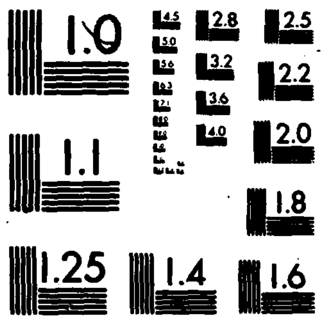
AD-A092 260 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
AN ADA LANGUAGE MODEL OF THE AN/SPY-1A COMPONENT OF THE AEGIS W--ETC(U)
AUG 80 E E MCCOY
UNCLASSIFIED NPS52-80-011 NL

END
DATE
FILMED
1-81
DTIC



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

NPS52-80-011

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AN ADA LANGUAGE MODEL OF THE
AN/SPY-1A COMPONENT OF THE
AEGIS WEAPON SYSTEM

by

Earl E. McCoy

August, 1980

Prepared for
Naval Postgraduate School
Monterey, CA 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

Jack R. Borsting
Provost

Reproduction of all or part of this research is authorized.

Earl E. McCoy

EARL E. McCOY
Visiting Assistant Professor of
Computer Science

Reviewed by:

[illegible]

G. H. BRADLEY, Chairman
Department of Computer Science

Released by:

William M. Tolles

W. M. TOLLES
Dean of Research

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 NPS52-80-011 /	2. GOVT ACCESSION NO. AD-A092260	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 AN ADA LANGUAGE MODEL OF THE AN/SPY-1A COMPONENT OF THE AEGIS WEAPON SYSTEM.		5. TYPE OF REPORT & PERIOD COVERED 9 Technical Report.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Earl E. / McCoy		8. CONTRACT OR GRANT NUMBER(s) 12 44
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS SEATASK Project No. 80-109
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE 11 Aug[redacted]80 /
		13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ADA, Specification Languages, Modeling, Systems Design, SPY-1, AEGIS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An ADA-like language is proposed as a high level, specification oriented modeling tool. It is asserted that the very early system design modeling tasks are typically not given adequate stress, with the result that poor system designs are carried forward into the mid design phases. A lack of suitable modeling tools is likely one reason, and so an ADA-like modeling technique is proposed. It has many of the properties of specification languages, including the ability to be machine processed to indicate incomplete or inconsistant (con't)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251450 [illegible]

systems. The unclassified portions of the SPY-1 radar component of the AEGIS weapon system is used as a test vechicle to illustrate the modeling technique.

ABSTRACT

An ADA-like language is proposed as a high level, specification oriented modeling tool. It is asserted that the very early system design modeling tasks are typically not given adequate stress, with the result that poor system designs are carried forward into the mid design phases. A lack of suitable modeling tools is likely one reason, and so an ADA-like modeling technique is proposed. It has many of the properties of specification languages, including the ability to be machine processed to indicate incomplete or inconsistant systems. The unclassified portions of the SPY-1 radar component of the AEGIS weapon system is used as a test vechicle to illustrate the modeling technique.

I Purpose of the Research

This research evaluates the use of the new DoD programming language ADA [Ichbiah79, Wegner80] (or an ADA-like language) as a modeling technique for large scale multiprocessing computer systems. In particular, the research is to evaluate the effectiveness of such a modeling technique vis-a-vis other, more conventional techniques such as flowcharts, block diagrams, and narratives. The measure of effectiveness is a qualitative assessment of its ease of use and understandibility by designers and "customers" (the purchasers of the modelled system).

II Background

The ADA language is currently undergoing final definition as the new standard DoD embedded systems language. ADA, when adopted, will be the standard language for programming computers that are components in weapon systems. The language incorporates the latest concepts in algorithmic language design, including modern control structures, user-defined data types, and the ability to coordinate concurrently executing "tasks". As of this writing no production ADA compiler exists, and so the ADA programs included in this report have not been verified as being syntactically or semantically correct.

A major problem with all large scale systems (multiprocessing systems like AEGIS in particular) is the early design phase in which customer requirements are specified. Typically system development is retarded and budgets exceeded because the system is incompletely or incorrectly specified.

One reason a system may become incompletely or incorrectly specified is the lack of sufficient communication between the designers of the system (technical persons) and the buyer or customer of the system (the persons with the problem the system is meant to resolve). Due to their different knowledge of the problem these two groups have a different perception and understanding of the problem, and yet the design of a successful system largely depends upon how closely the perceptions can be brought together. Unfortunately, because of these misperceptions, the designers proceed with a system that satisfies the problem as they understand it, and the customer is unaware of this because of the lack of communication. At some point the difference is discovered, but usually after much effort has been wasted. Better communication is a solution to this problem, and a candidate mechanism for this purpose is a modeling technique more understandable by both the designers and the customers. Computer science specialists, known as software engineers, have been and are developing tools to deal with this problem [Teich77, Davis79, Jones79]. Collectively these tools are called "specification languages", and are usually interactive programming-like languages that can be computer processed to indicate specification related problems.

It is asserted here that ADA (or an ADA-like language) can be used as a specification tool. This results from the ability of ADA to allow top-down program development, in which functionality can be omitted at a high level, only to be incorporated at a later time. In addition, ADA incorporates control structures for multitasking, in which concurrently executing tasks are coordinated in a specified way. Information may be passed among the tasks in a manner which is independant of the underlying computer architecture (shared memory or distributed processing). If ADA proves useful in this role the conversion of the specifications into executable code would be less costly and quicker than otherwise possible, a worthwhile objective.

Other commonly used programming languages, such as PL/I, Fortran, or CMS-2, could also be a basis of a modeling tool. However ADA has most of the good features of those languages and some additional features as well. This is not unexpected as ADA is meant to be a contemporary refinement of all that is known about programming languages. For example ADA's multitasking capability is a more modern mechanism than the equivalent in PL/I. Thus ADA appears to be the best notation upon which to build a high level modeling tool.

In the remaining sections of this report the derivation of an ADA-like model of SPY-1 is accomplished. The narrative accompanying the derivation is meant to exhibit the derivation thought process, and assumes at least some familiarity with contemporary programming language practices and concepts.

III The Research Vehicle

The SPY-1 radar component of the AEGIS weapon system is chosen as a research vehicle for several reasons. First, it is an ongoing research project at the Naval Postgraduate School, where three faculty members and several graduate students are investigating a microprocessor based multiprocessing implementation of SPY-1. Secondly another faculty member (the author) has the requisite knowledge of ADA and is interested in the research topic. Fortunately enough unclassified documentation of SPY-1 exists for meaningful research to be carried out. Finally the project is small enough to allow current funding to be used.

The source of information for the research is almost entirely from a classified report [WS-10544] with all the classified pages removed. The omitted information results in some gaps in the study, but not enough to affect the modeling effort. In the remaining sections this document is referred to as the "referenced document" or "referenced source".

In the next section an overall summary of the SPY-1 radar system and its interface with other AEGIS components is presented.

IV The SPY-1 Radar

This section describes the SPY-1 radar system at the highest level. The information is derived from the document referenced above, consisting mostly of a narrative with accompanying tables and diagrams.

At the highest level the SPY-1 interfaces with other AEGIS components, namely a Command and Control System, a Weapons Control System, an Operational Readiness Test System, a Missile

Datalink System, and a Gun Fire Control System. Interfaces are defined to these components in the form of data structures; in some cases the details are classified. In addition an interface with the AEGIS Tactical Executive System is maintained. Listing 1 shows how ADA might be used to describe this level of detail.

```
PACKAGE spy_1 IS
   USE interface_types_package; -- global messages
   PROCEDURE c&d_interface ( parameters );
   PROCEDURE wcs_interface ( parameters );
   PROCEDURE orts_interface ( parameters ); -- classified
   PROCEDURE gfcs_interface ( parameters ); -- classified
   PROCEDURE ates_interface ( parameters ); -- classified
PRIVATE
    -- declaration of all private data types
    -- and private data objects, not important
    -- at this point in the modeling effort
END spy_1;
```

Listing 1. An ADA package specification for SPY-1.

The reader is reminded that the ADA-like model derived in the following paragraphs only represents a logical explanation of the SPY-1 radar system, and that no particular underlying computer architecture need be assumed. Thus it is not meaningful (at this stage of modeling) to think of procedures or tasks as being dynamically loaded, etc.; the essential point is that they are invoked when needed and execute according to the ADA language semantics.

Before discussing this ADA-like code, the notion of a "package" is defined. In ADA a package is a means of grouping related items together, typically data object declarations and subprogram declarations. A package is referred to, within the code of some other programming unit, by means of a USE statement that effectively inserts the "packaged" statements. Thus a package is not directly executable, but rather contains executable units (procedures and/or tasks) that may be invoked from outside the package. It may, however, have an executable initialization component. A package may be divided into two parts, a "specification" part that is "visible" to the external environment, and a "body" which is hidden from the external environment. Thus the "user" of the package can only access the material exhibited in the specification part.

The code shown is not exactly ADA; the word "parameters" would have to be replaced by the actual data objects being passed. Double dashes indicate the start of comments that continue until the end of the line. Not shown (because it is not relevant at this time) is the declaration of private data types and objects; such types and data objects would be known to the users of the SPY-1 package but their underlying structural components would not be accessible. The definition of the various procedures will be shown elsewhere.

How is the SPY-1 package used in a real ADA programming environment? It would be embedded in another "AEGIS package", where it ould be one of several packages, along with other code. This is beyond the scope of the research project and so is not pursued further, except to point out the various procedures would be invoked from within the encompassing AEGIS package.

The details of the messages that can be sent and received via the procedures and the corresponding procedure bodies are described in a corresponding package "body", as is shown in Listing 2. This is a companion component the specification package shown in Listing 1; together they will completely define the SPY-1 package.

```
PACKAGE BODY spy_1 IS
 -- Local type and object declarations and
 -- definitions of the five procedures whose
 -- headers are in the package specification
 -- shown in Listing 1. These would, in turn,
 -- call entry points in the task immediately
 -- following.
   PROCEDURE orts_interface (...) IS
      ...     -- invoked from an ORTS task
     BEGIN
      ...
      initialization_command; -- invokes the
           -- spy_1_control task
      ...
   END orts_interface;
   TASK spy_1_control IS
      -- procedure and entry declarations, etc.
      TYPE ...   -- message type declarations
      ENTRY initialization_command;
      ENTRY task_command; -- other entries symbolically
   END spy_1_control;
   TASK BODY spy_1_control IS
      -- local declarations
     BEGIN
      ACCEPT initialization_command; -- from the
            -- ORTS interface procedure shown above.
      LOOP
          SELECT
             ACCEPT task_command; -- others
             . . .
          END SELECT;
      END LOOP;
   END spy_1_control;
 BEGIN                        -- initialization part
   INITIATE spy_1_control;    -- starts this task
END spy_1;
```

Listing 2. ADA representation of SPY-1 package body.

The program shown in Listing 2 is a very high level representation of the overall SPY-1 system, and so is described in some detail in the following paragraphs. Note that within the SPY-1 package body the executable portion consists of at least one statement (between the package body BEGIN and END delimiters), namely "INITIATE spy_1_control". In the declarative portion of the package body is defined the "spy_1_control" task that is started by the initiate command. This task will control the overall execution of the SPY-1 system, doing most of what the "ates" component would do in the current SPY-1 system. By definition a task is a procedure that executes in parallel with other procedures. They can be initiated (started) or aborted (stopped), and can be made to coordinate their activities with other procedures.

Thus a controlling task is started from the SPY-1 package body, and so execution of the task body also shown in Listing 2 is initiated. The first statement within the task body is an "ACCEPT initialization_command" statement. An ACCEPT statement forces a "rendezvous" with another task or procedure that is executing a corresponding "call" statement. The label "initialization_command" is an entry point in the spy_1_control task, and is called from one (or more) of the procedures specified in Listing 1. As shown in Listing 2 the entry point is called indirectly from a task in ORTS (Operational Readiness Test System, another component of the AEGIS system). Until the rendezvous is accomplished the calling procedure is halted; in this case the spy_1_control task tests for an appropriate initialization command (indirectly received from ORTS); it does not proceed until the command arrives. In the example shown no messages (in the form of a parameter list) are passed between calling and called tasks; this additional detail will be added later.

When the initialization command is received the rendezvous is said to have occured, and after some processing in the spy_1_control task both the calling and called tasks or procedures are allowed to proceed independently of each other. As shown in Listing 2, this results in the entering of an infinite loop (LOOP --- END LOOP statements). It is not really an infinite loop because the loop body contains other ACCEPT statements, shown symbolically by the "ACCEPT task_command" statement. In this manner other SPY-1 tasks are initialized, coordinated, and aborted. These other tasks are discussed in a following section.

V The Rendezvous Concept

It is important that the rendezvous concept be fully understood before proceeding. Consider two programs (modeling some procedures) that require interaction to accomplish some common goal. Typically one program may want to invoke the other program to carry out some specific job on its behalf.

Several mechanisms for accomplishing this interaction have been devised. ADA adopts a "rendezvous" concept, in which a calling program must know something about the program it wishes to invoke, but in which the called program need not know anything about the calling program. Thus the mechanism is one sided in that sense. This is clearly superior to a symmetrical situation where both parties to the interaction must know of each other; the rendezvous concept allows a library of programs to exist and be called as needed, whereas the symmetrical mechanism does not.

Consider the situation in which a data structure represents some entity that must undergo certain processing during its existence. In a grocery store context a customer may require the services of a checkout stand, for example. Here the customer must enter a queue (possibly empty) to await his turn for service. While waiting the customer is necessarily idle, in the sense he can not do additional shopping or anything else. When prior customers have been serviced the particular customer may be processed (a modification of the data structure) and then released to some other environment. In the modeling technique used here we say a customer attempts a rendezvous with the server; if the server is not available then the calling entity must wait.

The rendezvous technique provides additional mechanisms allowing priority queues, pre-emption, and availability of a set of resources to determine service response. None of these is discussed in this report. However the rendezvous concept allows classical problems such as the consumer-producer, reader-writer, dining philosophers, etc. to be modelled in a straightfoward manner.

Again the reader is reminded that nothing in the notation dictates anything about the underlying implementation of the model on a real set of computers. Thus it is not meaningful to consider (at this stage) when or how programs are executed; they are just executed according to the semantics of the ADA-like language.

VI The SPY-1 Control Task

In this section the overall structure of the SPY-1 radar control mechanism is described.

The situation to this point is as follows: within the SPY-1 package a "spy_1_control" task is initiated and waits for an initialization command from one of the specified procedures. When an initialization command arrives the controlling task resumes execution and enters an infinite loop that contains a SELECT statement that selects among any rendezvous request corresponding to the ACCEPT statements. In this manner the various component tasks within the SPY-1 system are initiated, aborted, or coordinated as necessary. The various messages passed among the tasks have not yet been explicity shown.

We now address the internal structures of the SPY-1 radar system. It can be thought of as consisting of various tasks, executing in parallel but in a coordinated fashion, each sending and receiving certain messages. Most of this message passing and coordination is strictly from within the SPY-1 system, but in addition there is the external communication and coordination with the other AEGIS system components. It is the latter that is of primary interest here; we are most concerned in modeling the interface of SPY-1 with these other components, and in particular the logical relationships of the interfaces and the internal tasks. Recall that it is precisely these relationships that are significant in the early design stages, and that it is a premise of this research project that conventional notations are weak in this regard, and that the modeling technique used here would be an improvement.

SPY-1 must interface with five external AEGIS components as is shown by Listing 1: command and decision, weapons control, operational readiness and testing, gun fire control, and AEGIS tactical executive systems. Each of these interfaces consists of several different message types. Each different message type requests some particular service from the SPY-1 radar system. One example was already illustrated in the previous section by the initialization command indirectly from ORTS. Consider now how these messages might be modelled in an ADA-like notation.

The referenced source document indicates how the various messages are transmitted through the several interfaces; here we concentrate upon the unclassified ones. For example Tables 3.4.17-I and 3.4.17-III in the reference document indicate the inputs and outputs, respectively, of the command and decision interface. Each of the message types has a description of its message content fields, scaling precision, data rate, conditions, operator intervention requirement, and its destination or source; these are now modelled in an ADA-like notation.

According to the reference document, one of the message content fields is a "message type" field; presumably it is this field that designates any particular message as containing data of a particular format. In the modeling technique employed here it is only necessary to use the contents of this field to characterize the record. This can be accomplished by using a record structure of the "variant" type, in which a particular value of a variable indicates the record structure that follows. In this SPY-1 model we choose to define separate record types for each interface, and indeed for each direction of data flow within each interface. This is done so that the semantics of the modeling notation can be used to our advantage, namely the ability of "strong type checking" to enforce only legal sending and receiving of messages.

```
TYPE c&d_interface_input IS
   RECORD
     kind : (track_data, data_ack, burnthrough_rpt,
             track_acc_rej, radar_status, burnthrough_acc_rej,
             redundant_track, radar_load_status, rsc_status,
             radar_doctrine);  -- an enumerated data type
     CASE kind OF
        WHEN track_data =>
             ...   -- various field declarations
        WHEN data_ack =>
             ...   -- varous field declarations
        .
        .
        .
        WHEN radar_doctrine =>
             ...   -- various field declarations
     END CASE;
  END RECORD;
```

Listing 3. A variant record declaration.

An example of the variant record structure mechanism is shown in Listing 3. The record structure consists of a field indicating the kind of message stored in the remaining part of the structure, in this situation depending upon the value of the variable "kind"; each "when clause" declares the fields within each message type (not shown in Listing 3; see the appendix for a complete example). Similar record structures are defined for the command and decision output messages, and for all the other interfaces.

The record definition described above is of a type of record. In any particular procedure or task variables of that type may be declared and then manipulated as appropriate. For example, in a sending procedure the declaration of a variable "message" (by a "message : c&d_interface_input" statement) might be used to define a particular data structure in which its component fields can be assigned specific values. The message could be sent by invoking an appropriate procedure:

```
  c&d_user_services_input (message : OUT c&d_interface_input);
```

Here a procedure or task entry name is written followed by its output argument list; the "user services" notation is from the reference document. The direction of data flow is "out" from the calling procedure's point of view. The "c&d_user_services_input" entry name (or point) must be declared somewhere in an appropriate task or procedure.

Before revising Listing 1 to include message parameter detail, a convenient form of grouping related information is employed that enables only important details to be exhibited at a particular place. The less important detailed information is placed elsewhere. Listing 4 shows a package of type statements, each type declaration is a record definition similar to that shown in Listing 3. Details of the variant record structure are omitted.

Listing 5 shows a redefinition of the SPY-1 package specification shown in Listing 1. Here particular procedure headings are defined to include the interfaces in both directions. This allows messages to be sent in either direction without any interdepen-

```
PACKAGE interface_types_package IS
  TYPE c&d_interface_input IS
    ...    -- a variant record structure
  TYPE c&d_interface_output IS
    ...
  TYPE wcs_interface_input IS
    ...
  TYPE wcs_interface_output IS
    ...
  TYPE orts_interface_input IS
    ...
  TYPE orts_interface_output IS
    ...
  TYPE gfcs_interface_intput IS
    ...
  TYPE gfcs_interface_output IS
    ...
  TYPE ates_interface_input IS
    ...
  TYPE ates_interface_output IS
    ...
END interface_types_package;
```

Listing 4. A package of data type definitions.

dence upon each other. These ten procedures replace the five procedures of Listing 1. A corresponding redefinition of the procedure bodies (which are enclosed within the package body) would be required. Within each procedure body a message type would be determined and an appropriate entry point of the controlling task called, where in turn other appropriate task entry points may be called, thereby forcing appropriate SPY-1 services to be accomplished.

Again the private type definitions are omitted because they represent internal data structures not needed by the interface with the other AEGIS components. Note the use of the USE statement to enable the bulky definitions to be placed elsewhere (in a separate package).

```
PACKAGE spy_1 IS
  USE interface_types_package;
  PROCEDURE c&d_user_services_input
                       (m : IN c&d_interface_input);
  PROCEDURE c&d_user_services_output
                       (m : OUT c&d_interface_output);
  PROCUDURE wcs_user_services_input
                       (m : IN wcs_interface_input);
  PROCEDURE wcs_user_services_output
                       (m : OUT wcs_interface_output);
  PROCEDURE orts_user_services_input
                       (m : IN orts_interface_input);
  PROCEDURE orts_user_services_output
                       (m : OUT orts_interface_output);
  PROCEDURE gfcs_user_services_input
                       (m : IN gfcs_interface_input);
  PROCEDURE gfcs_user_services_output
                       (m : OUT gfcs_interface_output);
  PROCEDURE ates_user_services_input
                       (m : IN ates_interface_input);
  PROCEDURE ates_user_services_output
                       (m : OUT ates_interface_output);
PRIVATE
    -- declaration of private data types
END spy_1;
```

Listing 5. A revision of Listing 1.

We now address the internal tasks of SPY-1. Previously a "spy_1_control" task was defined and indicated that it controlled and coordinated other tasks by means of ACCEPT statements embedded within an infinite loop.

Section 3.3.5.2 of the reference document lists the various functions performed by the SPY-1 radar. These functions are divided into two groups: a "tactical function group" and an "element test function group". These are shown in Tables 1 and 2, respectively.

Listing 6 shows a skeleton representation of a SPY-1 function in the form of a task. Entry points (not explicitly shown) indicating how external tasks may interact with the

Initialization
Search Management
Track Management
Frequency Management
Radar Scheduling
Cross Gating
Beam Stabilization
Radar Input Processing
Radar Output Processing
Detection Processing
Track Processing
ECM/Clutter Processing
Track Association
Load Evaluation
Missile Communications
Historical Recording
Switch Action and Display Processing
Video Formatter

Table 1. SPY-1 Tactical Functions.

"search_management" task are of crucial importance; the reference document (or at least the unclassified part) does not indicate these details. Similar tasks would be defined for all the functions listed in Tables 1 and 2.

It is not important at this stage of modeling to know the details of how these functions work; the statement of what must be accomplished is of higher priority at the early design stages. However it is the correct interaction of these functions that is the next highest priority, certainly higher priority than the details of their algorithms. For this reason we now address the interaction of these functions.

The ramification of modeling the interactions is the definition in the ADA-like notation of task entry points that can be called in the SELECT statement in the "spy-1-control" task of Listing 2. The placement of the calls to these other tasks within the various select clauses describes the manner of interaction

EFT Control
Dynamic Test Targets
Operability and Performance Testing
Angle Calibration Testing
On-Line Scan Tests
Transmitter Power and Phase Test
Signal Processor Fault Isolation Support Testing
Nonmission Tests

Table 2. SPY-1 Element Test Functions

allowed, and the parameter lists indicate the transfer of messages between the tasks. Before describing the SPY-1 radar in this notation a review of the ADA notation is presented.

Recall that tasks execute in parallel with other tasks, and that they contain explicit entry points, and that each entry point can optionally have a parameter list that affects intertask communication. From within a particular task (or procedure) another task may be invoked or called, forcing it (the calling task) to halt execution until a rendezvous with the called task can legally occur. Meanwhile, in the called task, execution continues until the internal logic indicates that the current execution is completed. At that moment the calling and called tasks can legally attempt synchronization and, if possible, a rendezvous is said to occur. The SELECT statement in the called task allows the rendezvous to happen by selecting the corresponding ACCEPT statement clause to be executed; the select mechanism can force the calling task to remain halted while some particular code is being executed, after which both tasks may proceed on their independent ways. Possibly some additional code in the coordinating task may be executed prior to the next selection by the SELECT statement. The continuing selection process is

```
TASK search_management (...) IS
   -- include any specification details
   -- here such as type specifications,
   -- subprogram specifications, and
   -- entry point specifications.
END;
TASK BODY search_management IS
   -- local declarations
   BEGIN
      -- a select statement within an infinite
      -- loop accepts calls to this task's entry
      -- points, which when undertaken effect
      -- various actions on part of the SPY-1
      -- system.
END;
```

Listing 6. A representation of a SPY-1 function.

accomplished because (in this case) the SELECT statement is embedded in an infinite loop.

The reference document does not explicitly indicate the manner in which the various functions may legally interact, prohibiting the exact representation of this process in the ADA-like notation. Figures 3.3.5.3-3 through 3.3.5.3-7 in the reference document do indicate in block diagram form the flow of interfunction data (messages) among the various functions. At any rate the exact sequencing among the functions is believed classified and so would not be accessible to this study in any case.

However, since this part of the system model is so vital to the correct derivation of any complex system (such as SPY-1) and is so much an important point underlying the usefulness of the notation, a fictitious characterization of the interaction is presented. It is included to illustrate how the notation would appear, and is not to be construed to be an accurate representation of how the SPY-1 radar system really works. Listing 7 indicates how the various functional tasks might be coordinated by the ADA-like notation.

```
-- appears in the body of "spy_1_control" task
LOOP              -- an infinite loop structure
  SELECT          -- this statement selects among
                  -- the various "accept" statements,
                  -- each of which represents a task
                  -- entry point; note some are
                  -- are "guarded" by a "when" statement
    ACCEPT frequency_change_request (...) DO
          -- these statements are executed before
          -- the rendezvoused tasks are allowed
          -- to proceed independently
        END;
          -- these statements are executed before
          -- any other selection of a rendezvous;
          -- typically other tasks may be called
          -- upon to provide some kind of service.
    OR ACCEPT track_start_request (...) DO
          .
          .
        END;
          .
          .
    OR WHEN dwell_time_expired ACCEPT
        next_beam_request (...) DO
           -- this is an example of a guarded
           -- rendezvous; the condition must
           -- be true before a rendezvous may
           -- occur
          ...
          END;
          ...
      -- other guarded or unguarded accept statements
  END SELECT;
END LOOP;
```

Listing 7. Fictitious SPY-1 control task representation.

Within the SELECT statement a series of guarded and unguarded ACCEPT statements indicate the proper coordination and data transfer permitted among the various tasks. Guarded ACCEPT statements (having the WHEN predicate) permit a rendezvous only when the predicate is true. In general all tasks are executing in parallel, but are coordinated by the "spy_1_control" task of which Listing 7 is but a part. Because the SELECT statement is in an infinite loop, the controlling task is always "checking" for the next possible rendezvous. When more than one rendezvous is possible any particular one is selected at random. If no rendezvous is possible (because none is being requested or guard predicates are false) the controlling task just waits. Not shown in Listing 7 is a particular ACCEPT clause that would allow an ORTS task (or procedure) to abort or kill the SPY-1 radar system, effectively causing it to exit the infinite loop structure. An alternative notation for expressing the synchronization of tasks is presented in [Andler79]; it has the advantage of being more succinct.

```
ACCEPT frequency_change_request (m : IN c&d_interface_input) DO
   saved_message := m;
  END;      -- calling task or procedure may now continue
   -- interpret the message and invoke the appropriate
   -- set of internal tasks such that the request is
   -- carried out; fictitious tasks are illustrated here
  CASE saved_message.request OF
     WHEN phase_change =>
        change_phase (saved_message.rate); -- task invoked
        video_formatter ("phase changed"); -- to operator
     WHEN freq_change =>
        change_frequency (saved_message.value);
        video_formatter ("frequency changed");
     WHEN ...
     ...
  END CASE;
  -- this is the end of the accept clause
  -- another rendezvous may be selected
```

Listing 8. Fictitious detail added to a component of Listing 7.

A further fictitious expansion of the ACCEPT frequency_change_request clause portion of Listing 7 is shown in Listing 8. Again the general process is reviewed: some task external to the SPY-1 environment makes a request for a SPY-1 service by invoking one of the procedures specified in the SPY-1 package specification. For example the Command and Control component may request a frequency change by invoking the "c&d_user_services_input" procedure with an appropriate message argument. Within this procedure the message is interpreted and an appropriate SPY-1 internal function is invoked by calling an entry point in the spy_1_control task, whose sole purpose is to coordinate all such requests. When the spy_1_control task accepts the call (a rendezvous) the message is saved in a local variable, and then the calling procedure is released which in turn can release the task in the command and control component of AEGIS. Meanwhile, in the spy_1_control task the accept clause is continuing execution. As shown is Listing 8 the particular request may be determined by a CASE statement, where the appropriate WHEN clause can invoke some set of SPY-1 internal tasks (those corresponding to the functions listed in Tables 1 and 2). For example, in Listing 8, if the "freq_change" clause is executed two (fictitious) subtasks of the Frequency Management function are invoked, presumably adequate for carrying out the requested service. At this time the SELECT statement may select another pending request for service.

Even though the reference document does not contain enough unclassified information to permit the representation of this controlling task, it is restated again that it is precisely this aspect of the early design stages that is among the most essential to a successful design of a complex interactive system, and that the advantage of the ADA-like notation proposed here is that it allows such a representation.

VII Summary and Conclusions

This research study derived an ADA-like model of the SPY-1 radar system to the extent budgetary, reference documents, and security constraints permitted. The model is confined to the very highest level, for it is at this level that both system designers and system buyers (customers) must mutually understand their system problems. It is at this level that complex system designs usually deviate from the path that would lead directly to a successful system, primarily due to poor specification and understanding of the buyer's problem. The buyer is usually not in the position to prohibit the deviation because they are not (at that time) aware of it. By the time the existence of the deviation is known to either party, much time and effort is usually wasted, and additional time and effort is expended attempting to rectify the inappropriately designed system.

The assertion here is that a better high level modeling technique could at least alleviate the problem, by forcing the explicit concentration of effort on the proper specification of the system's high level design, and in a format easily understandable by both designers and the customers. An ADA-like notation is proposed, because it embodies those characteristics important to specification engineering, is (or will be) a notation familiar to both designers and customers, is amenable to machine processing to ascertain design completeness and consistency, and will be used (by DoD policy) in the actual coding of the logic.

Is the model presented here for the SPY-1 radar system a convincing argument for the usefulness of the notation? The goal of this research is not to definitively answer this question, but to illustrate the proposition, and to assess the utility of the technique in a nonquantitative manner. It is suggested here that the proposed technique does satisfy a nonquantitative assessment of its utility, by the clarity of the representation of SPY-1 as shown in the previous sections and again in the appendix. Only more examples of the method, done by several independent study groups and over a period of time, can provide data so that the usefulness of the technique can be adequately assessed. It is recommended that further study be undertaken for this purpose.

A further qualifying statement must be made. ADA is undergoing (at the time of writing) a final revision of its definition, and so it is possible (and even probable) that some portions of ADA illustrated here may not remain in the final definition of the language. This is one more reason for further study of the usefulness of the modeling technique proposed here.

BIBLIOGRAPHY

[Andler79] Andler, S., "Predicate Path Expressions; A High-Level Synchronization Mechanism", Carnegie-Mellon University Report CMU-CS-79-134, August 1979.

[Davis79] Davis, A. M., and T. G. Rauscher, "Formal Techniques and Automatic Processing to Ensure Correctness in Requirements Specifications", IEEE 1979 Specifications of Reliable Software, pp. 15-35.

[Ichbiah79] Ichbiah, J. D., et. al., "Preliminary ADA Reference Manual", Part A, and "Rational for the Design of the ADA Programming Language", Part B, ACM SIGPLAN Notices, Vol. 14, No. 6, June 1979.

[Jones79] Jones, C., "A Survey of Programming Design and Specification Techniques", IEEE 1979 Specifications of Reliable Software, pp. 91-103.

[Teich77] Teichroew, D., and E. A. Hershey III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Trans. on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 41-48.

[Wegner80] Wegner, P., Programming with ADA: Introduction by Means of Graduated Examples, Prentice-Hall, 1980.

[WS-10544] A classified report, of which only unclassified pages were referenced.

APPENDIX

This appendix lists the final form of the ADA-like model of the SPY-1 radar system developed over the course of the research. The extent of the model is limited due to the available reference document and to the amount of manpower used. The model is restated here to bring the various component parts of the SPY-1, as presented in the body of the report, into one place where a qualitative assessment as to its usefulness can more easily be made.

Subordinate packages used with the "spy_1" package are placed at the end of this appendix, where they can be easily referred to. The main package can be usefully studied with only occasional reference to the subordinate packages.

```
-- the following package is to be "used"
-- in an encompassing AEGIS package

PACKAGE spy_1 IS

  -- the following package contains
  -- type definitions for all messages
  USE interface_types_package;
  -- all the following procedures are callable from
  -- the encompassing AEGIS package and effect
  -- all SPY-1 services to AEGIS
  PROCEDURE c&d_user_services_input
            (m : IN c&d_interface_input);
  PROCEDURE c&d_user_services_output
            (m : OUT c&d_interface_output);
  PROCEDURE wcs_user_services_input
            (m : IN wcs_interface_input);
  PROCEDURE wcs_user_services_output
            (m : OUT wcs_interface_output);
  PROCEDURE orts_user_services_input
            (m : IN orts_interface_input);
  PROCEDURE orts_user_services_output
            (m : OUT orts_interface_output);
  PROCEDURE gfcs_user_services_input
            (m : IN gfcs_interface_input);
  PROCEDURE gfcs_user_services_output
            (m : OUT gfcs_interface_output);
  PROCEDURE ates_user_services_input
            (m : IN ates_interface_input);
  PROCEDURE ates_user_services_output
            (m : OUT ates_interface_output);
PRIVATE
  -- declaration of private data types
  -- probably not needed in this package
END spy_1;      -- the specification part
```

```
PACKAGE BODY spy_1 IS
 -- all local declarations
 USE interfunction_interfaces; -- a package
 PROCEDURE BODY c&d_user_services_input IS
    -- all local declarations
    BEGIN
    -- interpret the message and invoke appropriate
    -- SPY_1 control task entry points, where
    -- appropriate functional tasks will be invoked
    -- to carry out the request
    ...
    ...
       CASE m.kind OF
         WHEN change_mode =>
            -- request rendezvous with spy_1_control
            track_start_request (..); --possibly with argument
         WHEN change_status =>
            ...
          .
          .
          .
       END CASE;
       ...
   END c&d_user_services_input;
```

```
-- the remaining procedures are left as
-- separately compiled entities, not shown in appendix
PROCEDURE BODY c&d_user_services_output  IS SEPARATE;
PROCEDURE BODY wcs_user_services_input   IS SEPARATE;
PROCEDURE BODY wcs_user_services_output  IS SEPARATE;
PROCEDURE BODY orts_user_services_input  IS SEPARATE;
PROCEDURE BODY orts_user_services_output IS SEPARATE;
PROCEDURE BODY gfcs_user_services_input  IS SEPARATE;
PROCEDURE BODY gfcs_user_services_output IS SEPARATE;
PROCEDURE BODY ates_user_services_input  IS SEPARATE;
PROCEDURE BODY ates_user_services_output IS SEPARATE;


-- SPY-1 control task follows; schedules all
-- requests for services coming via the above procedures
TASK spy_1_control IS
  -- entry point definitions follow
  ENTRY initialization_command (m: IN orts_interface_input);
  ENTRY frequency_change_request (...);
  ENTRY track_start_request (...);
  ENTRY next_beam_request (...);
  -- all others entry declarations
  -- and other declarations
END spy_1_control;    -- the specification part
TASK BODY spy_1_control IS
   -- local declarations
  BEGIN
    ACCEPT initialization_command(...);
           -- indirectly from ORTS via
           -- orts_user_services_input procedure
           -- actually there is several kinds
           -- of initialization commands possible
           -- in SPY-1, but are not indicated here
    INITIATE search_management, track_management,
        radar_scheduling, cross_gating,
        beam_stabilization, radar_input_processing,
        radar_output_processing, detection_processing,
        track_processing, ecm_clutter_processing,
        track_association, load_evaluation,
        missile_communication, historical_recording,
        switch_action_display_processing,
        video_formatter, eft_control, dynamic_test_targets,
        operability_performance_testing,
        angle_calibration_testing, online_scan_tests,
        transmitter_power_phase_test,
        signal_processor_fault_isolation_support_testing,
        nonmission_tests;
        -- and whatever subtasks are needed
```

```
LOOP        -- an "infinite" loop entered
   SELECT   -- an accept clause for every
            -- entry point
      ACCEPT frequency_change_request
           (m : IN c&d_interface_input) DO
         saved_message := m;
       END; -- calling procedure may continue
         -- interpret the message and invoke
         -- the appropriate set of internal
         -- tasks such that the request is
         -- carried out; fictitious tasks
         -- are illustrated here
         CASE saved_message.request OF
           WHEN phase_change =>
             change_phase (saved_message.rate);
             video_formatter ("phase changed");
           WHEN freq_change =>
             change_frequency (saved_message.value);
             video_formatter ("frequency changed");
           WHEN ...
             .
             .
             .
         END CASE;
      OR ACCEPT .....
      OR ACCEPT .....
      OR WHEN .... ACCEPT .....
      .
      .
      .
   END SELECT;
END LOOP;
```

```
        -- remaining tasks are left as separately
        -- compiled units not shown in appendix
        TASK search_management                IS SEPARATE;
        TASK track_management                 IS SEPARATE;
        TASK frequency_management             IS SEPARATE;
        TASK radar_scheduling                 IS SEPARATE;
        TASK cross_gating                     IS SEPARATE;
        TASK beam_stabilization               IS SEPARATE;
        TASK radar_input_processing           IS SEPARATE;
        TASK radar_output_processing          IS SEPARATE;
        TASK detection_processing             IS SEPARATE;
        TASK track_processing                 IS SEPARATE;
        TASK ecm_clutter_processing           IS SEPARATE;
        TASK track_association                IS SEPARATE;
        TASK load_evaluation                  IS SEPARATE;
        TASK missile_communications           IS SEPARATE;
        TASK historical_recording             IS SEPARATE;
        TASK switch_action_display_processing IS SEPARATE;
        TASK video_formatter                  IS SEPARATE;
        TASK eft_control                      IS SEPARATE;
        TASK dynamic_test_targets             IS SEPARATE;
        TASK operability_performance_testing  IS SEPARATE;
        TASK angle_calibration_testing        IS SEPARATE;
        TASK online_scan_tests                IS SEPARATE;
        TASK transmitter_powera_phase_test    IS SEPARATE;
        TASK signal_processor_fault_isolation_support_testing
                                              IS SEPARATE;
        TASK nonmission_tests                 IS SEPARATE;
        -- and whatever subtasks are necessary


   BEGIN       -- initialization part of package
        INITIATE spy_1_control;
 END spy_1;    -- end of complete SPY-1 package
```

```
  -- the interface message type package
  -- all the following is from the reference
  -- document; detailed to the extent given there
  -- in some cases the detail varys slightly
  -- from the assumptions used in the model


PACKAGE interface_types_package IS

 TYPE c&d_interface_input IS
  RECORD
   kind : (track_data, data_ack, burnthrough_rpt,
    track_acc_rej, radar_status, burnthrough_acc_rej,
    redundant_track, radar_load_status, rsc_status,
    radar_doctrine);
   CASE kind OF    -- question marks indicate unknown types
     WHEN track_data =>
       no_of_tracks : ???;
       ctsl : ???;
       control_grp_trk_no : ???;
       weapon_control_indexes : ???;
       time_lag : ???;
       amplitude_estimates : ???;
       parent_ctsl : ???;
       type_track_indicators : (real, simulated);
       position : ARRAY (coordinates) OF REAL;
       velocity : ARRAY (coordinates) OF REAL;
       slant_range : ???;
       slant_range_rate : ???;
       lost_track_indicator : ???;
       bearing_data : ???;
       elevation_data : ???;
       track_coast_count : ???;
       track_indicators : (old, new);
       track_mode_indicators : (active, passive,
              coverpulse, missile);
        missile_indicator : ???;
        report_type : (normal, special_threat);
  WHEN data_ack =>
       ctsl : ???;
       control_gpr_trk_no : ???;
       range_rate : ???;
       status : (enable, disable);

       min_bearing, max_bearing : ???;
       min_range, max_range : ???;
    WHEN burnthrough_rpt =>
         console_address : ???;
         ctsl : ???;
         control_grp_trk_no : ???;
         time_tag : ???;
         elevation : ???;
         bearing : ???;
         target_ranges : ARRAY (no_of_targets) OF REAL;
    WHEN track_acc_rej =>
         ctsl : ???;
         response_code : (accept, busy, illegal);
    WHEN rsc_status =>
         control_grp_status : (start_up, ready);
         radar_status : (fore_off, fore_standby,
            fore_ready, fore_radiate, aft_off,
            aft_standby, aft_ready, aft_radiate);
    WHEN burnthrough_acc_rej =>
         c&d_trk_index : ???;
         control_grp_trk_no : ???;
         response_code : (accept, busy, illegal);
    WHEN redundant_track =>
         ctsl : ???;
         control_grp_trk_no : ???;
    WHEN radar_load_status =>
         trk_file_percent_load : ???;
         radar_percent_load ; ???;
         control_grp_percent_load : ???;
         time_tag : ???;
         search_frame_time : ???;
         radar_status : ???;
         percent_track_time : ???
     WHEN rsc_status =>
         console_address : ???;
         type_reply : (good, bad, missed);
     WHEN radar_doctrine =>
         -- AN/SPY-1 doctrine (not given)
   END CASE;
END RECORD;
```

```
TYPE c&d_interface_output IS
  RECORD
   kind : (track_id, auto_mode_parameters,
     cease_rpt_drop_trk, trk_acquisition_req,
     ships_speed, burnthrough_req, display_req,
     c&d_doctrine_control, passive_trk_data,
     alert_pending);
   CASE kind OF
     WHEN track_id =>
       ctsl : ???;
       weapon_control_index : ???;
       radar_set_trk_no : ???;
       category : (air, surface, clutter);
       id_class : (confirmed_hostile,
          assumed_hostile, unknown, assumed_friendly,
          confirmed_friendly, controlled_friendly);
       tactically_significant : ???;
     WHEN auto_mode_parameters =>
       min_range_rate : ???;
       ending_bearing, starting_bearing : ???;
       min_range, max_range : ???;
       status : (enable, disable);
     WHEN cease_rpt_drop_trk =>
       ctsl : ???;
       control_grp_trk_no : ???;
       weapon_control_index : ???;
       drop_trk_indicator : ???;
     WHEN track_acquisition_req =>
       time_tag : ???;
       dimensional_indicator : (two_d, three_d);
       position : ARRAY (coordinates) OF REAL;
       velocity : ARRAY (coordinates) OF REAL;
       ctsl : ???;
       weapon_control_index : ???;
       acquisition_indicator : (active, passive,
           surface);
       asimuth_extent : (normal, wide);
    WHEN ships_speed =>
       x_dot, y_dot : ???;
```

```
      WHEN burnthrough_req =>
         ctsl : ???;
         control_grp_trk_no : ???;
         request_type : (high_vel, low_vel);
         start_range : ???;
      WHEN display_req =>
         console_address : ???;
         video_formatter_type : ???;
         header_code : ???;
         display_mode : (above_horiz, horiz_clear,
            horiz_mti);
      WHEN c&d_doctrine_control =>
         radar_silence : (on, off);
      WHEN passive_trk_range_data =>
         ctsl : ???;
         radar_trk_no : ???;
         range : ???;
         range_rate : ???;
         time_tags : ???;
         data_source_id : ???;
         weapon_control_index : ???;
         associated_patl : ???;
         coast_count : ???;
      WHEN alert_pending =>
         -- notice of alert
    END CASE;
  END RECORD;
```

```
PACKAGE interfunction_interfaces IS

  -- these are typical of the data structures
  -- used for internal SPY-1 function communication,
  -- and are included here for illustrative purposes
  TYPE initialization_interface_input IS
    RECORD
     kind : (initialization_orders, test_results_summary,
      radar_status_summary, gryo_status_summary,
      ates_initial_complete, transmitter_status,
      loop_closure_confirmation);
     CASE kind OF
      WHEN initialization_orders =>
         -- initialization switch actions
         -- details not known
      WHEN test_results_summary =>
         -- results of CAL, OPT, OLS
      WHEN radar_status_summary =>
         -- summary of radar operable units
      WHEN gyro_status_data =>
         -- gyro data converter and gyro status data
      WHEN ates_initial_complete =>
         -- indication that radar initialization
         -- can commence
      WHEN transmitter_status =>
         -- transmitter, status, i. e., standby
         -- ready. (requested and received)
      WHEN loop_closure_confirmation =>
         -- notification of first radar return
  END CASE;
END RECORD;
TYPE initialization_interface_output IS
  RECORD
    kind : (xmtr_state, etf_test_requests,
      request_start_gyro, radar_system_status);
    -- next page classified, more may follow
   CASE kind OF
     WHEN xmtr_state =>
       -- command to change xmtr state up or down
     WHEN etf_test_requests =>
       -- none given in reference source
     WHEN request_start_gyro =>
       -- request to start gyro module
     WHEN radar_system_status =>
       -- control group status/radar system status
       -- dedicated etf-test flag or normal etf-
       -- test priority
   END CASE;
END RECORD;
```

```
 TYPE frequency_management_input IS
  RECORD
    kind : (beam_position, target_information,
     doctrine, hardware_operability,
     missile_information);
   CASE kind OF   -- question marks mean unkown type
     WHEN beam_position =>
      bearing : ???;
      elevation : ???;
      sector : ???;
      subsector : ???;
    WHEN target_information =>
      range_rate : ???;
      mode : ???;
      mti_pri_index : ???;
    WHEN doctrine =>
      -- frequency doctrine
      fixed : ???;
      prelook : ???;
      random : ???;
      -- sector doctrine
      min_max_bearing : ???;
      excluded_frequency_channels : ???;
      -- baseline doctrine
      excluded_frequency_channels : ???;
   WHEN hardware_operability =>
      -- frequency constrained by hardware
   WHEN missile_information =>
      uplink_freq, downlink_freq : ???;
 END CASE;
END RECORD;
```

```
 TYPE radar_scheduling_input IS
  RECORD
   kind : (search_lists_special, radiation_doctrine,
     time_sych, track_beam_scheduled,
     ships_motion_matrix, radar_silence,
     waveform_information, mode_status,
     xtmr_state, track_time_counter_reset);
  CASE  kind OF
   WHEN search_lists_special =>
     -- HS and AHS and special requests
     stc_data : ???;
     beam_position : ???;
     blanking_gates : ???;
     clutter_gates : ???;
     mode : ???;
     instrumented_range : ???;
     clutter_map_flag : ???;
     end_of_frame_flag : ???;
   WHEN radiation_doctrine =>
     -- relative bearings of radiation
     -- inhibit sectors
     power_option : (high, low, off);
   WHEN time_sync =>
     -- resynchronization time
   WHEN track_beam_scheduled =>
     -- track numbers by priority
     -- track file
   WHEN ships_motion_matrix =>
     -- stable deck orientation
     -- azimuth limits for search & track
     -- azimuth limit slope
   WHEN radar_silence =>
     -- radiation inhibit
   WHEN waveform_information =>
     -- frequency channel and bands
     -- phase code pri and dwell time for mti
   WHEN mode_status =>
     -- waveform substitution
   WHEN xmtr_state =>
     -- xmtr state
   WHEN track_time_counter_reset =>
     -- reset for track time counter
  END CASE;
 END RECORD;
 -- all the remaining interfunction interfaces
END interfunction_interfaces;
```

INITIAL DISTRIBUTION LIST

1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Earl E. McCoy Department of Computer Science Code 52My Naval Postgraduate School Monterey, California 93940	10
4.	Uno R. Kodres Department of Computer Science Code 52Kr Naval Postgraduate School Monterey, California 93940	10
5.	Department of Computer Science Code 52 Naval Postgraduate School Monterey, California 93940	30