerospace & Computer Corp.  
10440 State Hwy. 83  
Colorado Springs, CO 80908

David A. Fisher  
Incremental Systems Corporation  
319 S. Craig St.  
Pittsburgh, PA 15213

Dave Fittz  
STARS Program Office  
OUS  
OUSDRE (R&AT/CET)  
3D139  
1211 Fern St., C-112  
Washington, D.C. 20301-3081

Michel A. Floyd  
Integrated Systems Inc.  
101 University Av.  
Palo Alto, CA 94301-1695

Richard Frase  
SRS Technologies  
1500 Wilson Blvd., Suite 800  
P.O. Box 12707  
Arlington, VA 22209-8707

George Gearn  
Applied Research & Engineering  
7 Railroad Avenue, Suite F  
Bedford, MA 01730

Victor Giddings  
MITRE Corporation  
Burlington Road  
Bedford, MA 01730

Claren Giese  
SDIO  
Pentagon  
T/KE  
Washington, DC 20301-7100

Colin Gilyeat  
Advanced Technology  
2121 Crystal Drive  
Arlington, VA 22202
Robert T. Goettge  
Advanced Systems Technology  
12200 East Briarwood Av.  
Suite 260  
Englewood, CO  80112

H.T. Goranson  
Sirius Inc.  
P.O. Box 9258  
760 Lynnhaven Parkway  
Virginia Beach, VA 23452

Barbara Guyette  
Marketing Specialist  
Ada Products Division  
Intermetrics, Inc.  
733 Concord Av.  
Cambridge, MA 02138

Sarah Hadley  
National Security Agency  
9800 Savage Road  
Fort Meade, MD 20755-6000

Shmuel Halevi  
Ad Cad Inc.  
University Place, Suite 200  
124 Mt. Auburn St.  
Cambridge, MA 02138

Robert Haley  
Director, SDI Programs  
Cray Research, Inc.  
1331 Pennsylvania Avenue, N.W.  
Suite 1331 North  
Washington, DC 2004

Margaret Hamilton, President  
Hamilton Technologies, Inc.  
17 Inman St.  
Cambridge, MA 02139

Duane Harder  
MS B-218  
Los Alamos National Laboratory  
Los Alamos, NM 87545

Evans C. Harrigan  
Software Consultant  
Cray Research, Inc.  
2130 Main Street., Suite 280  
Huntington Beach, CA 92648

Hal Hart  
TRW Defense Systems Group  
One Space Park  
Redondo Beach, CA 90278
James R. Jill  
Manager, Advanced Technologies  
NTB Design  
Martin Marietta Information & Communications Systems  
P.O. Box 1260  
Denver, CO 80201-1260

Sumalee Johnson  
Rockwell Institute  
P.O. Box 3644  
Seal Beach, CA 90704-7644

W.G. (Gray) Jones  
Science Applications International Corporation  
1710 Goodridge Drive  
McLean, VA 22102

Irene G. Kazakova  
Director, Marketing  
Interactive Development Environments  
150 Fourth Street, Suite 210  
San Francisco, CA 94103

Peter Keenan  
Science Ltd.  
Wavendon Towe  
Milton, Keynes  
England MK17-8LX

Judy Kerner  
TRW R2/1134  
One Space Park  
Redondo Beach, CA 90278

Rebecca Kidd  
General Research Corp.  
307 Wynn Drive  
Huntsville, AL 35805

Virginia P. Kobler  
Chief, Technology Branch  
Battle Management Division  
Department of the Army  
Office of the Chief of Staff  
U.S. Army Strategic Defense Command  
P.O. Box 1500  
Huntsville, AL 35807-3801

Dr. Ijur Kulikov  
Intermetrics  
607 Louis Drive  
Warminster, PA 18974

Lt. Ann Kuo  
ESD/ATS  
Hanscom AFB, MA 07831
John Michael Lake  
2311 Galen Dr. #7  
Champaign, IL 61821

John Latimer  
Teledyne Brown Engineering  
300 Sparkman Dr.  
MS 44  
Huntsville, AL 35807

Steve Layton  
Senior Software Engineer  
Martin Marietta Denver Aerospace  
MS L0425  
P.O. Box 179  
Denver, CO 80201

Larry L. Lehman  
Integrated Systems Inc.  
2500 Mission College Road  
Santa Clara, CA 95054

Eric Leighninger  
Dynamics Research  
60 Frontage Road  
Andover, MA 01810

Peter Lempp  
Software Products and Services, Inc.  
14 East 38th Street, 14th Floor  
New York, NY 10016

Bob Liley  
Rockwell International Corporation  
2600 West Minister Blvd.  
Seal Beach, CA 90740-7644

Frank Poslajko  
U.S. Army SDC  
CSSD-H-SI  
Huntsville, AL 35807-3801

Brian Smith  
Mathematics & Computer Science Div.  
Argonne National Laboratory  
Building 221, Room C-219  
9700 South Cass Avenue  
Argonne, IL 60439-4844

Norman G. Snyder  
Director of Software Services  
Jodgrey Associates, Inc.  
462 Highfield Ct.  
Severna Park, MD 21146
Charles M. Vairin
Martin Marietta Denver Aerospace
MS L8079
P.O. Box 179
700 W. Mineral Avenue
Littleton, CO 80201

Dr. Brooks Van Horne
TRW
One Federal Systems Park Drive
Fairfax, VA 22033

John L. Walsh
Riverside Research Institute
Washington Research Office
1701 North Fort Myer Drive, Suite 700
Arlington, VA 22209

G. Karl Warmbrod
SPARTA, Inc.
7926 Jones Branch Road
Suite 1070
Mclean, VA 22180

Erwin H. Warshawsky, President
JRS Research Laboratories, Inc.
202 W. Lincoln Av.
Orange, CA 92665-1040

Capt. Charles R. Waryk
OSD/SDIO/S/SA
Pentagon
Room 1E149
Washington, DC 20301-7100

Anthony Wasserman, President
Interactive Development Environments
150 Fourth St., Suite 210
San Francisco, CA 94103

Gio C. Weiderhold
Department of Computer Science
Stanford University
Stanford, CA 94305-2085

Bruce White
500 Montezuma Avenue, Suite 118
Santa Fe, NM 87501

Dave J. Whitley
Software Engineering Section
SCICON, Ltd.
Wavedon Tower
Wavedon Village
Milton Keynes, England MK178LX
<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>General W.Y. Smith</td>
<td>HQ</td>
<td>1 copy</td>
</tr>
<tr>
<td>Mr. Seymour Deitchman</td>
<td>HQ</td>
<td>1 copy</td>
</tr>
<tr>
<td>Mr. Philip Major</td>
<td>HQ</td>
<td>1 copy</td>
</tr>
<tr>
<td>Ms. Karen H. Weber</td>
<td>HQ</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Jack Kramer</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Robert I. Winner</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. John Salasin</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Cathy J. Linn</td>
<td>CSED</td>
<td>100 copies</td>
</tr>
<tr>
<td>Dr. Joseph L. Linn</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Mr. Michael R. Kappel</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Mr. Howard Cohen</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Ms. Deborah Heystek</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Reginald Meeson</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Karen Gordon</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Norman Howes</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Dennis Fife</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Dr. Cy Ardoin</td>
<td>CSED</td>
<td>1 copy</td>
</tr>
<tr>
<td>Ms. Julia Sensiba</td>
<td>CSED</td>
<td>2 copies</td>
</tr>
<tr>
<td>Albert J. Perrella, Jr.,</td>
<td>STD</td>
<td>1 copy</td>
</tr>
<tr>
<td>IDA Control &amp; Distribution Vault</td>
<td></td>
<td>3 copies</td>
</tr>
</tbody>
</table>