NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A275 024

THESIS

A CONCURRENT, OBJECT-BASED IMPLEMENTATION FOR THE TACTICAL LEVEL OF THE RATIONAL BEHAVIOR MODEL

by

Frederick Perry Boynton Thornton, Jr.

September 1993

Thesis Advisor: Dr. Se-Hung Kwak

Approved for public release; distribution is unlimited.

94-02776

94 1 26 204

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
A Concurrent, Object-Based Implementation for the Tactical Level of the Rational Behavior Model(U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Thornton Jr., Frederick Perry Boynton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified/Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The middle, or Tactical, level of the Rational Behavior Model (RBM) is the essential bridge linking the top and bottom levels of the model together. To insure an autonomous vehicle maintains control and thus exhibits rational behavior during such time-consuming tasks as search, homing, and route replanning, the Tactical level must be able to handle concurrency. Until now, this level has been implemented in only a limited way using an object-oriented language and sequential operations. The objective of this thesis is to construct an implementation model that represents the concurrency inherent in the Tactical level within the framework of the design model already developed.

The method for building this implementation is to use the Ada *task* construct for concurrency to represent the objects of the design model and their communication with each other.

This research creates a Tactical level implementation in Ada for the NPS Autonomous Underwater Vehicle (AUV) simulator that successfully executes a mission scenario involving *transit*, *search*, *task*, and *return* phases and the same mission scenario with route replanning. This work thus provides a foundation for future development of concurrent implementations of this level of RBM.

14. SUBJECT TERMS
Concurrency, Multitasking, Object-Based, Object-Oriented, Rational Behavior Model, Tactical Level, Autonomous Underwater Vehicle

15. NUMBER OF PAGES
142

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

A CONCURRENT, OBJECT-BASED IMPLEMENTATION FOR THE TACTICAL LEVEL OF THE RATIONAL BEHAVIOR MODEL

by

Frederick Perry Boynton Thornton, Jr.
Captain, United States Marine Corps
B.A., Duke University, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: _____
Frederick Perry Boynton Thornton, Jr.

Approved By: _____
Dr. Se-Hung Kwak, Thesis Advisor

_____
Dr. Robert B. McGhee, Second Reader

_____
Dr. Ted Lewis, Chairman,
Department of Computer Science

ii

ABSTRACT

The middle, or Tactical, level of the Rational Behavior Model (RBM) is the essential bridge linking the top and bottom levels of the model together. To insure an autonomous vehicle maintains control and thus exhibits rational behavior during such time-consuming tasks as search, homing, and route replanning, the Tactical level must be able to handle concurrency. Until now, this level has been implemented in only a limited way using an object-oriented language and sequential operations. The objective of this thesis is to construct an implementation model that represents the concurrency inherent in the Tactical level within the framework of the design model already developed.

The method for building this implementation is to use the Ada *task* construct for concurrency to represent the objects of the design model and their communication with each other.

This research creates a Tactical level implementation in Ada for the NPS Autonomous Underwater Vehicle (AUV) simulator that successfully executes a mission scenario involving *transit*, *search*, *task*, and *return* phases and the same mission scenario with route replanning. This work thus provides a foundation for future development of concurrent implementations of this level of RBM.

iii

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

This work drew support from many quarters. First of all, I would like to acknowledge my wife, Neva, for her patience and support during many long nights in the lab and many distracted hours at home.

I would like to thank Dr. Ron Byrnes and Don Brutzman for laying the foundation for those of us working on the AUV project in the Computer Science department. Thomas Scholz contributed to the success of this project by helping to debug a large volume of our combined code and keeping a sense of humor in the process. Dr. Se-Hung Kwak provided helpful guidance and encouragement all along the way.

Without the contributions of all these people and many others, this work would not have been possible.

I. INTRODUCTION

A. BACKGROUND

Controlling autonomous vehicles through software is a challenging area of software engineering requiring a variety of resources. Neither completely relying on a single programming paradigm nor simply throwing together all available programming resources can provide the long-term stability necessary for an autonomous vehicle software system. A software architecture with multiple levels of abstraction is extremely important for handling the complexity of autonomous operations in the real world. Such an architecture provides for the use of specific programming paradigms to address particular levels of a problem. Reliability and maintainability of software then become key factors in determining the applicability of a programming paradigm to a certain level of abstraction, and they are built into the system instead of being produced incidentally.

To model the real world, autonomous vehicle software systems need to be capable of managing concurrency. Events, and thus behaviors, in the real world are neither sequential in time nor centralized in a single, physical entity. Concurrency involves the twin issues of multitasking, in which a single entity performs multiple operations at the same time, and distribution, in which many entities perform separate tasks simultaneously. In addition, reuse of software is very desirable in this complex development environment. The object-oriented programming paradigm with its built-in inheritance mechanism facilitates the reuse of existing implementations [Kwak90] [Toml89]. The capability to implement a concurrent, object-oriented solution is a powerful tool in accurately modeling the problem domain and an effective weapon in battling against software complexity.

B. STATEMENT OF THE PROBLEM

The Rational Behavior Model (RBM) is a multi-level, multi-paradigm software architecture for the control of autonomous vehicles. The top, or Strategic, level consists of general mission directives and the bottom, or Execution, level consists of specific vehicle

1

commands [Byrn93]. Both have been specified and implemented in some detail. The middle, or Tactical, level, is responsible for breaking down the broad guidance of the Strategic level into simple pieces of behavior that the Execution level can carry out. This level is thus the crucial bridge that connects the other two distinct parts of the model, but it has been implemented in only a very limited way.

The design of the Tactical level is well-suited to the object-oriented paradigm and has been described in [Byrn93]. The behaviors of the Tactical level can be grouped together quite easily under objects in an object hierarchy. Implementing the relationships of this hierarchy requires an *object-oriented* or *object-based* language[1]. The complex, time-consuming nature of certain tasks such as search, homing, and mission replanning make concurrent programming facilities extremely desirable as well so that control of the vehicle can be maintained continuously throughout a mission, insuring the vehicle's rational behavior. Therefore, the problem is to find a programming language to represent the concurrency and the object-oriented nature of the Tactical level well and to build an implementation model.

C. SCOPE

The primary goal of this research is to develop a working model of the Tactical level of RBM in a currently available programming language using object-oriented techniques and programming language constructs for concurrency. For this research, concurrency is limited to multitasking, or the interleaving of multiple processes on a single processor. Distribution is beyond the scope of this work. This thesis focuses on a few areas of research, including representing concurrency in software, implementing object-oriented design, and the suitability of current programming languages for these two tasks.

---

1. *Object-based* languages have features to support the principles of data abstraction and information hiding, while *object-oriented* languages have mechanisms for inheritance, dynamic binding, and polymorphism in addition to those features. However, as Booch notes, " ... it is possible and highly desirable for us to use object-oriented design methods for both object-based and object-oriented programming languages." [Booc91, p. 36]

D. THESIS ORGANIZATION

Chapter II surveys previous work on software systems that have implemented object-oriented design and concurrency. Chapter III gives an overview of RBM. Chapter IV discusses the programming languages considered for implementing the Tactical level. In Chapter V, the Tactical level implementation is explained in detail. Chapter VI examines testing of the implementation in the laboratory on the AUV simulator. Chapter VII provides a summary of conclusions and suggestions for future research. Appendix A lists the source code for the Tactical level. Appendix B gives a trace of the execution of two multi-phase mission scenarios. Appendix C is a user's guide to the AUV simulator used in this research.

II. PREVIOUS WORK

A. INTRODUCTION

There have been numerous efforts to implement concurrency using multi-tasking in real-time software applications. Three projects with varying timing requirements are described here. All three projects have employed some form of the Ada programming language and have either attempted to use or intend to use Ada's task construct for concurrency.

B. NASA OMV

NASA's Orbital Maneuvering Vehicle (OMV) is a semi-autonomous spacecraft designed to provide services to other spacecraft, including delivery, retrieval, reboosting, and deboosting. The craft has automatic navigation and rendezvous capabilities but requires human control for terminal operations such as docking with NASA's Space Station. Control for the OMV can be provided from the space shuttle, from the ground, or from the Space Station. The OMV can carry various mission kits and has a nine month on-orbit capability.

Standard Ada was used for prototyping on the software system. Tasking was rejected for this system, however, due to the system's strict real-time requirements. In particular, the need to change the priority of a task at run time and the need to specify a task as non-preemptible by other tasks to meet certain time constraints were seen as necessary features not provided by the Ada Run Time System (RTS). Prototype tasking algorithms were much slower and larger than the established sequential ones. As a result, Ada tasking was not used further in the project [Howl88].

C. NASA EXPLORER MMS

NASA's Explorer Multimission Modular Spacecraft (MMS) is an unmanned orbiting space vehicle with a replaceable payload. The payload is a science instrument replaced by

the space shuttle every 18 to 24 months. Control of Explorer, such as attitude commands are generated by the ground, the onboard processor, or the onboard coprocessor.

Standard Ada was used in a benchmark test with the intent of seeing how it would handle some of the spacecraft's software functions, including attitude determination support, coprocessor system monitoring, and coprocessor self-checks. Developers considered tasking viable for this system with some changes in the task scheduler to reduce overhead time. Published task rendezvous time of 800 microseconds was not critical for this implementation. What was important was that task priorities could be set and synchronous and asynchronous interrupts handled due to minimal human control (Communication with the ground is limited to about 15 minutes every 1 1/2 hours). Planned modifications to the Ada RTS were designed to identify the cause of an interrupt and the portion of code involved in a telemetry report to the ground [Scot88].

D. NAVAL POSTGRADUATE SCHOOL AUV

1. Vehicle Description

The Naval Postgraduate School Autonomous Vehicle (AUV) is an unmanned, untethered, robotic submarine. Its purpose is to provide multi-area research for students and faculty and its projected missions include search, surveillance, mapping and intervention activities. The current model of the vehicle, shown in Figure 1, is 7 feet long, weighs approximately 400 pounds, and has a maximum speed of 2 knots. Due to its relatively small size and low cost, the vehicle is an ideal research platform. Power for control surfaces and cross-body thrusters is provided by a battery-based system which can last 2 to 3 hours on a charge. The vehicle is controlled by two separate processors on Gespac platforms: one for vehicle actuator control and one for mission control and navigation. Sonar, inertial navigation, and global positioning systems are also incorporated onboard [Heal92].

Software control is provided by RBM, which is described in Chapter III. The high-level navigation and system-monitoring functions comprise the Tactical level. Byrnes in [Byrn93] developed a Tactical level instantiation using Classic-Ada, a preprocessor for
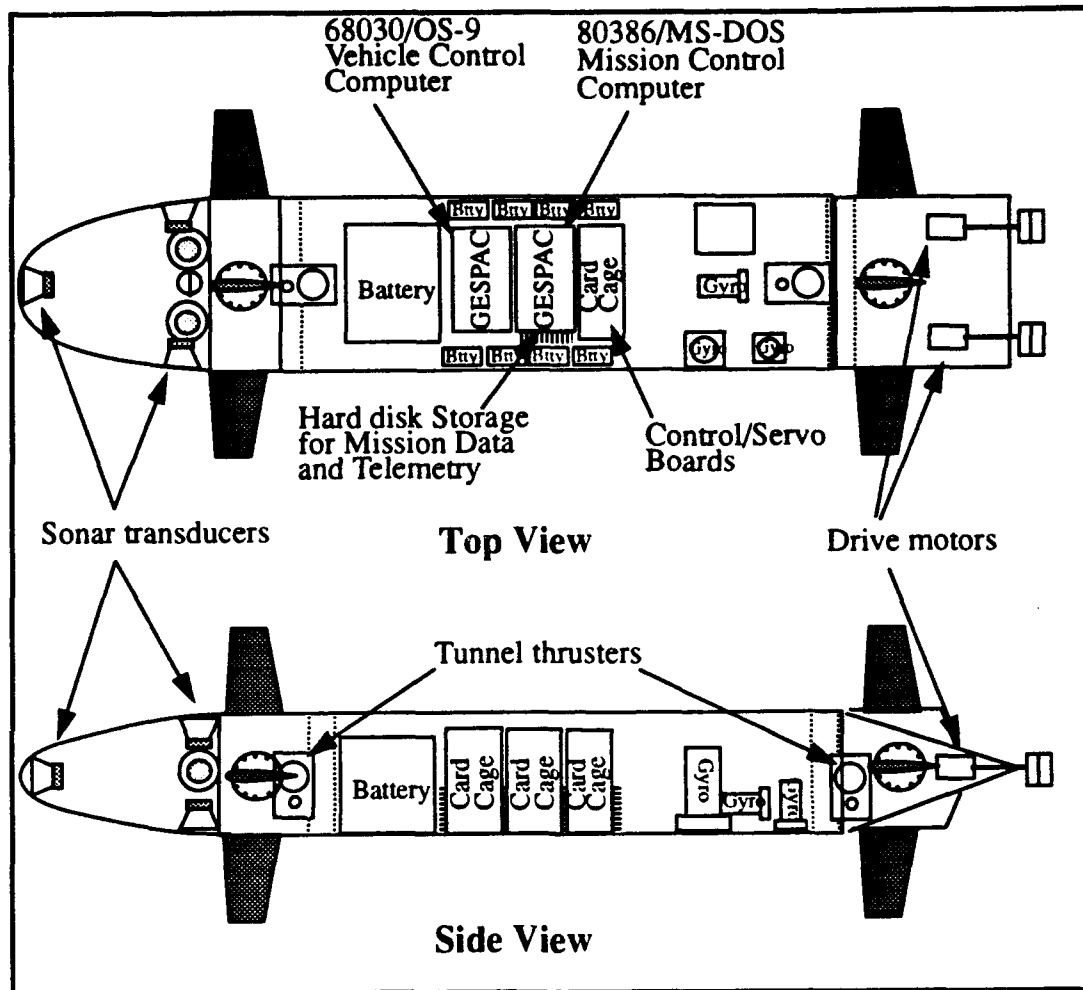
Figure 1 The Naval Postgraduate School AUV II

the Ada language which produces object-oriented extensions such as inheritance and dynamic binding.

2. Simulation Environment

Simulation testing is performed on the software in the laboratory before the software is placed in the actual vehicle. Testing of the model in the laboratory was accomplished by linking three separate processors through an Ethernet connection using stream socket communications. The Strategic level was programmed in Prolog and CLIPS and ran on a Sun SPARCstation 4/280 using the UNIX operating system. The Tactical level was written in Classic-Ada and was also hosted on a Sun SPARCstation 4/280 running

UNIX. The Execution level and the simulator itself were programmed in C and ran on a Silicon Graphics 4D/340VGX workstation using the IRIX operating system. The three-processor test configuration is shown in Figure 2.
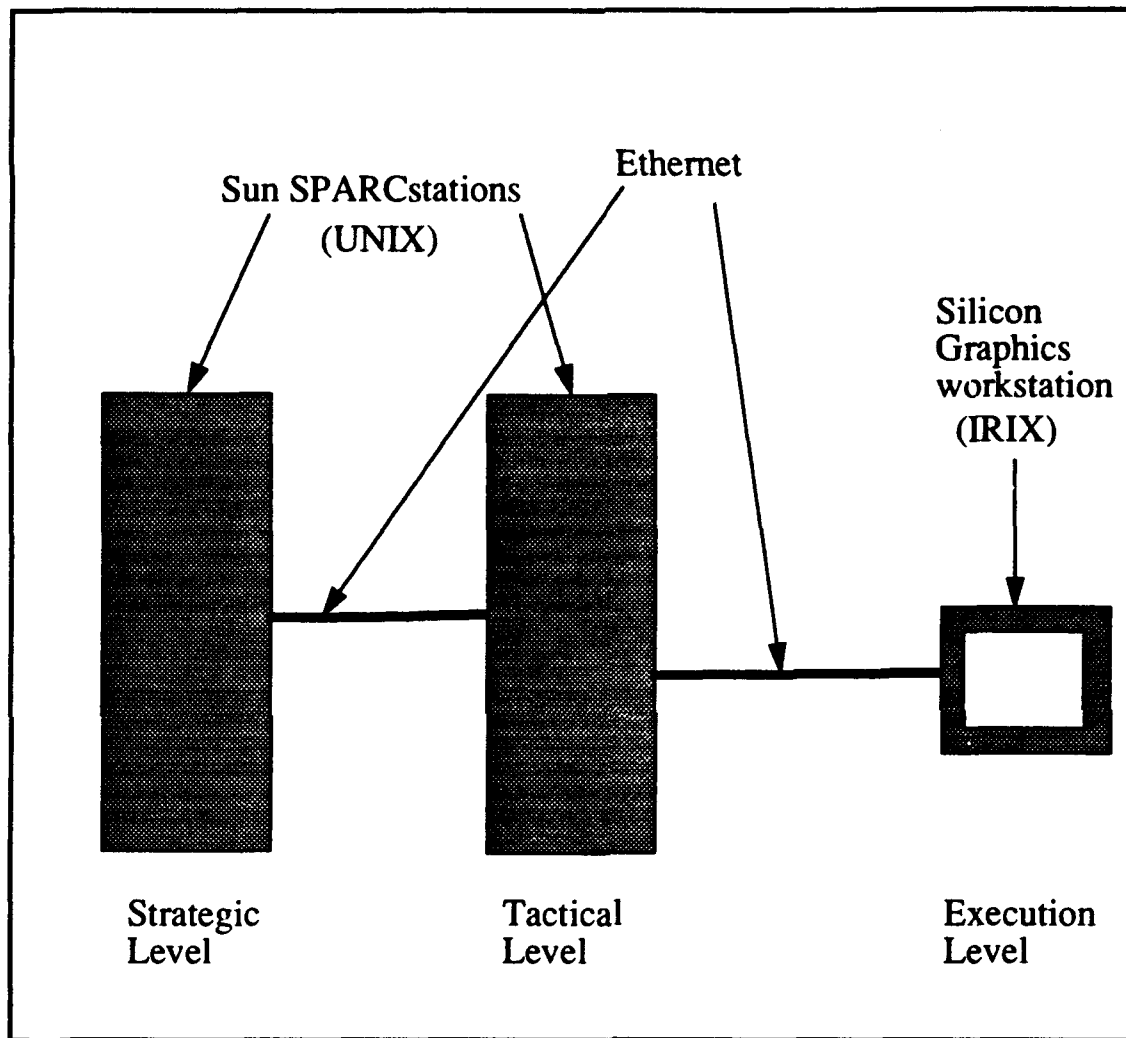
Figure 2 Original AUV Simulator Test Configuration

This Classic-Ada implementation of the Tactical level is truly object-oriented in the sense that it allows inheritance of object characteristics and provides dynamic binding of operations to objects. However, this version employs a sequential approach to carry out required behaviors which presents some problems for multiple modes of operations. This thesis research is an extension of that work in an attempt to add Ada tasking for concurrent

operations on the Mission Control Computer to fulfill the intent of RBM. The new Tactical level implementation relies on the Ada RTS without modification for task scheduling and is discussed in Chapter V.

III. THE RATIONAL BEHAVIOR MODEL

A. INTRODUCTION

The Rational Behavior Model (RBM) is an autonomous vehicle control software architecture composed of three distinct levels. The levels of RBM are based on the degree of abstraction of the problem domain, and they are, from highest to lowest: the Strategic, Tactical, and Execution levels [Kwak92]. The structure of RBM is illustrated in Figure 3.
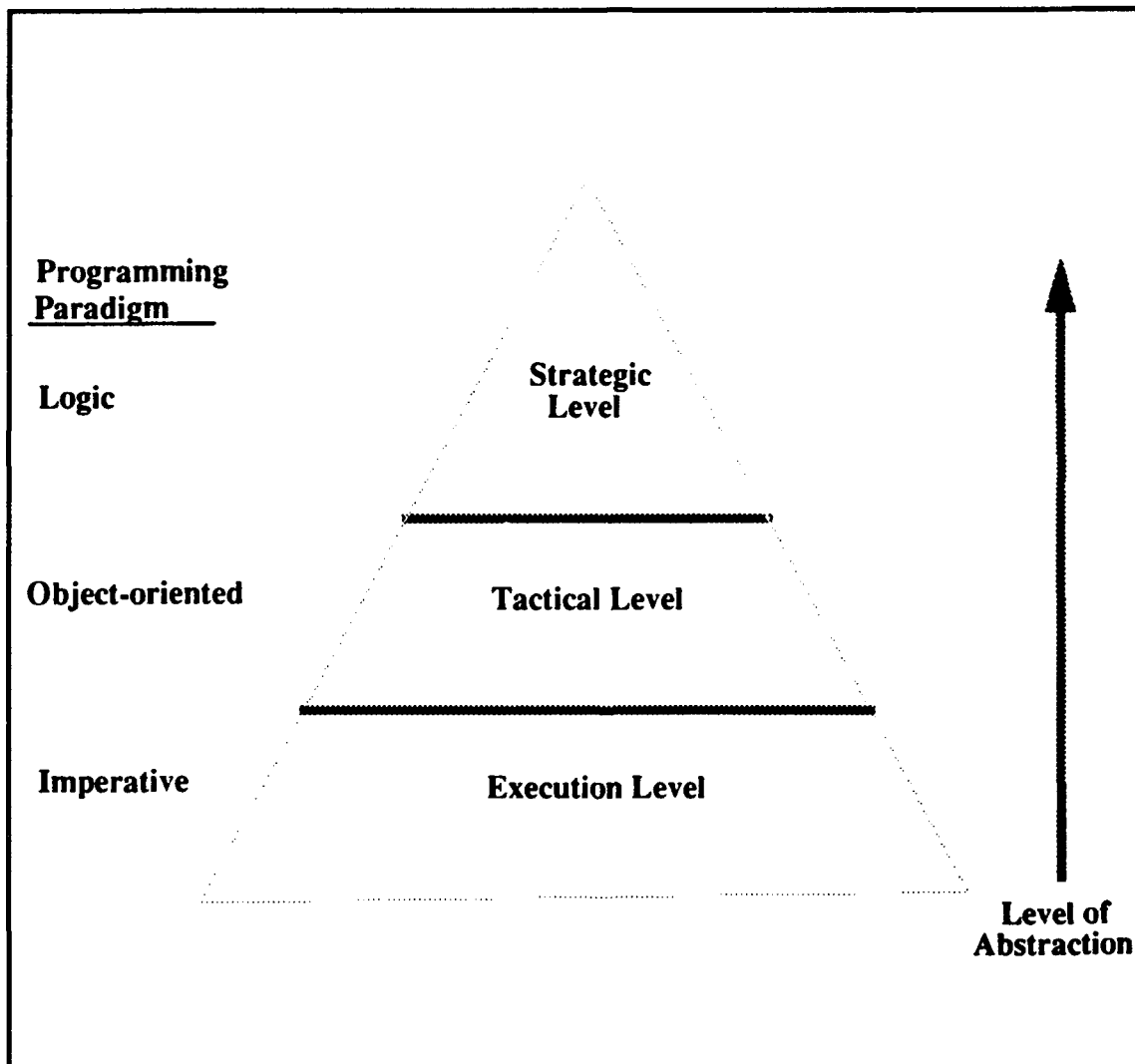
Figure 3 RBM Structure

The power of RBM for software engineering lies in its tailoring available design resources to address the important aspects of the problem at hand. When the programming paradigm matches the abstraction of the problem instead of being forced into it, the result is robust and easily understood software. Such software can be modified with little difficulty, satisfying one of the key objectives of software engineering.

B. STRATEGIC LEVEL

The Strategic level stands at the top of the RBM hierarchy. At this level, the essence of a mission is expressed using clear, high-level logic so that the vehicle can act in a rational manner. Logic for sequencing behaviors is encapsulated at this top level. Simplicity is maintained by the Strategic level having no internal memory and no knowledge of operational details. Required mission behaviors are provided by the process of goal-driven decomposition. A root or mission goal is repeatedly refined into its constituent subgoals until primitive goals are reached. Implementation is initiated at this point. Because the reasoning process proceeds according to a deliberate sequence, the Strategic level can be expressed quite naturally in a rule-based programming language like Prolog or CLIPS. The rule set of the Strategic level is divided into *mission specification* and *doctrine*. The mission specification part deals with knowledge unique to a mission, while the doctrine part concerns mission-independent knowledge that is usually tied to the nature of the vehicle.

Once a primitive goal is identified, the Strategic level calls on the Tactical level to start some type of appropriate behavior. These calls can be either *queries* or *commands*. Queries are information requests which require a binary response. Commands are orders requiring no feedback other than an acknowledgment of completion of the ordered task. If more information is needed to make a decision after a command has been issued, queries are used to poll the Tactical level [Byrn93].

C. EXECUTION LEVEL

The Execution level lies at the other end of the RBM hierarchy. It is responsible for the multitude of complex physical actions that comprise the primitive goals of the Strategic

level; therefore, it must guarantee basic vehicle stability. Stability is provided by a series of autopilots driven by servo loops. In addition, processes with hard real-time scheduling constraints are encapsulated at the Execution level. While computation at the Strategic level is purely symbolic, computation at the Execution level is completely numeric to ensure timing requirements are met. Implementation of this level requires an imperative programming language with good numeric computation speed such as C or Fortran.

Since it is the base of the RBM hierarchy, the Execution level must act as the intermediary between the software and the hardware. This level receives setpoints and vehicle mode information from the Tactical level, and its autopilots must use these data repeatedly until they are updated. Autopilot commands are sent to motors, control surfaces, and other hardware devices using digital and analog signals. Information is received from analog hardware devices in the form of digital readings. Changes in hardware are mostly contained within the Execution level unless new tasks or new hardware capabilities are added. In this case, the Tactical level must be modified as well [Byrn93].

D. TACTICAL LEVEL

The Tactical level is the middle level in the tri-level RBM hierarchy and is the focus of this research. This level is the crucial link between the knowledge-based orientation of the Strategic level and the numeric-based orientation of the Execution level. Therefore, the primary objective of the Tactical level is to act as a bridge between the two end levels and cannot be discussed without reference to these two levels. This level responds to queries and commands from the Strategic level and inputs from the Execution level through specific behaviors.

In its role as coordinator between the Strategic level and the Execution level, the Tactical level must be an analyst and translator. Abstract behaviors from the Strategic level must be analyzed and then translated into their executable details to be performed by the Execution level. The Tactical level takes the general descriptions of what the vehicle is supposed to do and supplements these with timing details and physical constraints of the

11

vehicle as it decomposes them into simpler and simpler behaviors. The resulting primitive behaviors, which consist of data requests and setpoint and control mode commands, are sent to the Execution level to be carried out [Kwak93].

Tactical level behaviors can be grouped under the entities which perform them. These entities have state, behavior, and identity and are called *software objects* [Booc91]. Objects, in turn, are organized into a hierarchy such that each parent object decomposes into one or more dependent, or child, objects. The object at the top of the hierarchy acts as the interface between the detail-free Strategic level and the rest of the hierarchy. An object at the Tactical level only has knowledge of its parent and its children and nothing else. To access any other object, including its own siblings, an object must go through the parent of that other object. The only exception to this rule is that data required by multiple objects can be retrieved directly from specifically designated database manager objects [Byrn93]. Modifications and additions to the object hierarchy are facilitated by this structure. In addition, parallel threads of control can be identified among objects under different parents for concurrent execution [Kwak93].

E. TACTICAL LEVEL REQUIREMENTS

Just as the quality of a bridge depends on its keystone, the strength of the Tactical level as an interface between the Strategic and Execution levels in RBM depends on its design specification. An appropriate structure for the design specification of the Tactical level is a basic requirement for implementation. The design pattern used for this research was the watch crew of a submarine, which provides a representative, well-understood model for Tactical level relationships [Byrn93].

The design specification is not very useful unless it is supported by appropriate programming facilities. A programming language is the raw material out of which the Tactical level bridge is built. Its utility as a bridge depends on the appropriateness and power of the language chosen for implementation. The least that is required to represent the relationships of this level is an object-based language, although an object-oriented

language is preferred to accommodate future modification and growth. Some method for implementing concurrency is also necessary. Choosing a programming language is discussed in the next chapter.

IV. TACTICAL LEVEL PROGRAMMING LANGUAGES

A. BACKGROUND

There are numerous programming languages that are object-oriented or object-based. This number is reduced substantially when the criterion of constructs to support concurrency is considered. Many powerful object-oriented languages such as C++ and CLOS do not presently provide explicit support for concurrency. The remaining subset of languages is limited to Ada and its variants. The applicability of these languages to the Tactical level problem domain is now examined.

B. ADA

Ada is an object-based language developed for the United States Department of Defense to handle very large, software-intensive systems. Ada has numerous features which support object-oriented design, including packages, tasks, and generic units [Booc91]. Since Ada has objects but does not have explicit classes, however, it has no mechanism for inheritance, dynamic binding, or polymorphism in its present form. Therefore, message passing between objects is detailed, complicating design in a large software system incorporating many related classes of objects. This does not pose a problem for the Tactical level as it is currently designed for the AUV, because an object hierarchy is sufficient to specify relationships. Future growth and redesign would be better accommodated by a class-based language.

Concurrency is supported in Ada through its task construct. Tasks are based on the Communicating Sequential Processes (CSP) model [Hoar78] in which processes synchronize and then pass messages through *input* and *output* statements. This synchronization is called a *rendezvous* and is required between two processes before communication can occur. If one task reaches the rendezvous point before the other, it must wait or accept another task that is ready to pass a message. Exclusive access to data or a resource is thus built in with the CSP model, since a task can only communicate with one

14

other task at any given time. Ada's *accept* statements and entry calls function in the same way as CSP's *input* and *output* statements, respectively, with some added features. First, communication in Ada tasks is bidirectional, while it is strictly unidirectional in CSP tasks. Second, to CSP's parameter copying, the Ada rendezvous adds the capability for the called task to execute statements and return results to the calling task [Geha84]. Although tasks cannot stand alone, they can be encapsulated as objects, providing a powerful abstraction mechanism for object-based applications that are concurrent in nature. Task objects are an excellent representation for the objects of the Tactical level which must perform multiple functions.

C. CLASSIC-ADA

Classic-Ada is a preprocessor for Ada which adds capabilities needed to complete the object-oriented paradigm. Processing Classic-Ada code yields pure Ada source code with special data structures to support inheritance, dynamic binding, and polymorphism. Data and behaviors for an object are written as *instance variables* and *instance methods*, respectively. These characteristics are unique to that object and its class. An object communicates with another object simply by using a *send* statement with the object name and the *instance method* name [Soft92]. This extension to Ada provides a much more concise method for message passing between objects. Messages can be passed without any bulky or artificial syntax as in Ada. Also, a class structure can be built which facilitates modifications to the Tactical level because of the built-in inheritance mechanism.

Concurrency is supported in Classic-Ada through the Ada task construct. However, there is no provision for implementing tasks at the object level. Tasks can only be declared within methods, severing the link between objects and tasks that is available in Ada. This restriction severely limits the usefulness of Classic-Ada for implementing object-oriented designs that involve a significant amount of concurrency, such as the Tactical level.

D. ADA 9X

Ada 9X is a revised version of Ada which updates the 1983 ANSI Ada standard. Although it is not yet commercially available, Ada 9X deserves examination. It will soon become the standard for Ada, and it incorporates some object-oriented capabilities. Ada 9X provides for inheritance, dynamic binding, and polymorphism through its *tagged* type construct, which allows components to be added to a type when it is derived. Public and private *record* types are the only types that can be *tagged*.

Ada 9X also enhances the basic task construct for concurrent programming. More flexibility is provided in choosing priority and scheduling rules, task *delay* times can be made explicit, and asynchronous transfer of control is provided by additions to the task *select* statement [DoD93]. Nevertheless, the object-oriented paradigm is not extended to task types; task types cannot be *tagged* and thus are static in nature[1]. Since its task type is unchanged from Ada, Ada 9X offers no significant advantage for representing the concurrency of the Tactical level.

E. COMPARISON OF PROGRAMMING LANGUAGES

Ada, Classic-Ada, and Ada 9X all have advantages and disadvantages for the Tactical level application. Ada supports concurrency well with its *rendezvous*, providing a high-level model of communication to enforce mutual exclusion. Classic-Ada extends Ada but superimposes object-oriented features at a higher level rather than integrating them with Ada [Atki91]. The lack of object-level tasking is a serious drawback. Ada 9X offers promise for integrating object-oriented features with Ada in many areas but not in the area of concurrency. What is needed is a language that combines object-oriented and concurrent concepts, considering classes, objects, and tasks together. Figure 4 illustrates the current programming language situation. In the absence of such a language, Ada was chosen for its availability and the flexibility of its task construct.

---

1. In Ada 9X, as in Ada, the number of tasks of a task type can be dynamic.
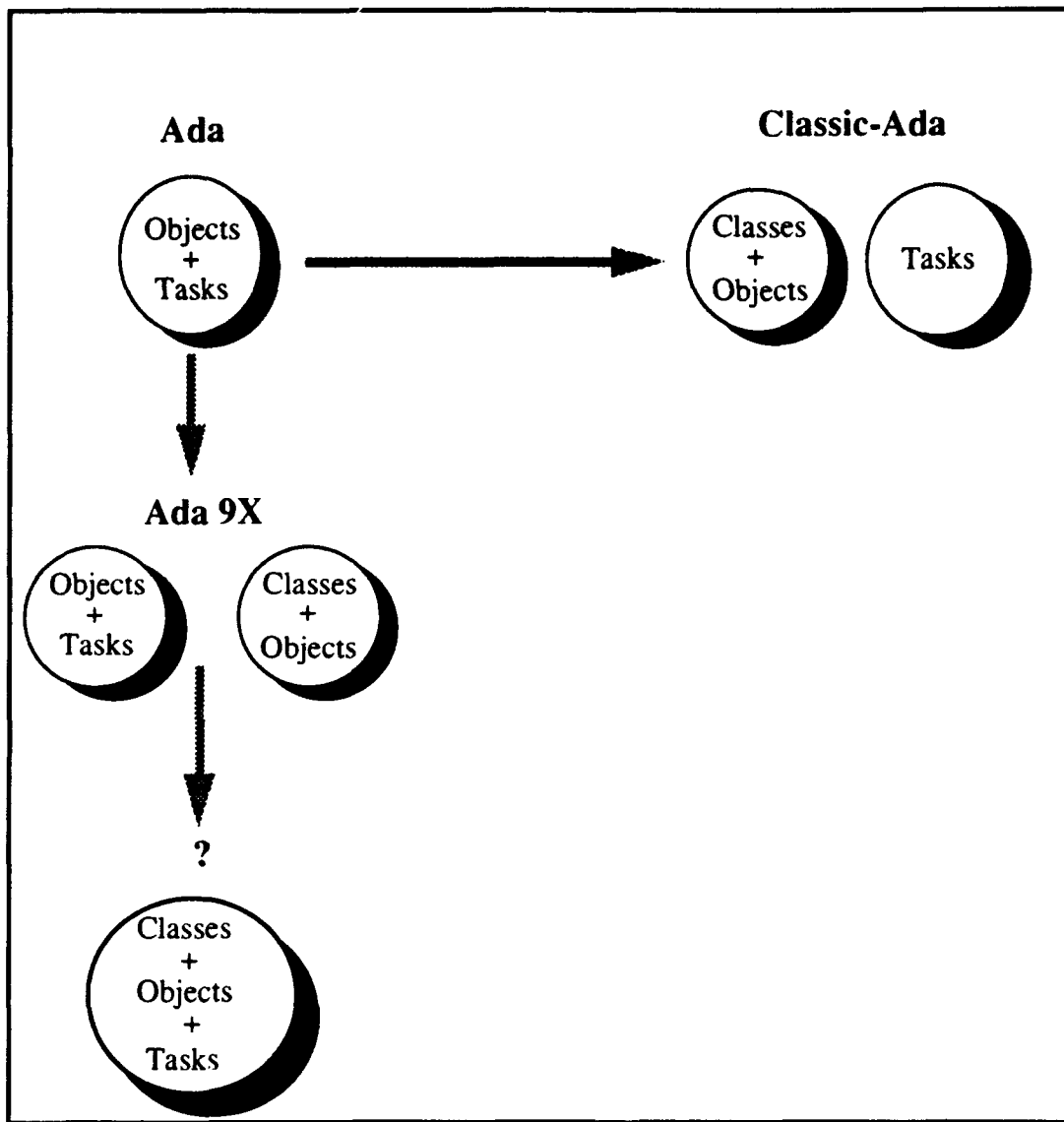
Figure 4 Tactical Level Programming Languages.

V. TACTICAL LEVEL IMPLEMENTATION

A. OVERVIEW

The quality of the Tactical level implementation depends significantly on the quality of its design. As mentioned in Chapter III, the watch crew of a manned submarine offers a natural model for representing the entities and behaviors of the Tactical level. Using this model, an object hierarchy can be built which supports an implementation model. The implementation model is the method of construction of the Tactical level bridge; it determines how the raw material of the programming language gets put together on the keystone of the design model.

B. DESIGN MODEL

The design specification for the Tactical level is given in Figure 5. The blocks in the diagram stand for distinct entities within the Tactical level structure, and each one corresponds to a software object. The hierarchical structure of the Tactical level encompasses most of the objects and is indicated by the dotted lines between them. The AUV Officer of the Deck (OOD) provides overall operational control at this level and stands at the top of the hierarchy. The OOD also provides the sole interface between the Strategic and Tactical levels. Top level primitive goals are handed to the OOD so that he can activate the behaviors understood by the Tactical level to satisfy those goals. In the watch crew, the Captain gives commands or requests the status of the submarine's systems from the OOD. The OOD, in turn, in gives the required orders to satisfy the goal or answer the query issued by the Captain.

The Tactical level objects cover all the behaviors that the vehicle can perform. Coordinating the operations of each object, the OOD insures each task is completed appropriately. Behaviors are implemented as methods within an object. For the most part, behaviors require the involvement of multiple objects. Communication between objects is accomplished through message passing. As mentioned. communication is limited to
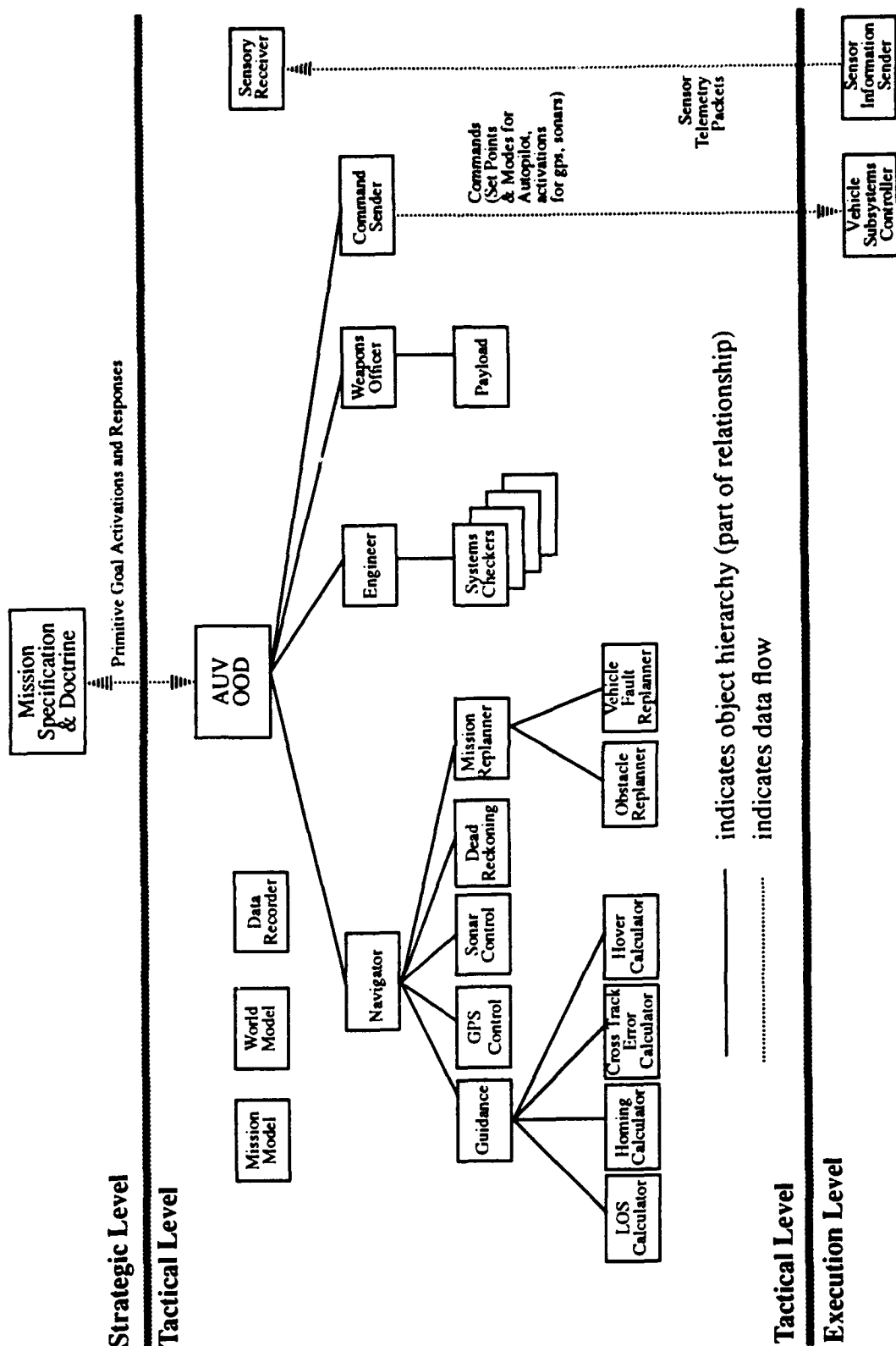
Figure 5 Tactical Level Design Model

parent-child pairs. In this scheme, efficiency is sacrificed to gain modularity of code and ease of understanding for the user.

Just as all Strategic level communications must go through the conduit of the OOD, all contact with the Execution level is similarly constrained. Command packets comprised of setpoints and modes are transferred solely through the Command Sender object under the direction of the OOD. In addition, telemetry data is accepted from the Execution level by the Sensory Receiver object exclusively. The limitations on these interfaces eliminate command and data discrepancies.

There are a number of objects that are disconnected from the object hierarchy in the Tactical level. These correspond to databases that serve any other requesting object any time their respective data are needed. They contain the state of the mission (Mission Model), the perceived state of the environment (World Model), recorded mission history (Data Recorder), and current sensor readings (Sensory Receiver) [Byrn93].

C. IMPLEMENTATION MODEL

The implementation model gives life to the relationships expressed in the design model. The structure of the implementation model using Ada is illustrated in Figure 6. The methodology for this design was to provide concurrency between objects while adhering to the control requirements of RBM. Getting the AUV to execute a mission involving multiple modes of operation and showing that it can replan a mission in progress without giving up control were the goals of the implementation. The code for the implementation in Ada is found in Appendix A.

1. Description of Communication

Commands and queries are passed between Tactical level objects by means of task *entry* calls with boolean flags. Each command issued to the OOD has a *goal flag* which gets set to true when execution of the command is complete. A command is attempted until the *goal flag* is set to true to insure that it gets executed. Each query has a *return flag* and a *goal flag*. The *return flag* gets set to true when the appropriate object has received the
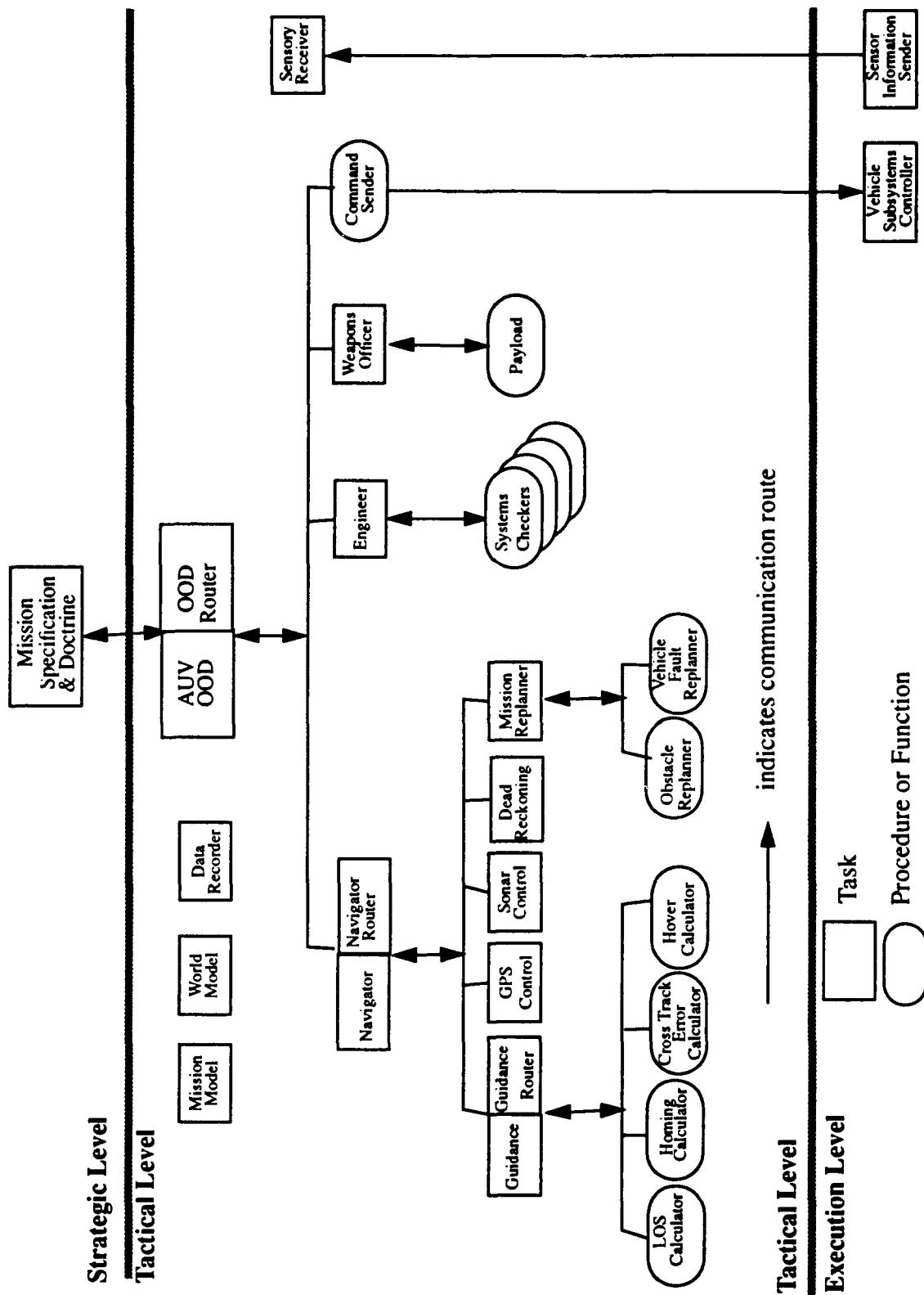
Figure 6 Tactical Level Implementation Model

21

query. In this case, the *goal flag* gets set based on a positive or negative response to the query. A query is attempted until the *return flag* is set to true to insure that the query has been communicated to the target object.

All upper level objects in the hierarchy are represented as tasks in Ada. Each of these tasks consists of a set of *accept* statements, which are messages for behaviors that the respective object or its children perform. Each *accept* statement further contains entry calls to child objects, and this chain of message passing continues until an object is reached that can execute part or all of a given command or answer a given query. An example of the message passing pattern is shown in Figure 7.

```
task A is
        .
        .

    accept QUERY_A(GOAL_FLAG, RETURN_FLAG : out BOOLEAN) do
      if QUERY_A =TRUE then
        GOAL_FLAG := TRUE;
      else
        GOAL_FLAG := FALSE;
      end if;
      RETURN_FLAG := TRUE;
    end QUERY_A;

        .
        .

    accept COMMAND_A(GOAL_FLAG : out BOOLEAN) do
      task A_1.COMMAND_A(GOAL_FLAG_1);
      if GOAL_FLAG_1 = TRUE then
        GOAL_FLAG := TRUE;
      else
        GOAL_FLAG := FALSE;
      end if;
    end COMMAND_A;
  end task A


    task A_1 is

    accept COMMAND_A(GOAL_FLAG_1 : out BOOLEAN) do
      do COMMAND_A;
      GOAL_FLAG_1 := TRUE;
    end COMMAND_A;
    end task A_1;
```

Figure 7 Example of Task Communication

22

The lowest level objects are represented as procedures or functions, since these objects consist of only basic operations. As leaves on the object hierarchy tree, these objects require no further communication with any objects so implementing them as tasks would introduce unnecessary overhead. However, these objects must still be able to communicate with their parent objects while performing their respective functions to support RBM's control scheme. Since the parent object task is suspended while it waits for the child to complete its required behavior, some alternate way must be used to pass messages to the parent during this time.

The method of alternate communication used in this research was a series of router, or *relay*[1], tasks. A relay task waits until it is called by a task with data to send and then immediately calls the next task in the series. This process continues until the data is consumed. Use of these intermediary tasks allows for a loosely coupled implementation, but this advantage must be balanced against the overhead of added tasks [Lema89] [Niel88]. Relay tasks allow time-consuming behaviors such as search and homing to continue while the primary route of communication is suspended awaiting an answer to send back to the Strategic level. The situation is illustrated in Figure 8 using homing as an example.

The database objects are also all implemented as tasks to insure only one object at a time can access any one of them. Otherwise, Sonar Control, for example, could set the vehicle's mission mode in the Mission Model while the OOD is attempting to read that value. The Ada *rendezvous* enforces mutual exclusion, preventing such data inconsistencies. Only the first entry call is allowed to participate in the *rendezvous*. All others are queued and serviced sequentially.

---

1. *Relay* tasks are one of three types of intermediary tasks. *Buffer* tasks, which have an entry to accept data from a producer and an entry to send data to a consumer when requested, and *transporter* tasks, which request data using an entry call to a producer task and then provide the data to a consumer through an entry call, are the other types of intermediary tasks.
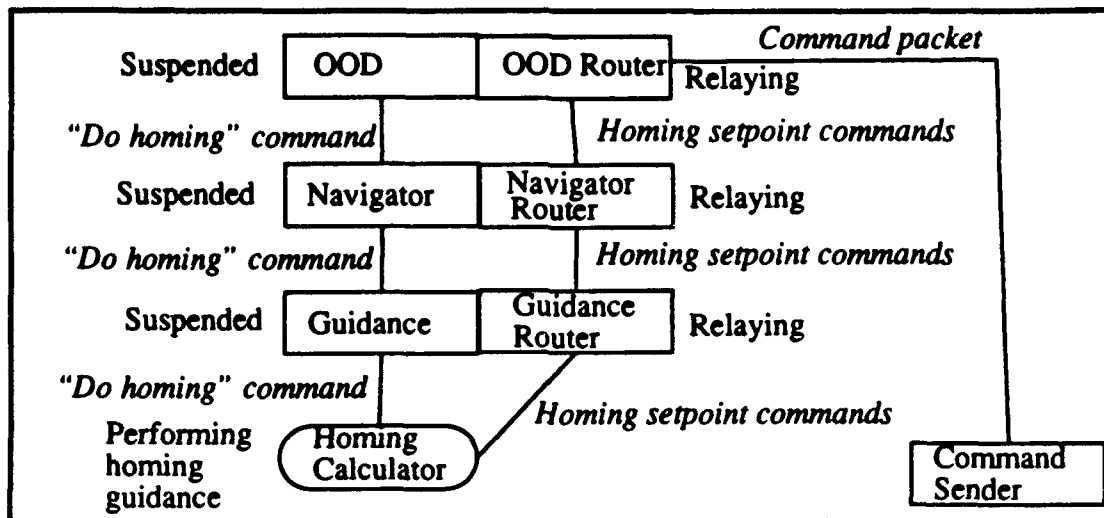
Figure 8 Router Task Communication

2. Description of Objects

a. OOD

This object consists of two tasks, one for the main OOD functions and one for routing. As the top level of the object hierarchy, the main OOD task must contain *accept* statements for all of the primitive goals issued by the Strategic level. *Entry* calls within each *accept* statement activate the behaviors necessary to satisfy a particular goal. The main OOD task must also coordinate these behaviors. The OOD relay task acts as a backup channel to the Command Sender when the main OOD task is suspended waiting for a command to be executed.

b. Navigator

This object also contains a main task and a routing task. The main Navigator task is responsible for guidance, position estimation, and path replanning. This task's view of the world at any given time extends only from its present position to the next waypoint to make its operation as generic as possible. All mission details are encapsulated in the Mission Model. Following the OOD's model, the main Navigator task passes on orders to its subordinates using entry calls and coordinates their actions. In the case of mission replanning, this coordination involves concurrency, as guidance for loitering must be

provided at the same time as the mission route is being replanned. The Navigator relay task acts as a backup channel to the OOD when the main Navigator task is suspended waiting for a command to be executed.

c. *Guidance*

This object is comprised of a main task and a routing task as well. The responsibility of the main Guidance task is to provide the heading and depth setpoints to be included in the command packet sent to the Execution level. The *accept* statements in this task contain calls to procedures that do various types of guidance.

For this study, line-of-sight (LOS) guidance and homing guidance were both implemented. The new command heading to a waypoint is computed for LOS guidance as follows:

$$\Psi_{cmd} = \text{atan}\left[\frac{(Y_{next} - Y_{curr})}{(X_{next} - X_{curr})}\right] \qquad \text{(Eq 1)}$$

where:

X_{curr}, Y_{curr} = X, Y components of AUV's current position.

X_{next}, Y_{next} = X, Y components of next waypoint.

The new command heading to a target is computed for homing guidance using the following equation:

$$\Psi_{cmd} = \Psi_{curr} + \beta \qquad \text{(Eq 2)}$$

where:

Ψ_{curr} = Current vehicle heading.

β = Sonar relative bearing to target.

The Guidance relay task acts as a backup channel to the Navigator when the main Guidance task is suspended waiting for a command to be executed.

25

d. GPS Control

This object is responsible for controlling the Global Positioning System receiver and accessing it for navigation. This capability was not modeled for this research. The GPS Control task in this implementation simply returns a positive response when a GPS fix is requested. Research on integrating GPS in this environment is included in [Stev93].

e. Sonar Control

This object issues sonar commands, checks for and logs objects, and monitors the sonar for various tasks such as search. In this study, this object consists of a single task which monitors the sonar range and bearing values while the vehicle executes the command "do search pattern". The task executes an expanding box search algorithm until threshold values for both range and bearing are detected from the sonar. The search pattern and algorithm are shown in Figure 9.

f. Dead Reckoning

This object provides present position based on a known position fix, actual heading, and elapsed time. The Tactical level dead reckoner serves as a backup to the Execution level dead reckoner to crosscheck its operation. The dead reckoner was not implemented for this study.

g. Mission Replanner

This object has a single task to perform local replanning for avoiding obstacles and global replanning to accommodate a vehicle fault. Global replanning was modeled by using a *delay* statement and instantaneously changing the mission route through the Mission Model.

h. Engineer

This object consists of one task to monitor the condition of each vehicle system. For this study, a thruster system problem was modeled by reducing the thrust level

Bi_k indicates the position in the search pattern where:

i = Box number

k = Leg number

```
Algorithm DO_SEARCH_PATTERN
begin
  NEXT_TIME := CLOCK + INTERVAL - TURN_TIME;
  LEG_NUM := 0;
  Initialize SEARCH_HEADING
  loop
      if CLOCK > NEXT_TIME then --Change heading for new leg
        if LEG_NUM = 2 then --Expand the box
          LEG_TIME := LEG_TIME + INTERVAL;
          LEG_NUM := 1;
        end if;
        --Change heading to make box corner and normalize
        if SEARCH_HEADING > (PI / 2) then --Command heading > 0
          SEARCH_HEADING := SEARCH_HEADING - (PI / 2);
        else --Command heading <= 0
          SEARCH_HEADING := SEARCH_HEADING + (3 PI / 2);
        end if;
        LEG_NUM := LEG_NUM + 1;
        NEXT_TIME := NEXT_TIME + LEG_TIME;
      end if;
      Receive SONAR_BEARING and SONAR_RANGE
      Send SEARCH_HEADING and SEARCH_MODE
      exit when SONAR_RANGE < RNG_LIMIT and ABS(SONAR_BEARING) < BRG_LIMIT;
  end loop;
end DO_SEARCH_PATTERN;
```

Figure 9 Expanding Box Search Pattern and Algorithm

gradually from an initial value until it moved below a given threshold. *Accept* statements for all other system checks give a negative response to indicate the systems are operating properly.

i. *Weapons Officer*

The Weapons Officer is comprised of one task that is responsible for monitoring and delivering the vehicle's payload. This capability was not implemented for this research. The command to employ weapons simply returns a positive response.

j. Command Sender

This object accepts command packets built by the OOD and sends them to the Execution level. A command packet consists of command X and Y coordinates, command heading, command depth, command speed, and mode. Since this object just relays data and cannot be accessed by any object other than the OOD, it was implemented as a procedure. The physical separation of the Tactical and Execution levels in this study required additional procedures for network communications.

k. Sensory Receiver

This object consists of a single task that accepts telemetry records from the Execution level, stores the individual values, and provides the data to other Tactical level objects when requested. Each sensory packet contains vehicle position represented as X and Y coordinates, altitude above the bottom, and depth. This object is also responsible for putting a time stamp on a sensory packet before sending it to the Data Recorder, although this feature was not implemented in this work.

l. Mission Model

This object is comprised of one task to hold and manage the waypoints that make up the mission route and the vehicle modes for the various phases of the mission. For the purposes of this thesis, these values were entered in data files which were read in by the Mission Model upon initialization of the simulator.

m. World Model

This object has one task to hold and manage known objects and other environmental data. Obstacles were the only type of environmental data used in this study. These data were entered in files and read in during initialization as the Mission Model data was.

n. Data Recorder

This object consists of a single task to accept and maintain telemetry records and other explanatory messages for post-mission analysis. This object was not modeled for this research.

3. Mission Environment

A mission in reality involves multiple phases and the possibility of unforeseen system problems. Such an environment requires the AUV to operate in more than one mode and the OOD to coordinate the behaviors of Tactical level objects concurrently as well as sequentially.

The target mission for this research was a search-and-rescue mission developed by the 1992 National Science Foundation workshop on furthering and evaluating autonomy in the area of underwater vehicle technology [Stee92]. In this mission, the AUV must traverse a given search area, locate a subsurface buoy, cut the buoy's mooring line, drop a package as close to the buoy as possible, return to the launch site, and surface. The interpreted rule set for this mission written in Prolog is presented in [Byrn93]. The mission is broken down into the following four phases: *transit*, *search*, *task*, and *return*.

The vehicle has four modes that correspond directly to the four mission phases. *Transit* and *return* are basically the same at the Tactical level. Navigation is executed using LOS guidance after the Navigator receives each query about whether a waypoint is reached. The only concurrency implemented in these modes is this execution of LOS guidance as the Tactical level releases control back to the Strategic level for the next command to be issued, and this is minimal.

Initiation of the *search* mode creates problems for a sequential implementation. The Strategic level must know the search is completed before issuing the next command, and so it waits on the OOD. The OOD waits on the Navigator, which waits on Sonar Control. While all these tasks are suspended, control of the vehicle must be maintained for the search through the objects that are waiting for the search to complete. Therefore, a

series of relay tasks is required in Ada to provide *intra-object* concurrency. The situation is the same in the *task* mode while homing is being performed. The OOD waits on the Navigator, which waits on Guidance, which waits on the Homing Calculator. The sequence of router tasks allows homing guidance commands to get through while these other tasks await the completion of homing.

When a system problem occurs, multitasking is required to maintain control of the vehicle during route replanning. The Strategic level issues the command to start replanning to the Tactical level when a system problem is encountered. The Navigator must send a command to the Mission Replanner to start replanning simultaneously with a command to Guidance to loiter. In Ada, this is accomplished by first issuing a parameterless entry call to the Mission Replanner, which has a simple accept call and a separate set of statements to perform replanning. This entry call is followed by an entry call to Guidance to loiter, and the Navigator task is suspended until loitering is done. Suspension of the Navigator task requires Guidance to utilize the router tasks to send commands to the Execution level as in the case of the *search* and *task* modes. The replanning operation and loitering guidance continue in parallel until replanning is done with the Ada RTS providing the scheduling of the two tasks. The situation is illustrated in Figure 10. Thus, *inter-object* concurrency is provided in addition to the intra-object concurrency provided by the relay tasks.

Operation of the implementation in a mission -oriented environment is discussed in the next chapter.
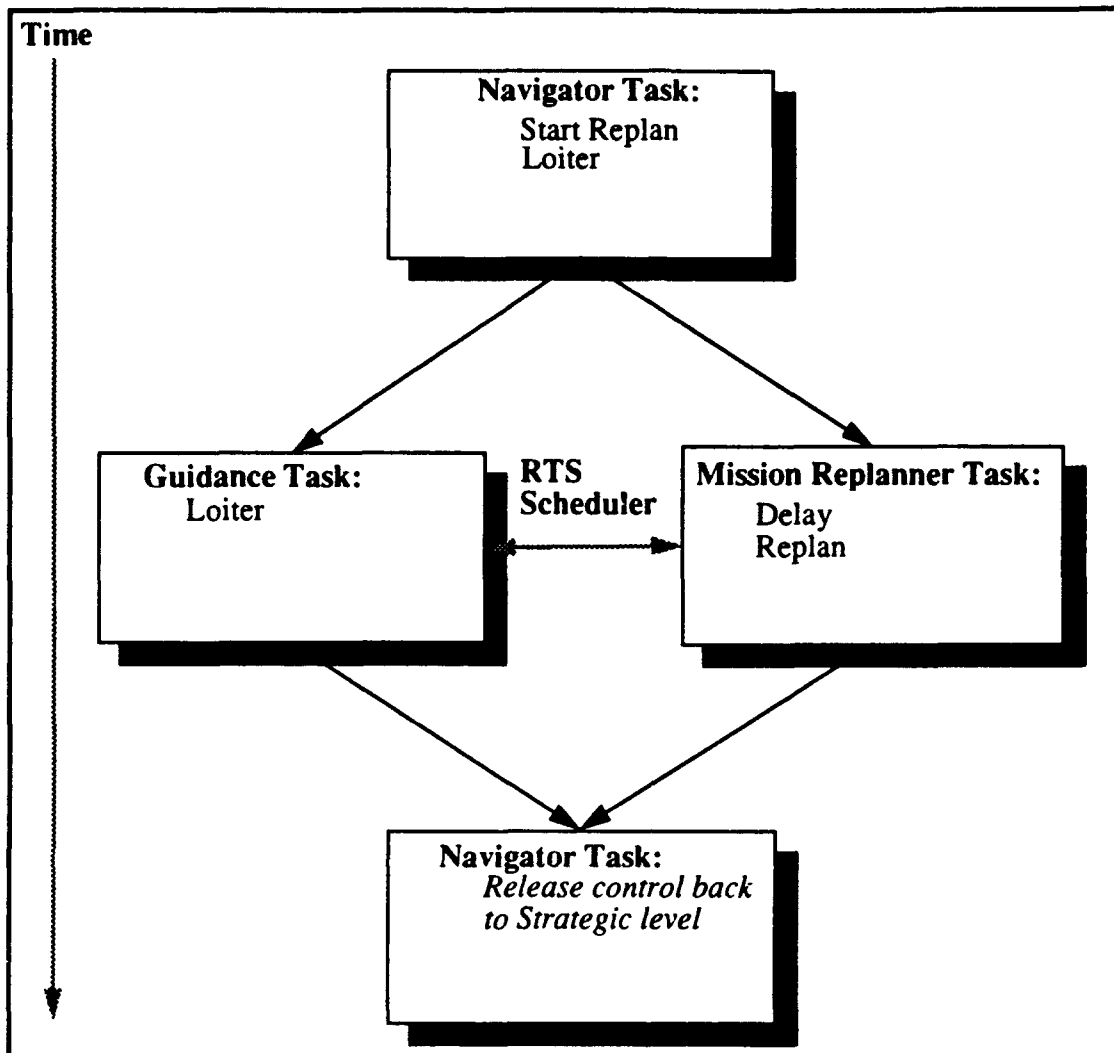
Figure 10 Multitasking in Route Replanning

VI. TESTING AND RESULTS

A. INTRODUCTION

Testing the Tactical level implementation was accomplished using the simulation facilities available in the laboratory. The simulation environment was set up to reflect the actual hardware and software configuration on the NPS AUV. Mission scenarios were then developed to represent the conditions of the search-and-rescue mission described in Chapter V. The AUV graphical simulator provided for the entry of waypoints and obstacles using Cartesian coordinates in a visual model of the NPS pool to support this scenario development [Ong90].

B. SIMULATION ENVIRONMENT

To test the implementation, modifications were made to the configuration described in Chapter II to reproduce the environment on the vehicle. Two processors were used to represent the two processors on the actual vehicle. The Strategic and Tactical levels were run together under the UNIX operating system on a Sun SPARCstation 3/180, corresponding to the Mission Control Computer. The Strategic level was coded in CLIPS-Ada, a preprocessor which compiles CLIPS code to Ada source code, to allow the Strategic and Tactical levels to reside on the same processor. A description of this CLIPS-Ada implementation and the code are presented in [Scho93]. The Tactical level was coded in Ada, as described in Chapter V. The Execution level used the same C code as the previous implementation and was again run under the IRIX operating system on a Silicon Graphics 4D/340VGX Workstation, corresponding to the Vehicle Control Computer. The two-processor test configuration is shown in Figure 11.

A sonar model was required for the simulation so that all phases of the mission could be tested. Sonar was simulated by adding code to the Sensory Receiver to track range and bearing to a target, which was represented by an obstacle entered into the World Model.

32
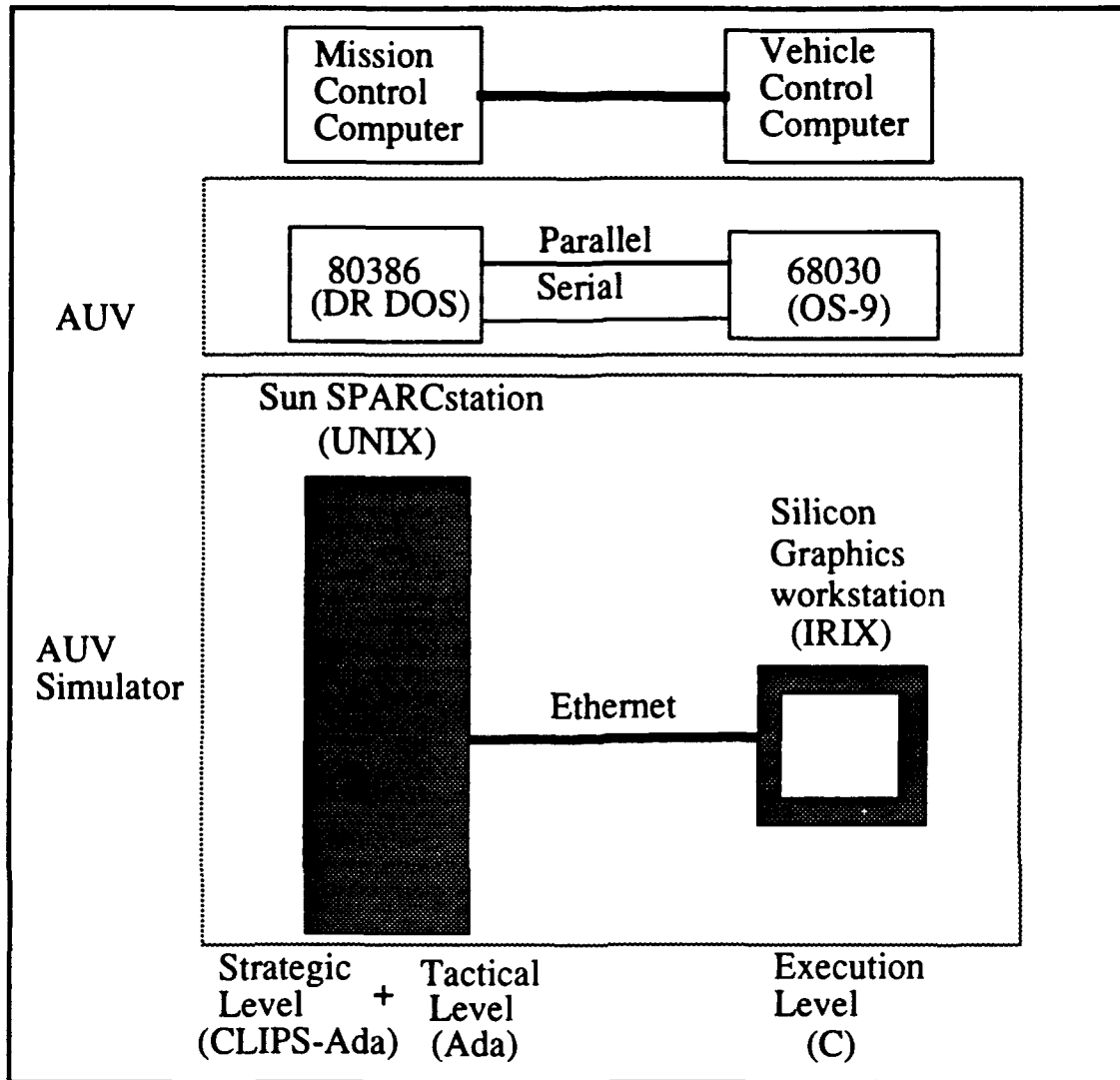
Figure 11 AUV Simulator Test Configuration and Vehicle Configuration

This modification allowed the *search* and *task* modes of the AUV to be demonstrated realistically.

A vehicle mode was entered along with each waypoint in the waypoint data file that the simulator read into the Mission Model. In this way, a vehicle mode could be selected at each waypoint based on the mission profile. Available choices for the vehicle mode include *transit*, *search*, and *return*[1].

---

1. *Task* is an invalid choice because this mode is automatically triggered by the successful completion of the *search* mode. When the search ends, homing begins, initiating the *task* mode.

C. SCENARIOS

1. Multi-Phase Mission

The first scenario tested was the straight four-phase search-and-rescue mission. For this scenario, a set of three waypoints and a single obstacle were chosen to cover the four mission phases. Figure 12 depicts the mission route. The vehicle was programmed for
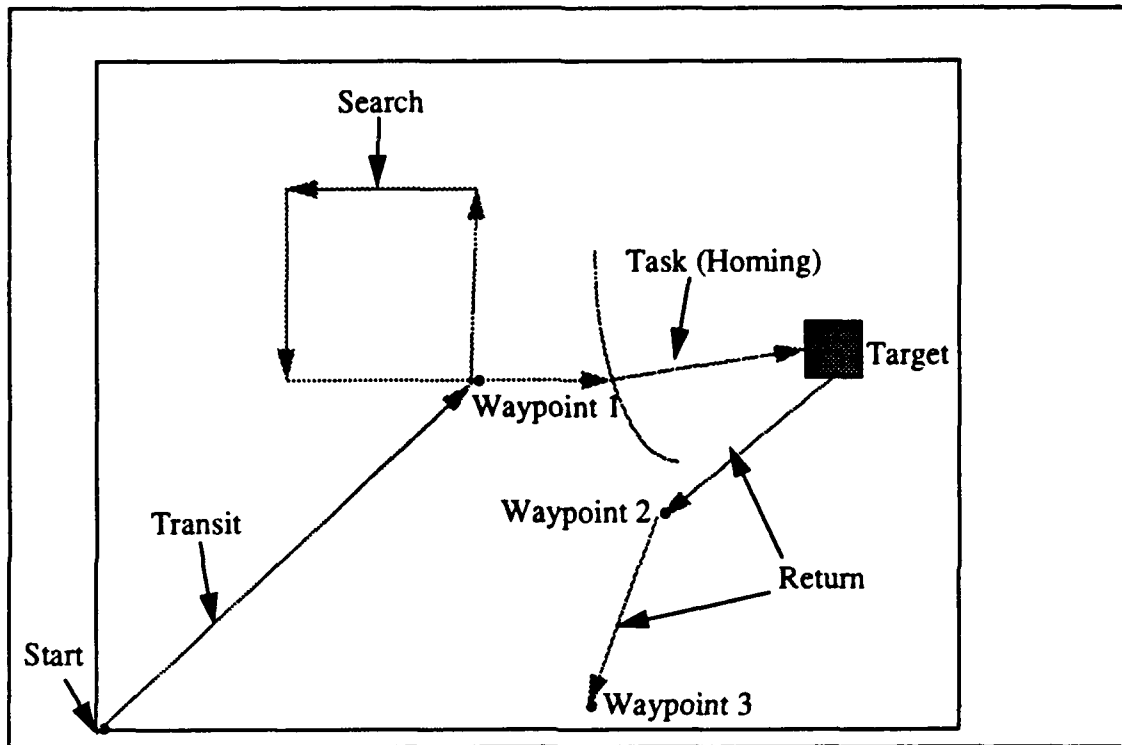
Figure 12 Multi-Phase Mission Scenario

the *transit* mode during the first leg, corresponding to the *transit* phase of the mission. The vehicle simply executes LOS guidance between waypoints in this mode. At the first waypoint, the vehicle was programmed to change to the *search* mode and execute an expanding box search pattern, corresponding to the *search* phase of the mission. The vehicle was then set to transition automatically to its *task* mode, corresponding to the *task* phase of the mission. The vehicle executes homing guidance in this mode with the obstacle as its target. The vehicle completes the task upon reaching its target. After reaching the

target, the vehicle was programmed to change to the *return* mode for the last two legs, corresponding to the *return* phase of the mission.

2. Multi-Phase Mission With Route Replanning

This scenario used the same mission route and vehicle modes as the first one. A low thrust level, simulating a thruster system problem, was programmed to occur during the transit phase. When faced with such a problem, the vehicle simultaneously loiters and shortens its mission route to insure it reaches its final goal before system degradation becomes too serious. Route replanning is accomplished in this implementation by sending a message to the Mission Model requesting a shortened route. In reality, the Mission Replanner would d ermine this shortened route and pass the modified waypoint data to the Mission Model in the message. The vehicle was programmed in this run to eliminate the *search* and *task* phases of the mission and to go straight to its *return* mode for the mission's *return* phase.

D. RESULTS

In the first scenario, the vehicle successfully executed all phases of the mission, transitioning through all its modes and reaching all waypoints and the target. There was a problem with communication between the Tactical and Execution levels due to the simulator protocol[2]. This problem arose because of the combination of the long line of communication to the Command Sender and the short line of communication to the Sensory Receiver under RBM. The problem was averted by using a short *delay* during the search and task modes.

In the second scenario, the vehicle accomplished both of its simultaneous tasks. It loitered in place after detecting the system problem for the time of the programmed *delay*,

---

2. The simulator requires an even balance between transmissions and receptions. Whenever it sends a set of data, it must receive a command packet before it can send another set. The actual vehicle is not subject to this constraint.

proceeded to the first waypoint, transitioned to the *return* mode, and completed the *return* phase of the mission.

Traces of the execution of the Tactical level code under these two mission scenariosare found in Appendix B. A user's guide for the AUV simulator is provided in Appendix C.

VII. CONCLUSIONS AND FUTURE WORK

In this thesis, a concurrent, object-based implementation is developed and evaluated for the Tactical level of the Rational Behavior Model. Previous work in this area has focused on object-oriented implementation exclusively or minimal use of concurrent programming facilities. However, the Tactical level is the essential bridge between the top and bottom levels of RBM, and it must handle concurrent, as well as sequential, operations among its objects for the success of the model in practice. In the absence of a programming language that combines object-oriented features with constructs for concurrency, Ada remains the best choice for an implementation of the Tactical level. The Tactical level implementation in this work uses *relay* tasks for intra-object concurrency to handle multiple phases of a mission and parameterless task entry calls for inter-object concurrency to handle route replanning. Both of these mechanisms insure control of the vehicle is maintained throughout a mission. Simulation testing shows that control of the vehicle is indeed maintained continuously with such an implementation even in the face of time-consuming tasks.

A. RESEARCH CONTRIBUTIONS

This research has numerous benefits. First, it provides an example for implementing multitasking to aid in the control of autonomous vehicles. This capability is very important for them to reflect rational behavior. Second, this work reiterates the value of the object-oriented paradigm for this problem domain. Object-oriented techniques increase the modularity and simplicity of the Tactical level implementation, improving the reliability and maintainability of the software. Finally, this research reveals the weakness of current programming languages in integrating concurrency with the object-oriented paradigm.

B. SUGGESTIONS FOR FUTURE RESEARCH

There are many ways to build on the foundation this research has established. One area that was started in this work but not completed was transferring the simulator

implementation to the actual vehicle and testing it. Another area for future research is developing a more complete implementation and testing how much load one processor can bear. Extensive use of Ada tasks, especially such intermediary tasks as *relay* tasks, imposes a significant amount of overhead, and time did not permit a full analysis of this factor in this work. Finally, distributed implementations of the Tactical level represent fertile ground for future work, since the NPS AUV is fitted with a transputer board. Progress in any of these areas would make the Tactical level a stronger, more robust link in RBM.

APPENDIX A. TACTICAL LEVEL SOURCE CODE

```
--Title       : tac_lv_s.a
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specifications for procedures for Ada side of Clips-Ada/Ada
--              interface for simulator version of AUV Tactical level
-------------------------------------------------------------------------

package TACTICAL_LEVEL1 is

  procedure READY_VEHICLE_FOR_LAUNCH(GOAL_FLAG : in out INTEGER);
  procedure SELECT_FIRST_WAYPOINT(GOAL_FLAG : in out INTEGER);
  procedure ALERT_USER(GOAL_FLAG : in out INTEGER);
  procedure IN_TRANSIT_P(GOAL_FLAG : in out INTEGER);
  procedure TRANSIT_DONE_P(GOAL_FLAG : in out INTEGER);
  procedure IN_SEARCH_P(GOAL_FLAG : in out INTEGER);
  procedure SEARCH_DONE_P(GOAL_FLAG : in out INTEGER);
  procedure IN_TASK_P(GOAL_FLAG : in out INTEGER);
  procedure TASK_DONE_P(GOAL_FLAG : in out INTEGER);
  procedure IN_RETURN_P(GOAL_FLAG : in out INTEGER);
  procedure RETURN_DONE_P(GOAL_FLAG : in out INTEGER);
  procedure WAIT_FOR_RECOVERY(GOAL_FLAG : in out INTEGER);
  procedure SURFACE(GOAL_FLAG : in out INTEGER);
  procedure DO_SEARCH_PATTERN(GOAL_FLAG : in out INTEGER);
  procedure HOMING(GOAL_FLAG : in out INTEGER);
  procedure DROP_PACKAGE(GOAL_FLAG : in out INTEGER);
  procedure GET_GPS_FIX(GOAL_FLAG : in out INTEGER);
  procedure GET_NEXT_WAYPOINT(GOAL_FLAG : in out INTEGER);
  procedure SEND_SETPOINTS_AND_MODES(GOAL_FLAG : in out INTEGER);
  procedure REACH_WAYPOINT_P(GOAL_FLAG : in out INTEGER);
  procedure GPS_NEEDED_P(GOAL_FLAG : in out INTEGER);
  procedure UNKNOWN_OBSTACLE_P(GOAL_FLAG : in out INTEGER);
  procedure LOG_NEW_OBSTACLE(GOAL_FLAG : in out INTEGER);
  procedure LOITER(GOAL_FLAG : in out INTEGER);
  procedure START_LOCAL_REPLANNER(GOAL_FLAG : in out INTEGER);
  procedure START_GLOBAL_REPLANNER(GOAL_FLAG : in out INTEGER);
  procedure POWER_GONE_P(GOAL_FLAG : in out INTEGER);
  procedure COMPUTER_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure PROPULSION_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure STEERING_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure DIVING_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure BUOYANCY_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure THRUSTER_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER);
  procedure LEAK_TEST_P(GOAL_FLAG : in out INTEGER);
  procedure PAYLOAD_PROB_P(GOAL_FLAG : in out INTEGER);
end TACTICAL_LEVEL1;
```

```
--Title       : tac_lv_b.a
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Procedures for Ada side of CLIPS-Ada/Ada interface for
--               simulator version of AUV tactical level
-------------------------------------------------------------------------------------

with TEXT_IO, OOD;
use  TEXT_IO, OOD;

package body TACTICAL_LEVEL1 is

 package FLOAT_INOUT is new FLOAT_IO(FLOAT);
 package INTEGER_INOUT is new INTEGER_IO(INTEGER);
 use FLOAT_INOUT, INTEGER_INOUT;

 procedure READY_VEHICLE_FOR_LAUNCH(GOAL_FLAG : in out INTEGER) is
 begin
  THE_OOD.CREATE;
  THE_OOD.READY_VEHICLE_FOR_LAUNCH(GOAL_FLAG);
  PUT("READY_VEHICLE_FOR_LAUNCH GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
 end READY_VEHICLE_FOR_LAUNCH;

 procedure SELECT_FIRST_WAYPOINT(GOAL_FLAG : in out INTEGER) is
 begin
  THE_OOD.SELECT_FIRST_WAYPOINT(GOAL_FLAG);
  PUT("SELECT_FIRST_WAYPOINT GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
 end SELECT_FIRST_WAYPOINT;

 procedure ALERT_USER(GOAL_FLAG : in out INTEGER) is
 begin
  loop
   THE_OOD.ALERT_USER(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
  PUT("ALERT_USER GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
 end ALERT_USER;

 procedure IN_TRANSIT_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
 begin
  loop
   THE_OOD.IN_TRANSIT_P(GOAL_FLAG, RETURN_FLAG);
```

40

```
    exit when RETURN_FLAG = 1;
   end loop;
   PUT("IN_TRANSIT_P GOAL FLAG = ");
   PUT(GOAL_FLAG, WIDTH=>3);
   NEW_LINE;
end IN_TRANSIT_P;

procedure TRANSIT_DONE_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
 loop
   THE_OOD.TRANSIT_DONE_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("TRANSIT_DONE_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end TRANSIT_DONE_P;

procedure IN_SEARCH_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
 loop
   THE_OOD.IN_SEARCH_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("IN_SEARCH_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end IN_SEARCH_P;

procedure SEARCH_DONE_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
 loop
   THE_OOD.SEARCH_DONE_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("SEARCH_DONE_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end SEARCH_DONE_P;

procedure IN_TASK_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
 loop
   THE_OOD.IN_TASK_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("IN_TASK_P GOAL FLAG = ");
```

41

```
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end IN_TASK_P;

procedure TASK_DONE_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
  loop
   THE_OOD.TASK_DONE_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("TASK_DONE_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end TASK_DONE_P;

procedure IN_RETURN_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
  loop
   THE_OOD.IN_RETURN_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("IN_RETURN_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end IN_RETURN_P;

procedure RETURN_DONE_P(GOAL_FLAG : in out INTEGER) is
  RETURN_FLAG : INTEGER := 0;
begin
  loop
   THE_OOD.RETURN_DONE_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
  end loop;
  PUT("RETURN_DONE_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end RETURN_DONE_P;

procedure WAIT_FOR_RECOVERY(GOAL_FLAG : in out INTEGER) is
begin
  loop
   THE_OOD.WAIT_FOR_RECOVERY(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
  PUT("WAIT_FOR_RECOVERY GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end WAIT_FOR_RECOVERY;
```

```
procedure SURFACE(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.SURFACE(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
 PUT("SURFACE GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end SURFACE;

procedure DO_SEARCH_PATTERN(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.DO_SEARCH_PATTERN(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
 PUT("DO_SEARCH_PATTERN GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end DO_SEARCH_PATTERN;

procedure HOMING(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.HOMING(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
 PUT("HOMING GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end HOMING;

procedure DROP_PACKAGE(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.DROP_PACKAGE(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
 PUT("DROP_PACKAGE GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end DROP_PACKAGE;

procedure GET_GPS_FIX(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.GET_GPS_FIX(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
  end loop;
 PUT("GET_GPS_FIX GOAL FLAG = ");
```

```
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end GET_GPS_FIX;

procedure GET_NEXT_WAYPOINT(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.GET_NEXT_WAYPOINT(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
 end loop;
 PUT("GET_NEXT_WAYPOINT GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end GET_NEXT_WAYPOINT;

procedure SEND_SETPOINTS_AND_MODES(GOAL_FLAG : in out INTEGER) is
begin
 loop
  THE_OOD.SEND_SETPOINTS_AND_MODES(GOAL_FLAG);
   exit when GOAL_FLAG = 1;
 end loop;
 PUT("SEND_SETPOINTS_AND_MODES GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end SEND_SETPOINTS_AND_MODES;

procedure REACH_WAYPOINT_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.REACH_WAYPOINT_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
 end loop;
 PUT("REACH_WAYPOINT_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end REACH_WAYPOINT_P;

procedure GPS_NEEDED_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.GPS_NEEDED_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
 end loop;
 PUT("GPS_NEEDED_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end GPS_NEEDED_P;

procedure UNKNOWN_OBSTACLE_P(GOAL_FLAG : in out INTEGER) is
```

```
    RETURN_FLAG : INTEGER := 0;
begin
  loop
    THE_OOD.UNKNOWN_OBSTACLE_P(GOAL_FLAG, RETURN_FLAG);
    exit when RETURN_FLAG = 1;
  end loop;
  PUT("UNKNOWN_OBSTACLE_P GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end UNKNOWN_OBSTACLE_P;

procedure LOG_NEW_OBSTACLE(GOAL_FLAG : in out INTEGER) is
begin
  loop
    THE_OOD.LOG_NEW_OBSTACLE(GOAL_FLAG);
    exit when GOAL_FLAG = 1;
  end loop;
  PUT("LOG_NEW_OBSTACLE GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end LOG_NEW_OBSTACLE;

procedure LOITER(GOAL_FLAG : in out INTEGER) is
begin
  loop
    THE_OOD.LOITER(GOAL_FLAG);
    exit when GOAL_FLAG = 1;
  end loop;
  PUT("LOITER GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end LOITER;

procedure START_LOCAL_REPLANNER(GOAL_FLAG : in out INTEGER) is
begin
  loop
    THE_OOD.START_LOCAL_REPLANNER(GOAL_FLAG);
    exit when GOAL_FLAG = 1;
  end loop;
  PUT("START_LOCAL_REPLANNER GOAL FLAG = ");
  PUT(GOAL_FLAG, WIDTH=>3);
  NEW_LINE;
end START_LOCAL_REPLANNER;

procedure START_GLOBAL_REPLANNER(GOAL_FLAG : in out INTEGER) is
begin
  loop
    THE_OOD.START_GLOBAL_REPLANNER(GOAL_FLAG);
    exit when GOAL_FLAG = 1;
  end loop;
  PUT("START_GLOBAL_REPLANNER GOAL FLAG = ");
```

```ada
   PUT(GOAL_FLAG, WIDTH=>3);
   NEW_LINE;
end START_GLOBAL_REPLANNER;

procedure POWER_GONE_P(GOAL  FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.POWER_GONE_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("POWER_GONE_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end POWER_GONE_P;

procedure COMPUTER_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.COMPUTER_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("COMPUTER_SYSTEM_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end COMPUTER_SYSTEM_PROB_P;

procedure PROPULSION_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.PROPULSION_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("PROPULSION_SYSTEM_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end PROPULSION_SYSTEM_PROB_P;

procedure STEERING_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.STEERING_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("STEERING_SYSTEM_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end STEERING_SYSTEM_PROB_P;
```

```
procedure DIVING_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.DIVING_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("DIVING_SYSTEM_PROBLEM_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end DIVING_SYSTEM_PROB_P;

procedure BUOYANCY_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.BUOYANCY_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("BUOYANCY_SYSTEM_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end BUOYANCY_SYSTEM_PROB_P;

procedure THRUSTER_SYSTEM_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.THRUSTER_SYSTEM_PROB_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("THRUSTER_SYSTEM_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end THRUSTER_SYSTEM_PROB_P;

procedure LEAK_TEST_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
begin
 loop
  THE_OOD.LEAK_TEST_P(GOAL_FLAG, RETURN_FLAG);
  exit when RETURN_FLAG = 1;
 end loop;
 PUT("LEAK_TEST_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end LEAK_TEST_P;

procedure PAYLOAD_PROB_P(GOAL_FLAG : in out INTEGER) is
 RETURN_FLAG : INTEGER := 0;
```

```
begin
 loop
   THE_OOD.PAYLOAD_PROB_P(GOAL_FLAG, RETURN_FLAG);
   exit when RETURN_FLAG = 1;
 end loop;
 PUT("PAYLOAD_PROB_P GOAL FLAG = ");
 PUT(GOAL_FLAG, WIDTH=>3);
 NEW_LINE;
end PAYLOAD_PROB_P;

end TACTICAL_LEVEL1;
```

```
--Title        : ood_s.a
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Specification for OOD task
-------------------------------------------------------------------------------

package OOD is

  task THE_OOD is

    entry CREATE;
    entry READY_VEHICLE_FOR_LAUNCH(G_FLAG : out INTEGER);
    entry SELECT_FIRST_WAYPOINT(G_FLAG : out INTEGER);
    entry ALERT_USER(G_FLAG : out INTEGER);
    entry IN_TRANSIT_P(G_FLAG, R_FLAG : out INTEGER);
    entry TRANSIT_DONE_P(G_FLAG, R_FLAG : out INTEGER);
    entry IN_SEARCH_P(G_FLAG, R_FLAG : out INTEGER);
    entry SEARCH_DONE_P(G_FLAG, R_FLAG : out INTEGER);
    entry IN_TASK_P(G_FLAG, R_FLAG : out INTEGER);
    entry TASK_DONE_P(G_FLAG, R_FLAG : out INTEGER);
    entry IN_RETURN_P(G_FLAG, R_FLAG : out INTEGER);
    entry RETURN_DONE_P(G_FLAG, R_FLAG : out INTEGER);
    entry WAIT_FOR_RECOVERY(G_FLAG : out INTEGER);
    entry SURFACE(G_FLAG : out INTEGER);
    entry DO_SEARCH_PATTERN(G_FLAG : out INTEGER);
    entry HOMING(G_FLAG : out INTEGER);
    entry DROP_PACKAGE(G_FLAG : out INTEGER);
    entry GET_GPS_FIX(G_FLAG : out INTEGER);
    entry GET_NEXT_WAYPOINT(G_FLAG : out INTEGER);
    entry SEND_SETPOINTS_AND_MODES(G_FLAG : out INTEGER);
    entry REACH_WAYPOINT_P(G_FLAG, R_FLAG : out INTEGER);
    entry GPS_NEEDED_P(G_FLAG, R_FLAG : out INTEGER);
    entry UNKNOWN_OBSTACLE_P(G_FLAG, R_FLAG : out INTEGER);
    entry LOG_NEW_OBSTACLE(G_FLAG : out INTEGER);
    entry LOITER(G_FLAG : out INTEGER);
    entry START_LOCAL_REPLANNER(G_FLAG : out INTEGER);
    entry START_GLOBAL_REPLANNER(G_FLAG :out INTEGER);
    entry POWER_GONE_P(G_FLAG, R_FLAG : out INTEGER);
    entry COMPUTER_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry PROPULSION_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry STEERING_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry DIVING_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry BUOYANCY_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry THRUSTER_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER);
    entry LEAK_TEST_P(G_FLAG, R_FLAG : out INTEGER);
    entry PAYLOAD_PROB_P(G_FLAG, R_FLAG : out INTEGER);
  end THE_OOD;

end OOD;
```

```
--Title       : ood_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for OOD task
-----------------------------------------------------------------------------

with TEXT_IO, COMMAND_SENDER, MISSION_MODEL, WORLD_MODEL,
SENSORY_RECEIVER,
   OOD_ROUTER, NAVIGATOR, ENGINEERING, WEAPONS;
use  TEXT_IO, MISSION_MODEL, WORLD_MODEL, SENSORY_RECEIVER,OOD_ROUTER,
   NAVIGATOR, ENGINEERING, WEAPONS;

package body OOD is

-----------------------------------------------------------------------------
--Task to handle OOD functions
-----------------------------------------------------------------------------


task body THE_OOD is

   GOAL_FLAG_1  : BOOLEAN := FALSE;  --Flags for lower level objects
   RETURN_FLAG_1 : BOOLEAN := FALSE;
   OOD_X : FLOAT;
   OOD_Y : FLOAT;
   OOD_DEPTH : FLOAT;
   OOD_HEADING : FLOAT;
   OOD_SPEED : FLOAT;
   OOD_MODE : INTEGER;

begin
  loop
   --Flags for lower level objects are checked for each command or predicate
   --query and then the result is sent back to the Strategic level
   select
     --Create tactical level objects
     accept CREATE;
      PUT_LINE("Creating OOD");
      THE_MISSION_MODEL.CREATE;
      THE_WORLD_MODEL.CREATE;
      THE_SENSORY_RECEIVER.CREATE;
      THE_OOD_ROUTER.CREATE;
      THE_NAVIGATOR.CREATE;
      THE_ENGINEERING.CREATE;
      THE_WEAPONS.CREATE;
   or
     accept READY_VEHICLE_FOR_LAUNCH(G_FLAG : out INTEGER) do
      THE_WORLD_MODEL.INITIALIZE(GOAL_FLAG_1);
      if (GOAL_FLAG_1 = TRUE) then
        THE_MISSION_MODEL.INITIALIZE(GOAL_FLAG_1);
```

50

```
      if (GOAL_FLAG_1 = TRUE) then
        G_FLAG := 1;
        GOAL_FLAG_1 := FALSE;
      else
        G_FLAG := 0;
      end if;
    else
      G_FLAG := 0;
    end if;
  end READY_VEHICLE_FOR_LAUNCH;
or
  accept SELECT_FIRST_WAYPOINT(G_FLAG : out INTEGER) do
    THE_NAVIGATOR.SELECT_FIRST_WAYPOINT(GOAL_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
  end SELECT_FIRST_WAYPOINT;
or
  accept ALERT_USER(G_FLAG : out INTEGER) do
    PUT_LINE("Failure detected during initialization.");
    G_FLAG := 1;
  end ALERT_USER;
or
  accept IN_TRANSIT_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.IN_TRANSIT_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end IN_TRANSIT_P;
or
  accept TRANSIT_DONE_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.TRANSIT_DONE_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
```

```
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end TRANSIT_DONE_P;
or
  accept IN_SEARCH_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.IN_SEARCH_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end IN_SEARCH_P;
or
  accept SEARCH_DONE_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.SEARCH_DONE_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end SEARCH_DONE_P;
or
  accept IN_TASK_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.IN_TASK_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
```

```
      R_FLAG := 0;
    end if;
  end IN_TASK_P;
or
  accept TASK_DONE_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.TASK_DONE_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end TASK_DONE_P;
or
  accept IN_RETURN_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.IN_RETURN_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end IN_RETURN_P;
or
  accept RETURN_DONE_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_MISSION_MODEL.RETURN_DONE_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end RETURN_DONE_P;
```

```
or
 accept WAIT_FOR_RECOVERY(G_FLAG : out INTEGER) do
  THE_NAVIGATOR.WAIT_FOR_RECOVERY(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end WAIT_FOR_RECOVERY;
or
 accept SURFACE(G_FLAG : out INTEGER) do
  THE_NAVIGATOR.SURFACE(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end SURFACE;
or
 accept DO_SEARCH_PATTERN(G_FLAG : out INTEGER) do
  THE_NAVIGATOR.DO_SEARCH_PATTERN(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end DO_SEARCH_PATTERN;
or
 accept HOMING(G_FLAG : out INTEGER) do
  THE_NAVIGATOR.DO_HOMING(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end HOMING;
or
 accept DROP_PACKAGE(G_FLAG : out INTEGER) do
  THE_WEAPONS.DROP_PACKAGE(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end DROP_PACKAGE;
or
```

```
accept GET_GPS_FIX(G_FLAG : out INTEGER) do
 THE_NAVIGATOR.GET_GPS_FIX(GOAL_FLAG_1);
 if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
 else
   G_FLAG := 0;
 end if;
end GET_GPS_FIX;
or
 accept GET_NEXT_WAYPOINT(G_FLAG : out INTEGER) do
  THE_NAVIGATOR.GET_NEXT_WAYPOINT(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
    G_FLAG := 1;
    GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
 end GET_NEXT_WAYPOINT;
or
 accept SEND_SETPOINTS_AND_MODES(G_FLAG : out INTEGER) do
  select
  THE_NAVIGATOR.SEND_SETPOINTS_AND_MODES(GOAL_FLAG_1);
  or
   delay 1.0;
  end select;
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
  else
   G_FLAG := 0;
  end if;
 end SEND_SETPOINTS_AND_MODES;
or
 accept REACH_WAYPOINT_P(G_FLAG, R_FLAG : out INTEGER) do
  THE_NAVIGATOR.REACH_WAYPOINT_P(GOAL_FLAG_1, RETURN_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
  else
   G_FLAG := 0;
  end if;
  if (RETURN_FLAG_1 = TRUE) then
   R_FLAG := 1;
   RETURN_FLAG_1 := FALSE;
  else
   R_FLAG := 0;
  end if;
 end REACH_WAYPOINT_P;
or
 accept GPS_NEEDED_P(G_FLAG, R_FLAG : out INTEGER) do
```

```
      THE_NAVIGATOR.GPS_NEEDED_P(GOAL_FLAG_1, RETURN_FLAG_1);
      if (GOAL_FLAG_1 = TRUE) then
        G_FLAG := 1;
        GOAL_FLAG_1 := FALSE;
      else
        G_FLAG := 0;
      end if;
      if (RETURN_FLAG_1 = TRUE) then
        R_FLAG := 1;
        RETURN_FLAG_1 := FALSE;
      else
        R_FLAG := 0;
      end if;
    end GPS_NEEDED_P;
or
  accept UNKNOWN_OBSTACLE_P(G_FLAG, R_FLAG : out INTEGER) do
    THE_NAVIGATOR.UNKNOWN_OBSTACLE_P(GOAL_FLAG_1, RETURN_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end UNKNOWN_OBSTACLE_P;
or
  accept LOG_NEW_OBSTACLE(G_FLAG : out INTEGER) do
    THE_NAVIGATOR.LOG_NEW_OBSTACLE(GOAL_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
  end LOG_NEW_OBSTACLE;
or
  accept LOITER(G_FLAG : out INTEGER) do
    G_FLAG := 1;
  end LOITER;
or
  accept START_LOCAL_REPLANNER(G_FLAG : out INTEGER) do
    THE_NAVIGATOR.START_LOCAL_REPLANNER(GOAL_FLAG_1);
    if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
    else
```

```
    G_FLAG := 0;
  end if;
 end START_LOCAL_REPLANNER;
or
 accept START_GLOBAL_REPLANNER(G_FLAG :out INTEGER) do
  THE_NAVIGATOR.START_GLOBAL_REPLANNER(GOAL_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
  else
   G_FLAG := 0;
  end if;
 end START_GLOBAL_REPLANNER;
or
 accept POWER_GONE_P(G_FLAG, R_FLAG : out INTEGER) do
  THE_ENGINEERING.POWER_GONE_P(GOAL_FLAG_1, RETURN_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
  else
    G_FLAG := 0;
  end if;
  if (RETURN_FLAG_1 = TRUE) then
   R_FLAG := 1;
   RETURN_FLAG_1 := FALSE;
  else
   R_FLAG := 0;
  end if;
 end POWER_GONE_P;

or
 accept COMPUTER_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
  THE_ENGINEERING.COMPUTER_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
   GOAL_FLAG_1 := FALSE;
  else
   G_FLAG := 0;
  end if;
  if (RETURN_FLAG_1 = TRUE) then
   R_FLAG := 1;
   RETURN_FLAG_1 := FALSE;
  else
   R_FLAG := 0;
  end if;
 end COMPUTER_SYSTEM_PROB_P;
or
 accept PROPULSION_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
  THE_ENGINEERING.PROPULSION_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
  if (GOAL_FLAG_1 = TRUE) then
   G_FLAG := 1;
```

```
      GOAL_FLAG_1 := FALSE;
    else
      G_FLAG := 0;
    end if;
    if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
    else
      R_FLAG := 0;
    end if;
  end PROPULSION_SYSTEM_PROB_P;
or
 accept STEERING_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
   THE_ENGINEERING.STEERING_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
   if (GOAL_FLAG_1 = TRUE) then
     G_FLAG := 1;
     GOAL_FLAG_1 := FALSE;
   else
     G_FLAG := 0;
   end if;
   if (RETURN_FLAG_1 = TRUE) then
     R_FLAG := 1;
     RETURN_FLAG_1 := FALSE;
   else
     R_FLAG := 0;
   end if;
  end STEERING_SYSTEM_PROB_P;
or
 accept DIVING_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
   THE_ENGINEERING.DIVING_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
   if (GOAL_FLAG_1 = TRUE) then
     G_FLAG := 1;
     GOAL_FLAG_1 := FALSE;
   else
     G_FLAG := 0;
   end if;
   if (RETURN_FLAG_1 = TRUE) then
     R_FLAG := 1;
     RETURN_FLAG_1 := FALSE;
   else
     R_FLAG := 0;
   end if;
  end DIVING_SYSTEM_PROB_P;
or
 accept BUOYANCY_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
   THE_ENGINEERING.BUOYANCY_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
   if (GOAL_FLAG_1 = TRUE) then
     G_FLAG := 1;
     GOAL_FLAG_1 := FALSE;
   else
     G_FLAG := 0;
```

```
     end if;
     if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
     else
      R_FLAG := 0;
     end if;
    end BUOYANCY_SYSTEM_PROB_P;
  or
    accept THRUSTER_SYSTEM_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
     THE_ENGINEERING.THRUSTER_SYSTEM_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
     if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
     else
      G_FLAG := 0;
     end if;
     if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
     else
      R_FLAG := 0;
     end if;
    end THRUSTER_SYSTEM_PROB_P;
  or
    accept LEAK_TEST_P(G_FLAG, R_FLAG : out INTEGER) do
     THE_ENGINEERING.LEAK_TEST_P(GOAL_FLAG_1, RETURN_FLAG_1);
     if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
     else
      G_FLAG := 0;
     end if;
     if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
      RETURN_FLAG_1 := FALSE;
     else
      R_FLAG := 0;
     end if;
    end LEAK_TEST_P;
  or
    accept PAYLOAD_PROB_P(G_FLAG, R_FLAG : out INTEGER) do
     THE_ENGINEERING.PAYLOAD_PROB_P(GOAL_FLAG_1, RETURN_FLAG_1);
     if (GOAL_FLAG_1 = TRUE) then
      G_FLAG := 1;
      GOAL_FLAG_1 := FALSE;
     else
      G_FLAG := 0;
     end if;
     if (RETURN_FLAG_1 = TRUE) then
      R_FLAG := 1;
```

```
        RETURN_FLAG_1 := FALSE;
      else
        R_FLAG := 0;
      end if;
    end PAYLOAD_PROB_P;
  end select;
  end loop;
 end THE_OOD;

end OOD;
```

```
--Title       : ood_r_s.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for OOD Router task
--------------------------------------------------------------------------------------------

package OOD_ROUTER is

 task THE_OOD_ROUTER is
   entry CREATE;
   entry TAKE_NAV_COMMANDS(WAYPOINT_X : in FLOAT;
                           WAYPOINT_Y : in FLOAT;
                           NAV_HEADING : in FLOAT;
                           NAV_SPEED : in FLOAT;
                           NAV_DEPTH : in FLOAT;
                           NAV_MODE : in INTEGER);
   entry TAKE_GUIDANCE_COMMANDS(NAV_HEADING : in FLOAT;
                                NAV_MODE : in INTEGER);
 end THE_OOD_ROUTER;

end OOD_ROUTER;
```

61

```
--Title      : ood_r_b.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for OOD Router task
---------------------------------------------------------------------------------

with TEXT_IO, MISSION_MODEL, COMMAND_SENDER;
use  TEXT_IO;

package body OOD_ROUTER is

---------------------------------------------------------------------------------
--Task to handle routing of requests to OOD, required to allow time-consuming
--tasks to continue (search, homing, replanning)
---------------------------------------------------------------------------------

task body THE_OOD_ROUTER is
 OOD_X : FLOAT;
 OOD_Y : FLOAT;
 OOD_HEADING : FLOAT;
 OOD_SPEED : FLOAT;
 OOD_DEPTH : FLOAT;
 OOD_MODE : INTEGER;
begin
 accept CREATE;
 PUT_LINE("Creating OOD ROUTER");
 loop
  select
   --Get Navigator commands to send to Command Sender
   accept TAKE_NAV_COMMANDS(WAYPOINT_X : in FLOAT;
                            WAYPOINT_Y : in FLOAT;
                            NAV_HEADING : in FLOAT;
                            NAV_SPEED : in FLOAT;
                            NAV_DEPTH : in FLOAT;
                            NAV_MODE : in INTEGER) do
    OOD_X := WAYPOINT_X;
    OOD_Y := WAYPOINT_Y;
    OOD_HEADING := NAV_HEADING;
    OOD_SPEED := NAV_SPEED;
    OOD_DEPTH := NAV_DEPTH;
    OOD_MODE := NAV_MODE;
   end TAKE_NAV_COMMANDS;
   COMMAND_SENDER.SEND(OOD_X, OOD_Y,OOD_HEADING, OOD_SPEED,
                       OOD_DEPTH, OOD_MODE);
  or
   accept TAKE_GUIDANCE_COMMANDS(NAV_HEADING : in FLOAT;
                                 NAV_MODE : in INTEGER) do
    OOD_HEADING := NAV_HEADING;
    OOD_MODE := NAV_MODE;
```

62

```
        end TAKE_GUIDANCE_COMMANDS;
        COMMAND_SENDER.SEND(OOD_X, OOD_Y, OOD_HEADING, OOD_SPEED,
                            OOD_DEPTH, OOD_MODE);
    end select;
  end loop;
 end THE_OOD_ROUTER;

end OOD_ROUTER;
```

```
--Title        : nav_b.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Specification for Navigator task
------------------------------------------------------------------------------------

package NAVIGATOR is

 task THE_NAVIGATOR is
   entry CREATE;
   entry SELECT_FIRST_WAYPOINT(G_FLAG_1 : out BOOLEAN);
   entry WAIT_FOR_RECOVERY(G_FLAG_1 : out BOOLEAN);
   entry SURFACE(G_FLAG_1 : out BOOLEAN);
   entry DO_SEARCH_PATTERN(G_FLAG_1 : out BOOLEAN);
   entry DO_HOMING(G_FLAG_1 : out BOOLEAN);
   entry GET_GPS_FIX(G_FLAG_1 : out BOOLEAN);
   entry GPS_NEEDED_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry GET_NEXT_WAYPOINT(G_FLAG_1 : out BOOLEAN);
   entry REACH_WAYPOINT_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry SEND_SETPOINTS_AND_MODES(G_FLAG_1 : out BOOLEAN);
   entry UNKNOWN_OBSTACLE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry LOG_NEW_OBSTACLE(G_FLAG_1 : out BOOLEAN);
   entry START_LOCAL_REPLANNER(G_FLAG_1 : out BOOLEAN);
   entry START_GLOBAL_REPLANNER(G_FLAG_1 : out BOOLEAN);
 end THE_NAVIGATOR;

end NAVIGATOR;
```

```
with TEXT_IO, MATH, MISSION_MODEL, SENSORY_RECEIVER, OOD_ROUTER,
   NAVIGATOR_ROUTER, GUIDANCE, GPS_CONTROL, SONAR_CONTROL,
MISSION_REPLANNER;
use  TEXT_IO, MATH, MISSION_MODEL, SENSORY_RECEIVER, OOD_ROUTER,
   NAVIGATOR_ROUTER, GUIDANCE, GPS_CONTROL, SONAR_CONTROL,
MISSION_REPLANNER;

package body NAVIGATOR is

------------------------------------------------------------------------------------------
--Task to handle navigation functions
------------------------------------------------------------------------------------------

task body THE_NAVIGATOR is

  GOAL_FLAG_2  : BOOLEAN := FALSE;    --Flags for lower level objects
  RETURN_FLAG_2 : BOOLEAN := FALSE;
  STARTED : BOOLEAN := FALSE;         --Flag to start comm protocol
  REPEATED : BOOLEAN := FALSE;        --Flag to continue comm protocol
  NAV_X : FLOAT;
  NAV_Y : FLOAT;
  NAV_DEPTH : FLOAT;
  NAV_HEADING : FLOAT;
  NAV_SPEED : FLOAT;
  NAV_MODE : INTEGER;
  NAV_BEARING : FLOAT;
  NAV_RANGE : FLOAT;
  WAYPOINT_X : FLOAT;
  WAYPOINT_Y : FLOAT;
  WAYPOINT_DEPTH : FLOAT;
  EPSILON : constant FLOAT := 20.0;    --Tolerance for achieving waypoint
  SURFACE_LIMIT : constant FLOAT := 5.0; --Tolerance for Surface condition

begin
  --Create Navigator's subobjects
  accept CREATE;
  PUT_LINE("Creating NAVIGATOR");
  THE_NAVIGATOR_ROUTER.CREATE;
  THE_GUIDANCE.CREATE;
  THE_GPS_CONTROL.CREATE;
  THE_MISSION_REPLANNER.CREATE;
  THE_SONAR_CONTROL.CREATE;
  --Receive initial state and first waypoint
```

```
accept SELECT_FIRST_WAYPOINT(G_FLAG_1 : out BOOLEAN) do
  THE_MISSION_MODEL.GIVE_FIRST_WAYPOINT(NAV_X, NAV_Y, NAV_DEPTH,
NAV_MODE,
                    NAV_HEADING,NAV_SPEED, WAYPOINT_X,
                    WAYPOINT_Y, WAYPOINT_DEPTH);
  G_FLAG_1 := TRUE;
end SELECT_FIRST_WAYPOINT;
loop
 select
  accept WAIT_FOR_RECOVERY(G_FLAG_1 : out BOOLEAN) do
   G_FLAG_1 := TRUE;
  end WAIT_FOR_RECOVERY;
  --Loop under Tactical level control until signaled for mission
  --download
  loop
   --Delay to comply with simulator Tactical-Execution comm protocol
   --For every set of data received a set of commands must be sent
   delay 0.2;
   THE_SENSORY_RECEIVER.RECEIVE(NAV_X, NAV_Y, NAV_DEPTH, NAV_HEADING,
                NAV_BEARING, NAV_RANGE);
   WAYPOINT_DEPTH := 0.0;
   NAV_SPEED := 0.0;
   THE_OOD_ROUTER.TAKE_NAV_COMMANDS(WAYPOINT_X, WAYPOINT_Y,
NAV_HEADING,
                    NAV_SPEED, WAYPOINT_DEPTH, NAV_MODE);
  end loop;
 or
  accept SURFACE(G_FLAG_1 : out BOOLEAN) do
   loop
    --Simulator protocol delay
    delay 0.2;
    THE_SENSORY_RECEIVER.RECEIVE(NAV_X, NAV_Y, NAV_DEPTH, NAV_HEADING,
                NAV_BEARING, NAV_RANGE);
    exit when NAV_DEPTH < SURFACE_LIMIT;
    WAYPOINT_DEPTH := 0.0;
    THE_OOD_ROUTER.TAKE_NAV_COMMANDS(WAYPOINT_X, WAYPOINT_Y,
                    NAV_HEADING, NAV_SPEED,
                    WAYPOINT_DEPTH, NAV_MODE);
   end loop;
   G_FLAG_1 := TRUE;
  end SURFACE;
 or
  accept DO_SEARCH_PATTERN(G_FLAG_1 : out BOOLEAN) do
   THE_SONAR_CONTROL.DO_SEARCH_PATTERN(GOAL_FLAG_2, NAV_HEADING);
   if (GOAL_FLAG_2 = TRUE) then
    G_FLAG_1 := TRUE;
    GOAL_FLAG_2 := FALSE;
   else
    G_FLAG_1 := FALSE;
   end if;
  end DO_SEARCH_PATTERN;
```

```
or
 accept DO_HOMING(G_FLAG_1 : out BOOLEAN) do
  THE_GUIDANCE.DO_HOMING(GOAL_FLAG_2);
  if (GOAL_FLAG_2 = TRUE) then
   G_FLAG_1 := TRUE;
   GOAL_FLAG_2 := FALSE;
  else
   G_FLAG_1 := FALSE;
  end if;
 end DO_HOMING;
or
 accept GET_GPS_FIX(G_FLAG_1 : out BOOLEAN) do
  THE_GPS_CONTROL.GET_GPS_FIX(GOAL_FLAG_2);
  if (GOAL_FLAG_2 = TRUE) then
   G_FLAG_1 := TRUE;
   GOAL_FLAG_2 := FALSE;
  else
   G_FLAG_1 := FALSE;
  end if;
 end GET_GPS_FIX;
or
 accept GPS_NEEDED_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
  G_FLAG_1 := FALSE;
  R_FLAG_1 := TRUE;
 end GPS_NEEDED_P;
or
 accept GET_NEXT_WAYPOINT(G_FLAG_1 : out BOOLEAN) do
  THE_MISSION_MODEL.GIVE_NEXT_WAYPOINT(WAYPOINT_X.WAYPOINT_Y,
                                       WAYPOINT_DEPTH. NAV_SPEED,
                                       NAV_MODE);
  G_FLAG_1 := TRUE;
 end GET_NEXT_WAYPOINT;
or
 accept REACH_WAYPOINT_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
  if SQRT((WAYPOINT_X - NAV_X)**2 + (WAYPOINT_Y - NAV_Y)**2)
              < EPSILON then --Reached waypoint
   G_FLAG_1 := TRUE;
   PUT_LINE("*****At waypoint, coming to new heading*****");
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end REACH_WAYPOINT_P;
 --Do guidance in the background
 if not REPEATED then --Update navigation
  if STARTED then
   --Get current status values from Sensory Receiver
   THE_SENSORY_RECEIVER.RECEIVE(NAV_X, NAV_Y, NAV_DEPTH, NAV_HEADING,
                                NAV_BEARING, NAV_RANGE);
  end if;
  --Send for new commands from Guidance
```

```
      THE_GUIDANCE.GET_GUIDANCE_COMMANDS(NAV_X, NAV_Y, NAV_DEPTH,
                    NAV_HEADING,NAV_SPEED, WAYPOINT_X,
                    WAYPOINT_Y, WAYPOINT_DEPTH);
       STARTED := TRUE;
       REPEATED := TRUE;
      end if;
    or
     accept SEND_SETPOINTS_AND_MODES(G_FLAG_1 : out BOOLEAN) do
      THE_OOD_ROUTER.TAKE_NAV_COMMANDS(WAYPOINT_X, WAYPOINT_Y,
                        NAV_HEADING, NAV_SPEED,
                        NAV_DEPTH, NAV_MODE);
       G_FLAG_1 := TRUE;
       REPEATED := FALSE;
     end SEND_SETPOINTS_AND_MODES;
    or
     accept UNKNOWN_OBSTACLE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      THE_SONAR_CONTROL.UNKNOWN_OBSTACLE_P(GOAL_FLAG_2, RETURN_FLAG_2);
      if (GOAL_FLAG_2 = TRUE) then
       G_FLAG_1 := TRUE;
       GOAL_FLAG_2 := FALSE;
      else
       G_FLAG_1 := FALSE;
      end if;
      if (RETURN_FLAG_2 = TRUE) then
       R_FLAG_1 := TRUE;
       RETURN_FLAG_2 := FALSE;
      else
       R_FLAG_1 := FALSE;
      end if;
     end UNKNOWN_OBSTACLE_P;
    or
     accept LOG_NEW_OBSTACLE(G_FLAG_1 : out BOOLEAN) do
      THE_SONAR_CONTROL.LOG_NEW_OBSTACLE(GOAL_FLAG_2);
      if (GOAL_FLAG_2 = TRUE) then
       G_FLAG_1 := TRUE;
       GOAL_FLAG_2 := FALSE;
      else
       G_FLAG_1 := FALSE;
      end if;
     end LOG_NEW_OBSTACLE;
    or
     accept START_LOCAL_REPLANNER(G_FLAG_1 : out BOOLEAN) do
      THE_MISSION_REPLANNER.START_LOCAL_REPLANNER;
      THE_GUIDANCE.LOITER(NAV_X, NAV_Y, NAV_DEPTH, NAV_HEADING, NAV_SPEED,
              NAV_MODE);
       G_FLAG_1 := TRUE;
     end START_LOCAL_REPLANNER;
    or
     accept START_GLOBAL_REPLANNER(G_FLAG_1 : out BOOLEAN) do
      THE_MISSION_REPLANNER.START_GLOBAL_REPLANNER;
      THE_GUIDANCE.LOITER(NAV_X, NAV_Y, NAV_DEPTH, NAV_HEADING, NAV_SPEED,
```

68

```
                NAV_MODE);
        G_FLAG_1 := TRUE;
        end START_GLOBAL_REPLANNER;
      end select;
    end loop;
  end THE_NAVIGATOR;

end NAVIGATOR;
```

```
--Title      : nav_r_s.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Navigator Router task
--------------------------------------------------------------------------------

package NAVIGATOR_ROUTER is

 task THE_NAVIGATOR_ROUTER is

  entry CREATE;
  entry TAKE_GUIDANCE_HEADING(GUIDANCE_HEADING : in FLOAT;
              GUIDANCE_MODE : in INTEGER);
  entry TAKE_LOITER_COMMANDS(GUIDANCE_X : in FLOAT;
              GUIDANCE_Y : in FLOAT;
              GUIDANCE_HEADING : in FLOAT;
              GUIDANCE_SPEED : in FLOAT;
              GUIDANCE_DEPTH : in FLOAT;
              GUIDANCE_MODE : in INTEGER;
              LOITER_GUIDANCE_DONE : out BOOLEAN);
  entry REPLAN_DONE;
 end THE_NAVIGATOR_ROUTER;

end NAVIGATOR_ROUTER;
```

70

```
--Title       : nav_r_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 17 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Navigator Router task
--------------------------------------------------------------------------------------

with TEXT_IO, OOD_ROUTER;
use  TEXT_IO, OOD_ROUTER;

package body NAVIGATOR_ROUTER is


--------------------------------------------------------------------------------------
--Task to handle routing of requests through Navigator
--------------------------------------------------------------------------------------

task body THE_NAVIGATOR_ROUTER is

 NAV_X : FLOAT;
 NAV_Y : FLOAT;
 NAV_DEPTH : FLOAT;
 NAV_SPEED : FLOAT;
 NAV_HEADING : FLOAT;
 NAV_MODE : INTEGER;
 NAV_REPLAN_DONE : BOOLEAN := FALSE; --Flag to signal replan done

begin
 accept CREATE;
 PUT_LINE("Creating NAVIGATOR ROUTER");
 loop
  select
   accept TAKE_GUIDANCE_HEADING(GUIDANCE_HEADING : in FLOAT;
                  GUIDANCE_MODE : in INTEGER) do
    NAV_HEADING := GUIDANCE_HEADING;
    NAV_MODE := GUIDANCE_MODE;
   end TAKE_GUIDANCE_HEADING;
   --In Search mode so take search commands immediately
   THE_OOD_ROUTER.TAKE_GUIDANCE_COMMANDS(NAV_HEADING, NAV_MODE);
  or
   accept TAKE_LOITER_COMMANDS(GUIDANCE_X : in FLOAT;
                  GUIDANCE_Y : in FLOAT;
                  GUIDANCE_HEADING : in FLOAT;
                  GUIDANCE_SPEED : in FLOAT;
                  GUIDANCE_DEPTH : in FLOAT;
                  GUIDANCE_MODE : in INTEGER;
                  LOITER_GUIDANCE_DONE : out BOOLEAN) do
    NAV_X := GUIDANCE_X;
    NAV_Y := GUIDANCE_Y;
    NAV_HEADING := GUIDANCE_HEADING;
    NAV_SPEED := GUIDANCE_SPEED;
```

71

```
            NAV_DEPTH := GUIDANCE_DEPTH;
            NAV_MODE := GUIDANCE_MODE;
            LOITER_GUIDANCE_DONE := NAV_REPLAN_DONE;
          end TAKE_LOITER_COMMANDS;
          THE_OOD_ROUTER.TAKE_NAV_COMMANDS(NAV_X, NAV_Y, NAV_HEADING,
                                      NAV_SPEED, NAV_DEPTH, NAV_MODE);
      or
        accept REPLAN_DONE;
        NAV_REPLAN_DONE := TRUE;
      end select;
    end loop;
  end THE_NAVIGATOR_ROUTER;

end NAVIGATOR_ROUTER;
```

```
--Title       : engin_s.a
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Engineering task
-------------------------------------------------------------------------

package ENGINEERING is

 task THE_ENGINEERING is
  entry CREATE;
  entry POWER_GONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry COMPUTER_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry PROPULSION_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry STEERING_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry DIVING_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry BUOYANCY_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry THRUSTER_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry LEAK_TEST_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
  entry PAYLOAD_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
 end THE_ENGINEERING;

end ENGINEERING;
```

```
--Title      : engin_b.a
--Author     : F.P. Thornton Jr.
--Revised    : 26 Aug 93
--Compiler   : VADS
--System     : Unix
--Description : Body for Engineering task
------------------------------------------------------------------------------

with TEXT_IO, MATH, CALENDAR;
use  TEXT_IO, MATH, CALENDAR;

package body ENGINEERING is

  ------------------------------------------------------------------------------
  --Task to handle engineering functions such as monitoring onboard systems
  ------------------------------------------------------------------------------

  task body THE_ENGINEERING is

   THRUSTER_LEVEL : FLOAT := 100.0;
   THRUSTER_MIN : FLOAT := 80.0;
   THRUSTER_LOSS : FLOAT := 1.0;

  begin
   accept CREATE;
   PUT_LINE("Creating ENGINEERING");
   loop
    select
     accept POWER_GONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      G_FLAG_1 := FALSE;
      R_FLAG_1 := TRUE;
     end POWER_GONE_P;
    or
     accept COMPUTER_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      G_FLAG_1 := FALSE;
      R_FLAG_1 := TRUE;
     end COMPUTER_SYSTEM_PROB_P;
    or
     accept PROPULSION_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      G_FLAG_1 := FALSE;
      R_FLAG_1 := TRUE;
     end PROPULSION_SYSTEM_PROB_P;
    or
     accept STEERING_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      G_FLAG_1 := FALSE;
      R_FLAG_1 := TRUE;
     end STEERING_SYSTEM_PROB_P;
    or
     accept DIVING_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
      G_FLAG_1 := FALSE;
      R_FLAG_1 := TRUE;
```

```
     end DIVING_SYSTEM_PROB_P;
   or
    accept BUOYANCY_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
     G_FLAG_1 := FALSE;
     R_FLAG_1 := TRUE;
    end BUOYANCY_SYSTEM_PROB_P;
   or
    accept THRUSTER_SYSTEM_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
     if THRUSTER_LEVEL > THRUSTER_MIN then
      THRUSTER_LEVEL := THRUSTER_LEVEL - THRUSTER_LOSS;
      G_FLAG_1 := FALSE;
     else
      G_FLAG_1 := TRUE;
     end if;
     R_FLAG_1 := TRUE;
    end THRUSTER_SYSTEM_PROB_P;
   or
    accept LEAK_TEST_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
     G_FLAG_1 := FALSE;
     R_FLAG_1 := TRUE;
    end LEAK_TEST_P;
   or
    accept PAYLOAD_PROB_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
     G_FLAG_1 := FALSE;
     R_FLAG_1 := TRUE;
    end PAYLOAD_PROB_P;
   end select;
  end loop;
 end THE_ENGINEERING;

end ENGINEERING;
```

```
--Title      : weapon_s.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Weapons task
--------------------------------------------------------------------------------

package WEAPONS is

  task THE_WEAPONS is
    entry CREATE;
    entry DROP_PACKAGE(G_FLAG_1 : out BOOLEAN);
  end THE_WEAPONS;

end WEAPONS;
```

```
--Title       : weapon_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Weapons task
--------------------------------------------------------------------------------------

with TEXT_IO;
use  TEXT_IO;

package body WEAPONS is


--------------------------------------------------------------------------------------
--Task to handle functions of weapons officer
--------------------------------------------------------------------------------------

 task body THE_WEAPONS is

 begin
  accept CREATE;
  PUT_LINE("Creating WEAPONS");
  loop
   accept DROP_PACKAGE(G_FLAG_1 : out BOOLEAN) do
    G_FLAG_1 := TRUE;
   end DROP_PACKAGE;
  end loop;
 end THE_WEAPONS;

end WEAPONS;
```

```
--Title      : sender_s.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Command Sender
----------------------------------------------------------------------------------------------

package COMMAND_SENDER is

  procedure SEND(NEW_X : in FLOAT;
          NEW_Y : in FLOAT;
          NEW_HEADING : in FLOAT;
          NEW_SPEED : in FLOAT;
          NEW_DEPTH : in FLOAT;
          NEW_MODE : in INTEGER);

end COMMAND_SENDER;
```

```
--Title       : sender_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Command Sender
-----------------------------------------------------------------------------------

with TEXT_IO, MATH, TRIG_MATH, NETWORK_SW;
use  TEXT_IO, MATH, TRIG_MATH, NETWORK_SW;

package body COMMAND_SENDER is

  package FLOAT_INOUT is new FLOAT_IO(FLOAT);
  package INTEGER_INOUT is new INTEGER_IO(INTEGER);
  use FLOAT_INOUT, INTEGER_INOUT;


  -----------------------------------------------------------------------------------
  --Procedure to send tactical level information to the execution level
  -----------------------------------------------------------------------------------
  procedure SEND(NEW_X : in FLOAT;
            NEW_Y : in FLOAT;
            NEW_HEADING : in FLOAT;
            NEW_SPEED : in FLOAT;
            NEW_DEPTH : in FLOAT;
            NEW_MODE : in INTEGER) is

  begin
   --Write updated command values to execution level
   PUT_FLOAT(RAD_TO_DEG(NEW_HEADING));
   PUT("Commanded Heading is: ");
   PUT(RAD_TO_DEG(NEW_HEADING), FORE=>3,AFT=>2, EXP=>0);
   NEW_LINE;

   PUT_FLOAT(NEW_DEPTH);
   PUT("Commanded Depth is: ");
   PUT(NEW_DEPTH, FORE=>3,AFT=>2, EXP=>0);
   NEW_LINE;

   PUT_FLOAT(NEW_SPEED);
   PUT("Commanded Speed is: ");
   PUT(NEW_SPEED, FORE=>3, AFT=>2, EXP=>0);
   NEW_LINE;

   PUT_FLOAT(NEW_X);
   PUT("Commanded X is: ");
   PUT(NEW_X, FORE=>3, AFT=>2, EXP=>0);
   NEW_LINE;

   PUT_FLOAT(NEW_Y);
```

```
      PUT("Commanded Y is: ");
      PUT(NEW_Y, FORE=>3, AFT=>2, EXP=>0);
      NEW_LINE;

      PUT_MODE(NEW_MODE);
      PUT("Commanded Mode is: ");
      case NEW_MODE is
        when 1 =>
          PUT("Transit");
        when 2 =>
          PUT("Search");
        when 3 =>
          PUT("Task");
        when 4 =>
          PUT("Return");
        when 5 =>
          PUT("Recover");
        when others =>
          PUT("Invalid Mode");
      end case;
      NEW_LINE(2);
    end SEND;

  end COMMAND_SENDER;
```

```
--Title      : guid_s.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System     : Unix
--Description  : Specification for Guidance task
-------------------------------------------------------------------------------------------

package GUIDANCE is

 task THE_GUIDANCE is
  entry CREATE;
  entry GET_GUIDANCE_COMMANDS(NAV_X : in out FLOAT;
                NAV_Y : in out FLOAT;
                NAV_DEPTH : in out FLOAT;
                NAV_HEADING : in out FLOAT;
                NAV_SPEED : in out FLOAT;
                WAYPOINT_X : in out FLOAT;
                WAYPOINT_Y : in out FLOAT;
                WAYPOINT_DEPTH : in out FLOAT);
  entry LOITER(NAV_X : in FLOAT;
        NAV_Y : in FLOAT;
        NAV_DEPTH : in FLOAT;
        NAV_HEADING : in FLOAT;
        NAV_SPEED : in FLOAT;
        NAV_MODE : in INTEGER);
  entry DO_HOMING(G_FLAG_2 : out BOOLEAN);
 end THE_GUIDANCE;

end GUIDANCE;
```

```
--Title        : guid_b.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Body for Guidance task
------------------------------------------------------------------------

with TEXT_IO, SENSORY_RECEIVER, GUIDANCE_ROUTER, NAVIGATOR_ROUTER,
    LOS_CALCULATOR, HOMING_CALCULATOR;
use  TEXT_IO, SENSORY_RECEIVER, GUIDANCE_ROUTER, NAVIGATOR_ROUTER;

package body GUIDANCE is

------------------------------------------------------------------------
--Task to handle guidance functions such as Homing and LOS calculations
------------------------------------------------------------------------

task body THE_GUIDANCE is

  GOAL_FLAG_3 : BOOLEAN := FALSE;        --Flag for lower level objects
  GUIDANCE_X : FLOAT;
  GUIDANCE_Y : FLOAT;
  GUIDANCE_DEPTH : FLOAT;
  GUIDANCE_WAYPOINT_X : FLOAT;
  GUIDANCE_WAYPOINT_Y : FLOAT;
  GUIDANCE_WAYPOINT_DEPTH : FLOAT;
  GUIDANCE_HEADING : FLOAT;
  GUIDANCE_SPEED : FLOAT;
  GUIDANCE_MODE : INTEGER;
  GUIDANCE_BEARING : FLOAT;
  GUIDANCE_RANGE : FLOAT;
  LOITER_GUIDANCE_DONE : BOOLEAN := FALSE; --Flag to signal replanning done

begin
 accept CREATE;
 PUT_LINE("Creating GUIDANCE");
 THE_GUIDANCE_ROUTER.CREATE;
 loop
  select
    accept GET_GUIDANCE_COMMANDS(NAV_X : in out FLOAT;
                 NAV_Y : in out FLOAT;
                 NAV_DEPTH : in out FLOAT;
                 NAV_HEADING : in out FLOAT;
                 NAV_SPEED : in out FLOAT;
                 WAYPOINT_X : in out FLOAT;
                 WAYPOINT_Y : in out FLOAT;
                 WAYPOINT_DEPTH : in out FLOAT) do
     LOS_CALCULATOR.DO_LOS_GUIDANCE(NAV_X, NAV_Y, NAV_DEPTH,
                 WAYPOINT_X, WAYPOINT_Y,
                 WAYPOINT_DEPTH, NAV_SPEED,
```

82

```
                        NAV_HEADING);
      end GET_GUIDANCE_COMMANDS;
    or
     accept DO_HOMING(G_FLAG_2 : out BOOLEAN) do
      HOMING_CALCULATOR.DO_HOMING_GUIDANCE(GOAL_FLAG_3);
      if (GOAL_FLAG_3 = TRUE) then
        G_FLAG_2 := TRUE;
        GOAL_FLAG_3 := FALSE;
      else
        G_FLAG_2 := FALSE;
      end if;
     end DO_HOMING;
    or
     accept LOITER(NAV_X : in FLOAT;
            NAV_Y : in FLOAT;
            NAV_DEPTH : in FLOAT;
            NAV_HEADING : in FLOAT;
            NAV_SPEED : in FLOAT;
            NAV_MODE : in INTEGER) do
      GUIDANCE_WAYPOINT_X := NAV_X;
      GUIDANCE_WAYPOINT_Y := NAV_Y;
      GUIDANCE_WAYPOINT_DEPTH := NAV_DEPTH;
      GUIDANCE_HEADING := NAV_HEADING;
      GUIDANCE_SPEED := NAV_SPEED;
      GUIDANCE_MODE := NAV_MODE;
      loop
       --Simulator protocol delay
       delay 0.5;
       THE_SENSORY_RECEIVER.RECEIVE(GUIDANCE_X, GUIDANCE_Y, GUIDANCE_DEPTH,
                  GUIDANCE_HEADING, GUIDANCE_BEARING,
                  GUIDANCE_RANGE);
       LOS_CALCULATOR.DO_LOS_GUIDANCE(GUIDANCE_X, GUIDANCE_Y,
                  GUIDANCE_DEPTH,
                  GUIDANCE_WAYPOINT_X,
                  GUIDANCE_WAYPOINT_Y,
                  GUIDANCE_WAYPOINT_DEPTH,
                  GUIDANCE_SPEED, GUIDANCE_HEADING);
       THE_NAVIGATOR_ROUTER.TAKE_LOITER_COMMANDS(GUIDANCE_WAYPOINT_X,
                  GUIDANCE_WAYPOINT_Y,
                  GUIDANCE_HEADING,
                  GUIDANCE_SPEED,
                  GUIDANCE_WAYPOINT_DEPTH,
                  GUIDANCE_MODE,
                  LOITER_GUIDANCE_DONE);
      exit when LOITER_GUIDANCE_DONE;
     end loop;
    end LOITER;
   end select;
  end loop;
 end THE_GUIDANCE;
end GUIDANCE;
```

```
--Title        : guid_r_s.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Specification for Guidance Router task
-------------------------------------------------------------------------------

package GUIDANCE_ROUTER is

 task THE_GUIDANCE_ROUTER is
  entry CREATE;
  entry TAKE_HOMING_HEADING(HOMING_HEADING : in FLOAT;
             HOMING_MODE : in INTEGER);
 end THE_GUIDANCE_ROUTER;

end GUIDANCE_ROUTER;
```

```
--Title       : guid_r_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Guidance Router task
-------------------------------------------------------------------------------------


with TEXT_IO, NAVIGATOR_ROUTER;
use  TEXT_IO, NAVIGATOR_ROUTER;

package body GUIDANCE_ROUTER is


-------------------------------------------------------------------------------------
--Task to handle routing of requests through Guidance
-------------------------------------------------------------------------------------


task body THE_GUIDANCE_ROUTER is

  GUIDANCE_HEADING : FLOAT;
  GUIDANCE_MODE : INTEGER;

begin
 accept CREATE;
 PUT_LINE("Creating GUIDANCE ROUTER");
 loop
  accept TAKE_HOMING_HEADING(HOMING_HEADING : in FLOAT;
                HOMING_MODE : in INTEGER) do
   GUIDANCE_HEADING := HOMING_HEADING;
   GUIDANCE_MODE := HOMING_MODE;
  end TAKE_HOMING_HEADING;
  THE_NAVIGATOR_ROUTER.TAKE_GUIDANCE_HEADING(GUIDANCE_HEADING,
                GUIDANCE_MODE);
 end loop;
 end THE_GUIDANCE_ROUTER;

end GUIDANCE_ROUTER;
```

```
--Title       : gps_s.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for GPS Control
------------------------------------------------------------------------------------------------

package GPS_CONTROL is

 task THE_GPS_CONTROL is
  entry CREATE;
  entry GET_GPS_FIX(G_FLAG_2 : out BOOLEAN);
 end THE_GPS_CONTROL;

end GPS_CONTROL;
```

```
--Title        : gps_b.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Body for GPS Control
--------------------------------------------------------------------------------------------

with TEXT_IO;
use  TEXT_IO;

package body GPS_CONTROL is

 task body THE_GPS_CONTROL is

 begin
   accept CREATE;
   PUT_LINE("Creating GPS CONTROL");
   loop
    accept GET_GPS_FIX(G_FLAG_2 : out BOOLEAN) do
      G_FLAG_2 := TRUE;
    end GET_GPS_FIX;
   end loop;
 end THE_GPS_CONTROL;


end GPS_CONTROL;
```

87

```
--Title       : sonar_s.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Sonar Control task
-------------------------------------------------------------------------------------

package SONAR_CONTROL is

 task THE_SONAR_CONTROL is
  entry CREATE;
  entry DO_SEARCH_PATTERN(G_FLAG_2 : out BOOLEAN;
              NAV_HEADING : in FLOAT);
   entry UNKNOWN_OBSTACLE_P(G_FLAG_2, R_FLAG_2 : out BOOLEAN);
   entry LOG_NEW_OBSTACLE(G_FLAG_2 : out BOOLEAN);
 end THE_SONAR_CONTROL;

end SONAR_CONTROL;
```

```
--Title        : sonar_b.a
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Body for Sonar task
----------------------------------------------------------------------


with TEXT_IO, MATH, CALENDAR, NAVIGATOR_ROUTER, MISSION_MODEL,
SENSORY_RECEIVER;
use  TEXT_IO, MATH, CALENDAR, NAVIGATOR_ROUTER, MISSION_MODEL,
SENSORY_RECEIVER;

package body SONAR_CONTROL is


----------------------------------------------------------------------
--Task to handle Sonar Control functions including search, checking for
--obstacles, and logging new obstacle position
----------------------------------------------------------------------


task body THE_SONAR_CONTROL is

  SECONDS : constant DURATION := 1.0;
  LEG_TIME : DURATION := 15 * SECONDS;--15 sec legs (+ 15 sec in turns)
  TURN_TIME : constant DURATION := 15.0;
  INTERVAL : constant DURATION := 15 * SECONDS;--Amount to increase box
  NEXT_TIME : TIME;
  LEG_NUM : INTEGER := 0;
  RANGE_LIMIT : constant FLOAT := 300.0;  --Limits for sonar in Search mode
  BEARING_LIMIT : constant FLOAT := PI / 3.0;
  SONAR_X : FLOAT;
  SONAR_Y : FLOAT;
  SONAR_DEPTH : FLOAT;
  SONAR_HEADING : FLOAT;
  SONAR_BEARING : FLOAT;
  SONAR_RANGE : FLOAT;
  SONAR_MODE : INTEGER := 2;
  SEARCH_HEADING : FLOAT;

begin
 accept CREATE;
 PUT_LINE("Creating SONAR CONTROL");
 loop
   select
     --Do expanding box search pattern
     accept DO_SEARCH_PATTERN(G_FLAG_2 : out BOOLEAN;
                 NAV_HEADING : in FLOAT) do
     SEARCH_HEADING := NAV_HEADING;
     NEXT_TIME := CLOCK + INTERVAL - TURN_TIME;
     loop
       if CLOCK > NEXT_TIME then --Change heading for new leg of search
```

```
if LEG_NUM = 2 then --Expand the box
  LEG_TIME := LEG_TIME + INTERVAL;
  LEG_NUM := 1;
end if;
--Change heading to make box corner and normalize
if (SEARCH_HEADING > (PI / 2.0)) then --Commanded heading > 0
  SEARCH_HEADING := SEARCH_HEADING - (PI / 2.0);
else --Commanded heading <= 0
  SEARCH_HEADING := SEARCH_HEADING + ((3.0 * PI) / 2.0);
end if;
LEG_NUM := LEG_NUM + 1;
NEXT_TIME := NEXT_TIME + LEG_TIME;
end if;
--Simulator protocol delay
delay 0.5;
THE_SENSORY_RECEIVER.RECEIVE(SONAR_X, SONAR_Y, SONAR_DEPTH,
              SONAR_HEADING, SONAR_BEARING,
              SONAR_RANGE);
--Send commanded heading to Navigator
THE_NAVIGATOR_ROUTER.TAKE_GUIDANCE_HEADING(SEARCH_HEADING,
                SONAR_MODE);
--Check for valid range and bearing from sonar to end search
exit when (SONAR_RANGE < RANGE_LIMIT and
      ABS(SONAR_BEARING) < BEARING_LIMIT);
end loop;
--Transition to Task mode
SONAR_MODE := 3;
THE_MISSION_MODEL.SET_MODE(SONAR_MODE);
G_FLAG_2 := TRUE;
end DO_SEARCH_PATTERN;
or
accept UNKNOWN_OBSTACLE_P(G_FLAG_2, R_FLAG_2 : out BOOLEAN) do
  G_FLAG_2 := FALSE;
  R_FLAG_2 := TRUE;
end UNKNOWN_OBSTACLE_P;
or
accept LOG_NEW_OBSTACLE(G_FLAG_2 : out BOOLEAN) do
  G_FLAG_2 := TRUE;
end LOG_NEW_OBSTACLE;
end select;
end loop;
end THE_SONAR_CONTROL;
```

```
--Title       : replan_s.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Mission Replanner task
-----------------------------------------------------------------------------------

package MISSION_REPLANNER is

  task THE_MISSION_REPLANNER is
    entry CREATE;
    entry START_LOCAL_REPLANNER;
    entry START_GLOBAL_REPLANNER;
  end THE_MISSION_REPLANNER;

end MISSION_REPLANNER;
```

```
--Title       : replan_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Mission Replanner task
-----------------------------------------------------------------------------------------

with TEXT_IO, MISSION_MODEL, NAVIGATOR_ROUTER;
use  TEXT_IO, MISSION_MODEL, NAVIGATOR_ROUTER;

package body MISSION_REPLANNER is

-----------------------------------------------------------------------------------------
--Task to handle local and global replanning due to obstacles and system
--faults
-----------------------------------------------------------------------------------------

task body THE_MISSION_REPLANNER is

begin
  accept CREATE;
  PUT_LINE("Creating MISSION REPLANNER");
  loop
    select
      accept START_LOCAL_REPLANNER;
      --Delay to simulate replan time
      delay 30.0;
      THE_MISSION_MODEL.SET_REPLAN_ROUTE;
      THE_NAVIGATOR_ROUTER.REPLAN_DONE;
    or
      accept START_GLOBAL_REPLANNER;
      --Delay to simulate replan time
      delay 30.0;
      THE_MISSION_MODEL.SET_REPLAN_ROUTE;
      THE_NAVIGATOR_ROUTER.REPLAN_DONE;
    end select;
  end loop;
end THE_MISSION_REPLANNER;

end MISSION_REPLANNER;
```

```
--Title        : los_s.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description  : Specification for LOS Calculator
-----------------------------------------------------------------------------------------

package LOS_CALCULATOR is

  procedure DO_LOS_GUIDANCE(FROM_X : in FLOAT;
                FROM_Y : in FLOAT;
                LOS_DEPTH : in out FLOAT;
                TO_X : in FLOAT;
                TO_Y : in FLOAT;
                TO_DEPTH : in FLOAT;
                LOS_SPEED : in FLOAT;
                LOS_HEADING : in out FLOAT);

end LOS_CALCULATOR;
```

```
--Title      : los_b.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler   : VADS
--System      : Unix
--Description : Body for LOS Calculator
----------------------------------------------------------------------------------------


with MATH, TRIG_MATH;
use  MATH, TRIG_MATH;

package body LOS_CALCULATOR is


----------------------------------------------------------------------------------------
--Procedure to calculate updated  heading to next waypoint
----------------------------------------------------------------------------------------


procedure DO_LOS_GUIDANCE(FROM_X : in FLOAT;
               FROM_Y : in FLOAT;
               LOS_DEPTH : in out FLOAT;
               TO_X : in FLOAT;
               TO_Y : in FLOAT;
               TO_DEPTH : in FLOAT;
               LOS_SPEED : in FLOAT;
               LOS_HEADING : in out FLOAT) is
  TIME_OF_ARRIVAL : FLOAT;
  DELTA_TIME : FLOAT := 10.0;
begin
  --Calculate updated heading to waypoint and normalize to 360 degrees
  LOS_HEADING := ATAN2((TO_X - FROM_X),(TO_Y - FROM_Y));
  if LOS_HEADING < 0.0 then
    LOS_HEADING := LOS_HEADING + 2.0 * PI;
  end if;
  --Calculate updated depth
  TIME_OF_ARRIVAL := SQRT((TO_X - FROM_X)**2 + (TO_Y - FROM_Y)**2) /
           (LOS_SPEED / 60.0);
  LOS_DEPTH := LOS_DEPTH + ((TO_DEPTH - LOS_DEPTH) *
        (DELTA_TIME / TIME_OF_ARRIVAL));
end DO_LOS_GUIDANCE;

end LOS_CALCULATOR;
```

```
--Title        : homing_s.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Specification for Homing Calculator
-----------------------------------------------------------------------------

package HOMING_CALCULATOR is

 procedure DO_HOMING_GUIDANCE(G_FLAG_3 : out BOOLEAN);

end HOMING_CALCULATOR;
```

```
--Title       : homing_b.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Homing Calculator
----------------------------------------------------------------------------------


with TEXT_IO, MATH, SENSORY_RECEIVER, GUIDANCE_ROUTER;
use  TEXT_IO, MATH, SENSORY_RECEIVER, GUIDANCE_ROUTER;

package body HOMING_CALCULATOR is


----------------------------------------------------------------------------------
--Procedure to calculate heading for homing
----------------------------------------------------------------------------------


 procedure DO_HOMING_GUIDANCE(G_FLAG_3 : out BOOLEAN) is
  HOMING_X : FLOAT;
  HOMING_Y : FLOAT;
  HOMING_DEPTH : FLOAT;
  HOMING_HEADING : FLOAT;
  HOMING_BEARING : FLOAT;
  HOMING_RANGE : FLOAT;
  HOMING_MODE : INTEGER := 3;        --Initialize to task mode
  EPSILON : constant FLOAT := 20.0;  --Tolerance for reaching target
 begin
  loop
   --Simulator protocol delay
   delay 0.5;
   THE_SENSORY_RECEIVER.RECEIVE(HOMING_X, HOMING_Y, HOMING_DEPTH,
                   HOMING_HEADING, HOMING_BEARING, HOMING_RANGE);
   --Calculate updated heading to target
   HOMING_HEADING := HOMING_HEADING + HOMING_BEARING;
   --Normalize heading to 360 degrees
   if HOMING_HEADING < 0.0 then
    HOMING_HEADING := HOMING_HEADING + (2.0 * PI);
   elsif HOMING_HEADING >= (2.0 * PI) then
    HOMING_HEADING := HOMING_HEADING - (2.0 * PI);
   else
    null;
   end if;
   --Send guidance commands to Guidance
   THE_GUIDANCE_ROUTER.TAKE_HOMING_HEADING(HOMING_HEADING,
                                            HOMING_MODE);
   exit when HOMING_RANGE < EPSILON;
  end loop;
  G_FLAG_3 := TRUE;
 end DO_HOMING_GUIDANCE ;
end HOMING_CALCULATOR;
```

```
package MISSION_MODEL is

 task THE_MISSION_MODEL is
  entry CREATE;
  entry INITIALIZE(G_FLAG_1 : out BOOLEAN);
  entry GIVE_FIRST_WAYPOINT(INITIAL_X : out FLOAT;
                 INITIAL_Y : out FLOAT;
                 INITIAL_DEPTH : out FLOAT;
                 INITIAL_MODE : out INTEGER;
                 INITIAL_HEADING : out FLOAT;
                 INITIAL_SPEED : out FLOAT;
                 FIRST_WAYPOINT_X : out FLOAT;
                 FIRST_WAYPOINT_Y : out FLOAT;
                 FIRST_WAYPOINT_DEPTH : out FLOAT);
   entry IN_TRANSIT_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry TRANSIT_DONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry IN_SEARCH_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry SEARCH_DONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry IN_TASK_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry TASK_DONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry IN_RETURN_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry RETURN_DONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN);
   entry GIVE_NEXT_WAYPOINT(NEXT_X : out FLOAT;
                 NEXT_Y : out FLOAT;
                 NEXT_DEPTH : out FLOAT;
                 NEXT_SPEED : out FLOAT;
                 NEXT_MODE : out INTEGER);
   entry SET_REPLAN_ROUTE;
   entry SET_MODE(MISSION_MODE : in INTEGER);
   entry GET_MODE(MISSION_MODE : out INTEGER);
  end THE_MISSION_MODEL;

end MISSION_MODEL;
```

```
--Title        : miss_b.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 28 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Body for Mission Model task
-----------------------------------------------------------------------------------


with TEXT_IO, NETWORK_SW;
use  TEXT_IO, NETWORK_SW;

package body MISSION_MODEL is

 package FLOAT_INOUT is new FLOAT_IO(FLOAT);
 package INTEGER_INOUT is new INTEGER_IO(INTEGER);
 use FLOAT_INOUT, INTEGER_INOUT;


 -----------------------------------------------------------------------------------
 --Task to manage mission database
 -----------------------------------------------------------------------------------


 task body THE_MISSION_MODEL is

  INITIAL_STATE_FILE : FILE_TYPE;
  WAYPOINT_FILE : FILE_TYPE;
  FINAL_GOAL_FILE : FILE_TYPE;
  --Data structure to hold waypoints
  type WAYPOINT is
   record
    X : FLOAT;
    Y : FLOAT;
    DEPTH : FLOAT;
    HEADING : FLOAT;
    MODE : INTEGER;
    SPEED : FLOAT;
   end record;
  INITIAL : WAYPOINT;
  FINAL : WAYPOINT;
  MAX_WAYPOINTS : INTEGER := 25;
  type WAYPOINTS is array (INTEGER range 1..MAX_WAYPOINTS) of WAYPOINT;
  WAYPOINT_LIST : WAYPOINTS;
  WAYPOINT_COUNT : INTEGER;
  I : INTEGER := 1;         --Counter for total number of waypoints
  K : INTEGER := 1;         --Counter for current waypoint
  CURRENT_MODE : INTEGER := 1; --Initialize mode to Transit

 begin
  accept CREATE;
  PUT_LINE("Creating MISSION MODEL");
  loop
   select
```

98

```
--Initialize Mission Model with initial state, waypoints, final goal
accept INITIALIZE(G_FLAG_1 : out BOOLEAN) do
 begin
   --Load initial state from file
   OPEN(INITIAL_STATE_FILE, MODE => IN_FILE, NAME => "initial_state");
   GET(INITIAL_STATE_FILE, INITIAL.X);
   GET(INITIAL_STATE_FILE, INITIAL.Y);
   GET(INITIAL_STATE_FILE, INITIAL.DEPTH);
   GET(INITIAL_STATE_FILE, INITIAL.HEADING);
   PUT_FLOAT(INITIAL.X);
   PUT_FLOAT(INITIAL.Y);
   PUT_FLOAT(INITIAL.DEPTH);
   PUT_FLOAT(INITIAL.HEADING);
   CLOSE(INITIAL_STATE_FILE);

   --Load waypoints from file
   OPEN(WAYPOINT_FILE, MODE => IN_FILE, NAME => "waypoints");
   GET(WAYPOINT_FILE, WAYPOINT_COUNT);
   SKIP_LINE(WAYPOINT_FILE);
   PUT_FLOAT(FLOAT(WAYPOINT_COUNT));
   while not END_OF_FILE(WAYPOINT_FILE) loop
    GET(WAYPOINT_FILE, WAYPOINT_LIST(I).SPEED);
    GET(WAYPOINT_FILE, WAYPOINT_LIST(I).X);
    GET(WAYPOINT_FILE, WAYPOINT_LIST(I).Y);
    GET(WAYPOINT_FILE, WAYPOINT_LIST(I).DEPTH);
    GET(WAYPOINT_FILE, WAYPOINT_LIST(I).MODE);
    SKIP_LINE(WAYPOINT_FILE);
    PUT_FLOAT(WAYPOINT_LIST(I).SPEED);
    PUT_FLOAT(WAYPOINT_LIST(I).X);
    PUT_FLOAT(WAYPOINT_LIST(I).Y);
    PUT_FLOAT(WAYPOINT_LIST(I).DEPTH);
    I := I + 1;
   end loop;
   CLOSE(WAYPOINT_FILE);

   --Load final goal from file
   OPEN(FINAL_GOAL_FILE, MODE => IN_FILE, NAME => "final_goal");
   GET(FINAL_GOAL_FILE, FINAL.X);
   GET(FINAL_GOAL_FILE, FINAL.Y);
   PUT_FLOAT(FINAL.X);
   PUT_FLOAT(FINAL.Y);
   CLOSE(FINAL_GOAL_FILE);

   G_FLAG_1 := TRUE;
 exception
   when others =>
    PUT_LINE("Error in mission files");
    G_FLAG_1 := FALSE;
  end;
 end INITIALIZE;
or
```

99

```
--Select initial state and first waypoint values
accept GIVE_FIRST_WAYPOINT(INITIAL_X : out FLOAT;
                            INITIAL_Y : out FLOAT;
                            INITIAL_DEPTH : out FLOAT;
                            INITIAL_MODE : out INTEGER;
                            INITIAL_HEADING : out FLOAT;
                            INITIAL_SPEED : out FLOAT;
                            FIRST_WAYPOINT_X : out FLOAT;
                            FIRST_WAYPOINT_Y : out FLOAT;
                            FIRST_WAYPOINT_DEPTH : out FLOAT) do

   INITIAL_X := INITIAL.X;
   INITIAL_Y := INITIAL.Y;
   INITIAL_DEPTH := INITIAL.DEPTH;
   INITIAL_HEADING := INITIAL.HEADING;
   INITIAL_MODE := CURRENT_MODE;
   INITIAL_SPEED := WAYPOINT_LIST(1).SPEED;
   FIRST_WAYPOINT_X := WAYPOINT_LIST(1).X;
   FIRST_WAYPOINT_Y := WAYPOINT_LIST(1).Y;
   FIRST_WAYPOINT_DEPTH := WAYPOINT_LIST(1).DEPTH;
 end GIVE_FIRST_WAYPOINT;
or
 --Entries to determine mission mode
 --Integer values equate to modes:
 -- 1 = Transit, 2 = Search, 3 = Task, 4 = Return, 5 = Recover

 accept IN_TRANSIT_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE = 1 then
    G_FLAG_1 := TRUE;
   else
    G_FLAG_1 := FALSE;
   end if;
   R_FLAG_1 := TRUE;
 end IN_TRANSIT_P;
or
 accept TRANSIT_DONE_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE > 1 then
    G_FLAG_1 := TRUE;
   else
    G_FLAG_1 := FALSE;
   end if;
   R_FLAG_1 := TRUE;
 end TRANSIT_DONE_P;
or
 accept IN_SEARCH_P(G_FLAG_1, R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE = 2 then
    G_FLAG_1 := TRUE;
   else
    G_FLAG_1 := FALSE;
   end if;
   R_FLAG_1 := TRUE;
```

```
    end IN_SEARCH_P;
or
 accept SEARCH_DONE_P(G_FLAG_1. R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE > 2 then
   G_FLAG_1 := TRUE;
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end SEARCH_DONE_P;
or
 accept IN_TASK_P(G_FLAG_1. R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE = 3 then
   G_FLAG_1 := TRUE;
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end IN_TASK_P;
or
 accept TASK_DONE_P(G_FLAG_1. R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE > 3 then
   G_FLAG_1 := TRUE;
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end TASK_DONE_P;
or
 accept IN_RETURN_P(G_FLAG_1. R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE = 4 then
   G_FLAG_1 := TRUE;
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end IN_RETURN_P;
or
 accept RETURN_DONE_P(G_FLAG_1. R_FLAG_1 : out BOOLEAN) do
  if CURRENT_MODE > 4 then
   PUT_LINE("**********Goal Reached**********");
   G_FLAG_1 := TRUE;
  else
   G_FLAG_1 := FALSE;
  end if;
  R_FLAG_1 := TRUE;
 end RETURN_DONE_P;
or
 --Retrieve next waypoint for Navigator
 accept GIVE_NEXT_WAYPOINT(NEXT_X : out FLOAT;
                           NEXT_Y : out FLOAT;
```

```
                    NEXT_DEPTH : out FLOAT;
                    NEXT_SPEED : out FLOAT;
                    NEXT_MODE : out INTEGER) do
         NEXT_MODE := WAYPOINT_LIST(K).MODE;
         NEXT_SPEED := WAYPOINT_LIST(K).SPEED;
         if (CURRENT_MODE = 1) or (CURRENT_MODE = 2) or
            (CURRENT_MODE = 4) then --Normal case:use next waypoint X,Y,DEPTH
            NEXT_X := WAYPOINT_LIST(K + 1).X;
            NEXT_Y := WAYPOINT_LIST(K + 1).Y;
            NEXT_DEPTH := WAYPOINT_LIST(K + 1).DEPTH;
            CURRENT_MODE := WAYPOINT_LIST(K).MODE;
            K := K + 1;
         else --Odd case:use current waypoint X,Y,DEPTH
            NEXT_X := WAYPOINT_LIST(K).X;
            NEXT_Y := WAYPOINT_LIST(K).Y;
            NEXT_DEPTH := WAYPOINT_LIST(K).DEPTH;
            CURRENT_MODE := WAYPOINT_LIST(K).MODE;
         end if;
        end GIVE_NEXT_WAYPOINT;
       or
        --Change waypoint, mode, and speed for replan route
        accept SET_REPLAN_ROUTE do
          K := 1 - 3;
          WAYPOINT_LIST(K).MODE := 4;
          WAYPOINT_LIST(K).SPEED := WAYPOINT_LIST(K + 1).SPEED;
        end SET_REPLAN_ROUTE;
       or
        accept SET_MODE(MISSION_MODE : in INTEGER) do
          CURRENT_MODE := MISSION_MODE;
        end SET_MODE;
       or
        accept GET_MODE(MISSION_MODE : out INTEGER) do
          MISSION_MODE := CURRENT_MODE;
        end GET_MODE;
       end select;
      end loop;
    end THE_MISSION_MODEL;

end MISSION_MODEL;
```

```
--Title      : world_s.a (CLIPS-Ada Simulator Version)
--Author     : F.P. Thornton Jr.
--Revised    : 26 Aug 93
--Compiler   : VADS
--System     : Unix
--Description : Specification for World Model task
-------------------------------------------------------------------------------------

package WORLD_MODEL is

 task THE_WORLD_MODEL is
   entry CREATE;
   entry INITIALIZE(G_FLAG_1 : out BOOLEAN);
   entry GET_SONAR_CONTACT(SONAR_X : out FLOAT;
               SONAR_Y : out FLOAT);
   entry ADD_OBSTACLE(OBSTACLE_X : in FLOAT;
            OBSTACLE_Y : in FLOAT;
            OBSTACLE_DEPTH : in FLOAT);
 end THE_WORLD_MODEL;

end WORLD_MODEL;
```

```
--Title        : world_b.a (CLIPS-Ada Simulator Version)
--Author       : F.P. Thornton Jr.
--Revised      : 26 Aug 93
--Compiler     : VADS
--System       : Unix
--Description   : Body for World Model task
------------------------------------------------------------------------------------------


with TEXT_IO, NETWORK_SW;
use  TEXT_IO, NETWORK_SW;

package body WORLD_MODEL is
 package FLOAT_INOUT is new FLOAT_IO(FLOAT);
 package INTEGER_INOUT is new INTEGER_IO(INTEGER);
 use FLOAT_INOUT, INTEGER_INOUT;


------------------------------------------------------------------------------------------
--Task to manage world database, which includes obstacles
------------------------------------------------------------------------------------------


task body THE_WORLD_MODEL is

 OBSTACLE_FILE : FILE_TYPE;
 --Data structure to hold obstacles
 type OBSTACLE is
  record
    X : FLOAT;
    Y : FLOAT;
    DEPTH : FLOAT;
  end record;
 CURRENT_OBSTACLE : OBSTACLE;
 NEXT_OBSTACLE : OBSTACLE;
 MAX_OBSTACLES : INTEGER := 25;
 type OBSTACLES is array (INTEGER range 1..MAX_OBSTACLES) of OBSTACLE;
 OBSTACLE_LIST : OBSTACLES;
 OBSTACLE_COUNT : INTEGER;
 J : INTEGER := 1; --Counter for total number of obstacles

begin
 accept CREATE;
 PUT_LINE("Creating WORLD MODEL");
 loop
  select
   --Initialize World Model by loading obstacles
   accept INITIALIZE(G_FLAG_1 : out BOOLEAN) do
    begin
     OPEN(OBSTACLE_FILE, MODE => IN_FILE, NAME => "obstacles");
     GET(OBSTACLE_FILE, OBSTACLE_COUNT);
     SKIP_LINE(OBSTACLE_FILE);
     PUT_FLOAT(FLOAT(OBSTACLE_COUNT));
     while not END_OF_FILE(OBSTACLE_FILE) loop
```

104

```ada
        GET(OBSTACLE_FILE, OBSTACLE_LIST(J).X);
        GET(OBSTACLE_FILE, OBSTACLE_LIST(J).Y);
        GET(OBSTACLE_FILE, OBSTACLE_LIST(J).DEPTH);
        SKIP_LINE(OBSTACLE_FILE);
        PUT_FLOAT(OBSTACLE_LIST(J).X);
        PUT_FLOAT(OBSTACLE_LIST(J).Y);
        PUT_FLOAT(OBSTACLE_LIST(J).DEPTH);
        J := J + 1;
      end loop;
      CLOSE(OBSTACLE_FILE);

      NEXT_OBSTACLE := OBSTACLE_LIST(J);
      G_FLAG_1 := TRUE;
    exception
      when others =>
        PUT_LINE("Error in world files");
        G_FLAG_1 := FALSE;
    end;
    end INITIALIZE;
  or
    --Get an obstacle for sonar target
    accept GET_SONAR_CONTACT(SONAR_X : out FLOAT;
                  SONAR_Y : out FLOAT) do
      SONAR_X := OBSTACLE_LIST(1).X;
      SONAR_Y := OBSTACLE_LIST(1).Y;
    end GET_SONAR_CONTACT;
  or
    --Add a new obstacle
    accept ADD_OBSTACLE(OBSTACLE_X : in FLOAT;
                  OBSTACLE_Y : in FLOAT;
                  OBSTACLE_DEPTH : in FLOAT) do
      NEXT_OBSTACLE.X := OBSTACLE_X;
      NEXT_OBSTACLE.Y := OBSTACLE_Y;
      NEXT_OBSTACLE.DEPTH := OBSTACLE_DEPTH;
      NEXT_OBSTACLE := OBSTACLE_LIST(J);
      J := J + 1;
    end ADD_OBSTACLE;
  end select;
  end loop;
 end THE_WORLD_MODEL;

end WORLD_MODEL;
```

```
--Title       : receiv_s.a (CLIPS-Ada Simulator Version)
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Specification for Sensory Receiver task
-----------------------------------------------------------------------------------

package SENSORY_RECEIVER is

 task THE_SENSORY_RECEIVER is
  entry CREATE;
  entry RECEIVE(CURRENT_X : in out FLOAT;
          CURRENT_Y : in out FLOAT;
          CURRENT_DEPTH : in out FLOAT;
          CURRENT_HEADING : in out FLOAT;
          CURRENT_BEARING : in out FLOAT;
          CURRENT_RANGE : in out FLOAT);
 end THE_SENSORY_RECEIVER;

end SENSORY_RECEIVER;
```

```
--Title       : receiv_b.a
--Author      : F.P. Thornton Jr.
--Revised     : 26 Aug 93
--Compiler    : VADS
--System      : Unix
--Description  : Body for Sensory Receiver task
----------------------------------------------------------------------------------

with TEXT_IO, MATH, TRIG_MATH, NETWORK_SW, WORLD_MODEL;
use  TEXT_IO, MATH, TRIG_MATH, NETWORK_SW, WORLD_MODEL;

package body SENSORY_RECEIVER is

 package FLOAT_INOUT is new FLOAT_IO(FLOAT);
 package INTEGER_INOUT is new INTEGER_IO(INTEGER);
 use FLOAT_INOUT, INTEGER_INOUT;


 ----------------------------------------------------------------------------------
 --Task to get navigation status values from execution level and provide
 --them to the tactical level
 ----------------------------------------------------------------------------------

 task body THE_SENSORY_RECEIVER is

  RECEIVED : BOOLEAN := FALSE;
  CURRENT_ALT : FLOAT;
  CURRENT_SPEED : FLOAT;
  SONAR_X : FLOAT;
  SONAR_Y : FLOAT;

 begin
  accept CREATE;
  PUT_LINE("Creating SENSORY RECEIVER");
  loop
   accept RECEIVE(CURRENT_X : in out FLOAT;
            CURRENT_Y : in out FLOAT;
            CURRENT_DEPTH : in out FLOAT;
            CURRENT_HEADING : in out FLOAT;
            CURRENT_BEARING : in out FLOAT;
            CURRENT_RANGE : in out FLOAT) do
     CURRENT_X := get_float;
     PUT("Current X = ");
     PUT(CURRENT_X, FORE=>3, AFT=>2,EXP=>0);
     NEW_LINE;

     CURRENT_Y := get_float;
     PUT("Current Y = ");
     PUT(CURRENT_Y, FORE=>3, AFT=>2,EXP=>0);
     NEW_LINE;

     CURRENT_ALT := get_float;
```

```
        CURRENT_DEPTH := get_float;
        PUT("Current Depth = ");
        PUT(CURRENT_DEPTH, FORE=>3, AFT=>2,EXP=>0);
        NEW_LINE;

        CURRENT_HEADING := DEG_TO_RAD(get_float);
        PUT("Current Heading = ");
        PUT(RAD_TO_DEG(CURRENT_HEADING), FORE=>3, AFT=>2,EXP=>0);
        NEW_LINE;

        --Speed does not come from the simulator
        CURRENT_SPEED := 0.0;

        --Calculate bearing and range to simulated sonar contact
        if not RECEIVED then
          THE_WORLD_MODEL.GET_SONAR_CONTACT(SONAR_X, SONAR_Y);
          RECEIVED := TRUE;
        end if;
        CURRENT_BEARING := CURRENT_HEADING -
                ATAN2((SONAR_X - CURRENT_X),(SONAR_Y - CURRENT_Y));
        --Normalize to 360 degrees but keep negative values for bearing
        if CURRENT_BEARING < 0.0 then
          CURRENT_BEARING := ABS(CURRENT_BEARING);
        elsif CURRENT_BEARING > PI then
          CURRENT_BEARING := (2.0 * PI) - CURRENT_BEARING;
        else --0.0 <= CURRENT_BEARING <= PI
          CURRENT_BEARING := 0.0 - CURRENT_BEARING;
        end if;
        PUT("Current Bearing = ");
        PUT(RAD_TO_DEG(CURRENT_BEARING), FORE=>3, AFT=>2, EXP=>0);
        NEW_LINE;

        CURRENT_RANGE := SQRT((SONAR_X - CURRENT_X)**2 +
                (SONAR_Y - CURRENT_Y)**2);
        PUT("Current Range = ");
        PUT(CURRENT_RANGE, FORE=>3, AFT=>2, EXP=>0);
        NEW_LINE;
      end RECEIVE;
    end loop;
  end THE_SENSORY_RECEIVER;

end SENSORY_RECEIVER;
```

```
--Title       : trig_mth.a
--Author      : R.B. Byrnes
--Revised     : 18 Aug 93 by F.P. Thornton Jr.
--Compiler    : VADS
--System      : Unix
--Description  : Trigonometric and conversion functions
-----------------------------------------------------------------------------------

with MATH;
use  MATH;

package TRIG_MATH is
   LOWER_LIMIT : constant FLOAT := 0.000001;
   function ATAN2(Y,X : FLOAT) return FLOAT;
   function RAD_TO_DEG(X : FLOAT) return FLOAT;
   function DEG_TO_RAD(X : FLOAT) return FLOAT;
end TRIG_MATH;

package body TRIG_MATH is


   ------------------------------------------------------------------------------
   --Trig functions for heading and bearing calculations
   ------------------------------------------------------------------------------


   function SIGNUM (R : FLOAT) return FLOAT is
   begin
    if R < 0.0 then
       return -1.0;
    else
       return +1.0;
    end if;
   end SIGNUM;

   function ATAN2(Y,X : FLOAT) return FLOAT is
   begin
    if ABS(X) > LOWER_LIMIT then
      if X > 0.0 then
        return ATAN(Y/X);
      else
        return ATAN(Y/X) + (SIGNUM(Y) * PI);
      end if;
    else
      return SIGNUM(Y) * (PI/2.0);
    end if;
   end ATAN2;


   ------------------------------------------------------------------------------
   --Conversion functions for angles
   ------------------------------------------------------------------------------


   function RAD_TO_DEG(X : FLOAT) return FLOAT is
```

```
begin
  return X * (180.0 / PI);
end RAD_TO_DEG;

function DEG_TO_RAD(X : FLOAT) return FLOAT is
begin
  return X * (PI / 180.0);
end DEG_TO_RAD;

end TRIG_MATH;
```

```
--Title      : netwk_i.a
--Author     : R.B. Byrnes
--Revised     : 30 Jul 93 by F.P. Thornton Jr.
--Compiler   : VADS
--System      : Unix
--Description : Interface to C communication routines
---------------------------------------------------------------------------------

package NETWORK_SW is
--
-- CLIENT comms, supporting Tactical<->Execution level
--
  procedure start_comms;    -- make connection to E-level
  procedure put_float (X : FLOAT);  -- send float to E-level
  function get_float return FLOAT;  -- receive float from E-level
  procedure put_mode (X : INTEGER);  -- send mode to E-level
  procedure stop_comms;     -- close connection to E-level
--
-- System clock access function. Better than Ada's
--
  procedure get_time;

private

  pragma INTERFACE(C, start_comms);
  pragma INTERFACE(C, put_float);
  pragma INTERFACE(C, get_float);
  pragma INTERFACE(C, stop_comms);
  pragma INTERFACE(C, put_mode);
  pragma INTERFACE(C, get_time);

  pragma LINK_WITH("network_sw.o");  -- lump all above files together

end NETWORK_SW;
```

APPENDIX B. TRACES OF MISSION EXECUTION

1. MULTI-PHASE MISSION

```
CLIPS> (assert (start))
CLIPS> (run)
Creating OOD
Creating MISSION MODEL
Creating WORLD MODEL
Creating SENSORY RECEIVER
Creating OOD ROUTER
Creating NAVIGATOR
Creating ENGINEERING
Creating WEAPONS
Creating NAVIGATOR ROUTER
Creating GUIDANCE
Creating GPS CONTROL
Creating MISSION REPLANNER
Creating SONAR CONTROL
Creating GUIDANCE ROUTER
READY_VEHICLE_FOR_LAUNCH GOAL FLAG =    1
SELECT_FIRST_WAYPOINT GOAL FLAG =    1
IN_TRANSIT_P GOAL FLAG =    1
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
UNKNOWN_OBSTACLE_P GOAL FLAG =    0
Commanded Heading is:  45.00
Commanded Depth is:   5.89
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit
```

```
SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
Current X =    8.81
Current Y =    0.00
Current Depth =   -0.00
Current Heading =   89.00
Current Bearing = -21.92
Current Range = 641.87
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is:   43.97
Commanded Depth is:    6.00
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit

SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
```

```
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
Current X =  17.39
Current Y =  -0.05
Current Depth =  -0.01
Current Heading =  88.00
Current Bearing = -21.23
Current Range = 634.00
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is:  42.93
Commanded Depth is:   6.09
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit

        .


        .



GPS_NEEDED_P GOAL FLAG =    0
Current X = 240.39
Current Y = 234.65
Current Depth =  48.17
Current Heading =  32.00
Current Bearing =  55.56
Current Range = 359.94
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is:  32.04
Commanded Depth is:  52.38
Commanded Speed is: 250.00
```

```
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit

SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
o crit-system-prob branch successful!

GPS_NEEDED_P GOAL FLAG =    0
*****At waypoint, coming to new heading*****
Current X = 245.06
Current Y = 241.97
Current Depth =  49.04
Current Heading =  32.00
Current Bearing =  56.70
Current Range = 355.04
REACH_WAYPOINT_P GOAL FLAG =    1
GET_NEXT_WAYPOINT GOAL FLAG =    1
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is:  31.61
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Search

SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
IN_SEARCH_P GOAL FLAG =    1
Current X = 249.73
Current Y = 249.32
```

```
Current Depth =  49.82
Current Heading =  32.00
Current Bearing =  57.89
Current Range = 350.27
Commanded Heading is:  31.61
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Search


Current X = 254.40
Current Y = 256.71
Current Depth =  50.51
Current Heading =  32.00
Current Bearing =  59.11
Current Range = 345.66
Commanded Heading is:  31.61
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Search


Current X = 259.08
Current Y = 264.11
Current Depth =  51.21
Current Heading =  32.00
Current Bearing =  60.37
Current Range = 341.22
Commanded Heading is:  31.61
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Search
        .


        .


        .


Current X = 301.47
Current Y = 222.07
Current Depth =  54.45
```

```
Current Heading = 123.00
Current Bearing = -38.35
Current Range = 299.84
Commanded Heading is: 121.61
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Search

DO_SEARCH_PATTERN GOAL FLAG =    1
SEARCH_DONE_P GOAL FLAG =    1
***********************************
*          SEARCH SUCCESSFUL.         *
***********************************
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    1
Current X = 308.74
Current Y = 217.49
Current Depth =  54.45
Current Heading = 123.00
Current Bearing = -39.37
Current Range = 293.07
Commanded Heading is:  83.63
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Task

Current X = 316.03
Current Y = 212.91
Current Depth =  54.45
Current Heading = 123.00
Current Bearing = -40.44
Current Range = 286.38
Commanded Heading is:  82.56
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Task
```

```
Current X = 323.31
Current Y = 208.28
Current Depth =  54.45
Current Heading = 121.00
Current Bearing = -39.58
Current Range = 279.82
Commanded Heading is:  81.42
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Task

        .


        .


        .


Current X = 576.69
Current Y = 243.88
Current Depth =  56.06
Current Heading =  76.00
Current Bearing =  -0.71
Current Range =  24.10
Commanded Heading is:  75.29
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Task

Current X = 585.25
Current Y = 246.07
Current Depth =  56.06
Current Heading =  76.00
Current Bearing =  -0.93
Current Range =  15.27
Commanded Heading is:  75.07
Commanded Depth is:  53.26
Commanded Speed is: 250.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Task

HOMING GOAL FLAG =    1
```

118

```
DROP_PACKAGE GOAL FLAG =    1
GET_GPS_FIX GOAL FLAG =    1
GET_NEXT_WAYPOINT GOAL FLAG =    1
TASK_DONE_P GOAL FLAG =    1
***********************************
*           TASK SUCCESSFUL.         *
***********************************
IN_TASK_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    1
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    0
Current X = 593.81
Current Y = 248.26
Current Depth =   56.06
Current Heading =   76.00
Current Bearing =  -1.74
Current Range =    6.43
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is: 235.66
Commanded Depth is:   47.08
Commanded Speed is: 360.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Return

SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    0
RETURN_DONE_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    1
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
```

```
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    0
Current X = 602.39
Current Y = 250.45
Current Depth =  56.06
Current Heading =  76.00
Current Bearing = -176.59
Current Range =    2.43
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is: 236.61
Commanded Depth is:  47.49
Commanded Speed is: 360.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Return
        .

        .

        .

GPS_NEEDED_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    0
Current X = 308.61
Current Y =  43.42
Current Depth =  20.17
Current Heading = 220.00
Current Bearing = -165.33
Current Range = 357.18
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
```

```
Commanded Heading is: 212.68
Commanded Depth is:  19.53
Commanded Speed is: 360.00
Commanded X is: 300.00
Commanded Y is:  30.00
Commanded Mode is: Return

SEND_SETPOINTS_AND_MODES GOAL FLAG =   1
IN_SEARCH_P GOAL FLAG =   0
IN_TASK_P GOAL FLAG =   0
RETURN_DONE_P GOAL FLAG =   0
IN_RETURN_P GOAL FLAG =   1
POWER_GONE_P GOAL FLAG =   0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =   0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =   0
STEERING_SYSTEM_PROB_P GOAL FLAG =   0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =   0
IN_TRANSIT_P GOAL FLAG =   0
*****At waypoint, coming to new heading*****
Current X = 300.79
Current Y =  34.16
Current Depth =  19.81
Current Heading = 217.00
Current Bearing = -162.81
Current Range = 368.93
REACH_WAYPOINT_P GOAL FLAG =   1
GET_NEXT_WAYPOINT GOAL FLAG =   1
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =   0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =   0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =   0
LEAK_TEST_P GOAL FLAG =   0
PAYLOAD_PROB_P GOAL FLAG =   0
No red-cap-system-prob branch successful!
Commanded Heading is: 190.82
Commanded Depth is:  22.45
Commanded Speed is: 360.00
Commanded X is:   0.00
Commanded Y is:   0.00
Commanded Mode is: Recover

SEND_SETPOINTS_AND_MODES GOAL FLAG =   1
IN_SEARCH_P GOAL FLAG =   0
```

```
IN_TASK_P GOAL FLAG =    0
**********Goal Reached**********
RETURN_DONE_P GOAL FLAG =    1
IN_RETURN_P GOAL FLAG =    0
WAIT_FOR_RECOVERY GOAL FLAG =    1
******************************Current X = 293.27
Current Y =  24.58
Current Depth =  19.54
Current Heading = 214.00
Current Bearing = -160.31
Current Range = 380.66
*****
*          RETURN SUCCESSFUL.        *
************************************
*******Commanded Heading is: 214.00
Commanded Depth is:    0.00
Commanded Speed is:    0.00
Commanded X is:    0.00
Commanded Y is:    0.00
Commanded Mode is: Recover

*****************************
*   MISSION EXECUTED SUCCESSFULLY.  *
*   AUV IS WAITING FOR RECOVERY...  *
****************************Current X = 286.27
Current Y =  14.60
Current Depth =  19.32
Current Heading = 208.00
Current Bearing = -154.88
Current Range = 392.22
********
```

2. MULTI-PHASE MISSION WITH ROUTE REPLANNING

```
CLIPS> (assert (start))
CLIPS> (run)
Creating OOD
Creating MISSION MODEL
Creating WORLD MODEL
Creating SENSORY RECEIVER
Creating OOD ROUTER
Creating NAVIGATOR
Creating ENGINEERING
Creating WEAPONS
Creating NAVIGATOR ROUTER
Creating GUIDANCE
Creating GPS CONTROL
Creating MISSION REPLANNER
Creating SONAR CONTROL
Creating GUIDANCE ROUTER
READY_VEHICLE_FOR_LAUNCH GOAL FLAG =    1
SELECT_FIRST_WAYPOINT GOAL FLAG =    1
WARNING: Reset Command may not be performed during the
execution of a rule
IN_TRANSIT_P GOAL FLAG =    1
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
GPS_NEEDED_P GOAL FLAG =    0
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P GOAL FLAG =    0
PAYLOAD_PROB_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
UNKNOWN_OBSTACLE_P GOAL FLAG =    0
Commanded Heading is:   45.00
Commanded Depth is:    5.89
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit
```

```
SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
IN_SEARCH_P GOAL FLAG =    0
IN_TASK_P GOAL FLAG =    0
IN_RETURN_P GOAL FLAG =    0
IN_TRANSIT_P GOAL FLAG =    1
TRANSIT_DONE_P GOAL FLAG =    0
POWER_GONE_P GOAL FLAG =    0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =    0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =    0
STEERING_SYSTEM_PROB_P GOAL FLAG =    0
No crit-system-prob branch successful!
Current X =    8.81
Current Y =    0.00
Current Depth =   -0.00
Current Heading =  89.00
Current Bearing = -21.92
Current Range = 641.87
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    0
LEAK_TEST_P COAL FLAG =    0
PAYLOAD_PROP_P GOAL FLAG =    0
No red-cap-system-prob branch successful!
Commanded Heading is:  43.97
Commanded Depth is:    6.00
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit
        .

        .

        .

Current X = 124.75
Current Y =  81.84
Current Depth =  18.64
Current Heading =  38.00
Current Bearing =  32.51
Current Range = 504.12
REACH_WAYPOINT_P GOAL FLAG =    0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
```

```
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    1
Commanded Heading is:  36.68
Commanded Depth is:  24.87
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit

SEND_SETPOINTS_AND_MODES GOAL FLAG =    1
LOITER GOAL FLAG =    1
Current X = 129.81
Current Y =  88.16
Current Depth =  19.87
Current Heading =  38.00
Current Bearing =  33.01
Current Range = 497.26
Commanded Heading is: 218.71
Commanded Depth is:  24.87
Commanded Speed is: 250.00
Commanded X is: 124.75
Commanded Y is:  81.84
Commanded Mode is: Transit

Current X = 134.89
Current Y =  94.49
Current Depth =  21.11
Current Heading =  38.00
Current Bearing =  33.51
Current Range = 490.42
Commanded Heading is: 218.73
Commanded Depth is:  24.87
Commanded Speed is: 250.00
Commanded X is: 124.75
Commanded Y is:  81.84
Commanded Mode is: Transit

Current X = 140.03
Current Y = 100.77
Current Depth =  22.37
Current Heading =  36.00
Current Bearing =  36.02
Current Range = 483.57
```

```
Commanded Heading is: 218.92
Commanded Depth is:  24.87
Commanded Speed is: 250.00
Commanded X is: 124.75
Commanded Y is:  81.84
Commanded Mode is: Transit

    .

    .


    .

Current X = 241.34
Current Y = 237.36
Current Depth =  48.08
Current Heading =  35.00
Current Bearing =  52.98
Current Range = 358.88
REACH_WAYPOINT_P GOAL FLAG =   0
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =   0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =   0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =   1
Commanded Heading is:  34.40
Commanded Depth is:  53.30
Commanded Speed is: 250.00
Commanded X is: 250.00
Commanded Y is: 250.00
Commanded Mode is: Transit

SEND_SETPOINTS_AND_MODES GOAL FLAG =   1
IN_SEARCH_P GOAL FLAG =   0
IN_TASK_P GOAL FLAG =   0
IN_RETURN_P GOAL FLAG =   0
IN_TRANSIT_P GOAL FLAG =   1
TRANSIT_DONE_P GOAL FLAG =   0
POWER_GONE_P GOAL FLAG =   0
COMPUTER_SYSTEM_PROB_P GOAL FLAG =   0
PROPULSION_SYSTEM_PROB_P GOAL FLAG =   0
STEERING_SYSTEM_PROB_P GOAL FLAG =   0
No crit-system-prob branch successful!
*****At waypoint, coming to new heading*****
Current X = 245.89
REACH_WAYPOINT_P GOAL FLAG =   1
Current Y = 244.09
Current Depth =  48.99
```

```
Current Heading =  35.00
Current Bearing =  54.04
Current Range = 354.16
GET_NEXT_WAYPOINT GOAL FLAG =    1
DIVING_SYSTEM_PROBLEM_P GOAL FLAG =    0
BUOYANCY_SYSTEM_PROB_P GOAL FLAG =    0
THRUSTER_SYSTEM_PROB_P GOAL FLAG =    1
Commanded Heading is:  34.83
Commanded Depth is:  54.84
Commanded Speed is: 360.00
Commanded X is: 450.00
Commanded Y is: 150.00
Commanded Mode is: Return
        .

        .

        .
```

APPENDIX C. AUV SIMULATOR USER'S GUIDE

To run the AUV simulator, the following is required: a file with a set of CLIPS rules, an executable file for CLIPS-Ada, an executable file for the AUV graphical simulator, and four data files for inputs to the simulator. The CLIPS rule file serves as the Strategic level. The executable file for CLIPS-Ada allows the CLIPS rules to call the Tactical level procedures. The executable file for the graphical simulator acts as the Execution level as well as the physical vehicle itself. The four data files for input are "initial_state", "waypoints", "final_goal", and "obstacles". These files must be initialized first.

Data is entered into the "initial_state" file in the format illustrated in Figure 1.
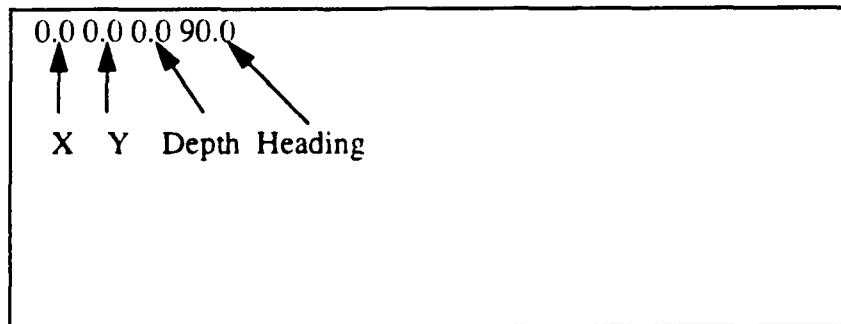

Figure 13 "initial_state" Data File

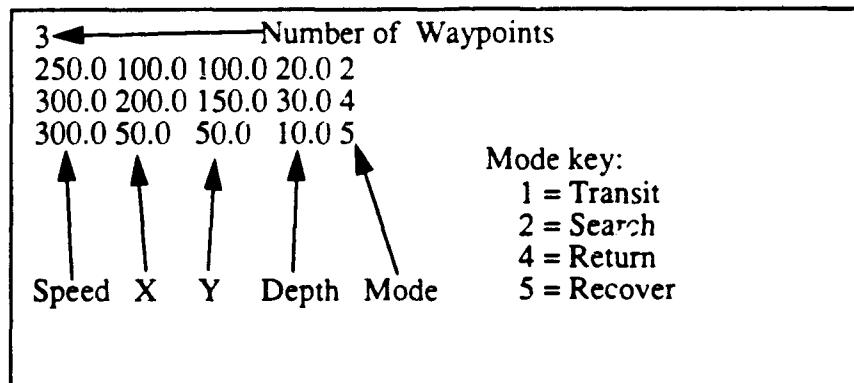Data is entered into the "waypoints" file in the format illustrated in Figure 2.

Figure 2 "waypoints" Data File

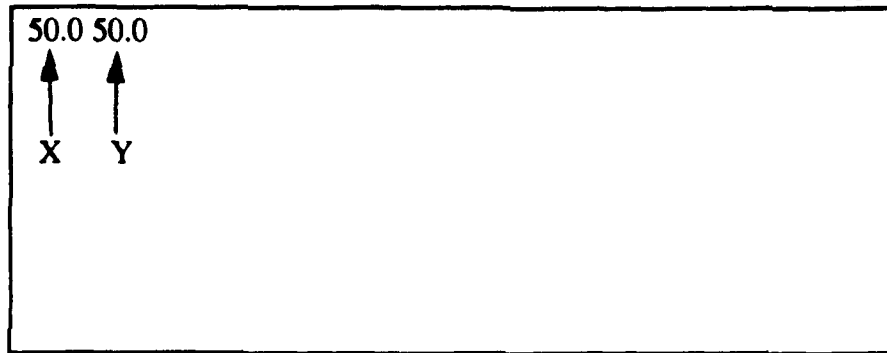Data is entered into the "final_goal" file in the format shown in Figure 3.

```
50.0 50.0
 ↑    ↑
 |    |
 X    Y
```

Figure 3   "final_goal" Data File

Data is entered into the "obstacles" file in the format shown in Figure4.

```
1 ◄───────────Number of Obstacles
50.0 100.0 20.0
 ↑     ↑    ↑
 |    /    /
 |   /    /
 X   Y   Depth
```
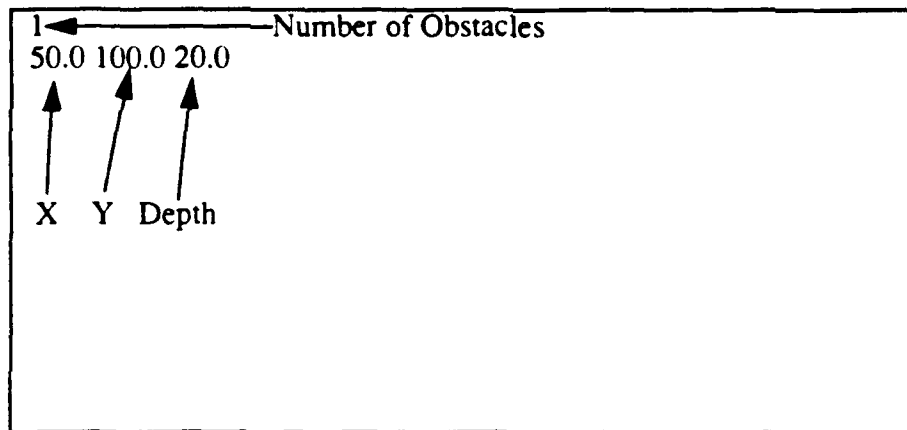
Figure 4   "obstacles" Data File

Once the data files are set up, the simulator can be run from any Silicon Graphics workstation in the Graphics laboratory. First, two window shells must be called up- the first to run the Execution level/graphical simulator and the second to run the Strategic/Tactical level. In the first window, the executable file "auv2" is run. In the second window, an rlogin to Virgo must be done and then either the "str_tac1" (multi-phase mission) or the "str_tac2" (multi-phase mission with route replanning) executable file for CLIPS-Ada must be run. At the prompt, the host name is entered as "irisn". Then the appropriate CLIPS rule set is loaded by entering "(load strlevx"). Finally, to start the simulation, a "start" fact must be asserted ("(assert (start))") and the run command must be given ("(run)"). The simulation can be stopped by killing the "auv2" process.

LIST OF REFERENCES

[Atki91]   Atkinson, C., *Object-Oriented Reuse, Concurrency and Distribution: An Ada-Based Approach*, ACM Press, New York, NY, 1991.

[Booc87]   Booch, G., *Software Engineering with Ada*, 2d ed.,Benjamin/Cummings, Menlo Park, CA, 1983.

[Booc91]   Booch, G., *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, CA, 1991.

[Byrn93]   Byrnes, R. B. , *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*, PhD Dissertation, Naval Postgraduate School, Monterey, CA, March 1993

[DoD93]   *Introducing Ada 9X: Ada 9X Project Report*, Office of the Under Secretary of Defense for Acquisition, Washington, DC, 1993.

[Geha84]   Gehani, N., *Ada: Concurrent Programming*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1984.

[Heal92]   Healey, A. J., et al., "Research on Autonomous Underwater Vehicles at the Naval Postgraduate School", *Naval Research Reviews*, Vol. 44, No. 1, pp. 16-30, August 1991.

[Hoar78]   Hoare, C. A. R., Communicating Sequential Processes, Communications of the ACM, Vol. 21, No. 8, pp. 666-677, August 1978.

[Howl88]   Howle, W. T., "Ada in Real-Time Embedded Systems: Orbital Maneuvering Vehicle (OMV)", *Proceedings of TRI-Ada'88*, pp. 363-370, Charleston, WV, Oct 24-27, 1988.

[Kwak90]   Kwak, S. H., *Rule-Based Motion Coordination for the Adaptive Suspension Vehicle on Ternary-Type Terrain*, Technical Report NPSCS-91-006, Naval Postgraduate School, Monterey, CA, December 1990.

[Kwak92]   Kwak, S. H., McGhee, R. B., and Bihari, T. E., *Rational Behavior Model: A Tri-Level Multiple Paradigm Architecture for Robot Vehicle Control Software*, Technical Report NPSCS-92-003, Naval Postgraduate School, Monterey, CA, March 1992

[Kwak93]   Kwak, S. H., *Rational Behavior Model: A Tri-Level Multiple Paradigm Architecture*, Technical Report NPSCS-93-006, Naval Postgraduate School, Monterey, CA, September 1993.

[Lema89]   Lemanski, W. J., and Hartrum, T. C., "An Assessment of the Development of a Tracking System Using Concurrent Ada", *Proceedings of the 1989 National Aerospace and Electronics Conference*,pp. 466-473, Dayton, OH, May 22-26, 1989.

[Niel88]   Nielsen, K. W. and Shumate, K., *Designing Large Real-Time Systems with Ada*, Multiscience Press, Inc., New York, 1988.

[Ong90]   Ong, S. M., *A Mission Planning Expert System with Three-Dimensional Path Optimization for the NPS Model 2 Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, June 1990.

[Scho93]   Scholz, T., *The State Transition Diagram with Path Priority and Its Applications*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1993.

[Scot88]   Scott, Barbara, "Explorer Platform Ada Flight Software", *Proceedings of TRI-Ada'88*, pp. 325-343, Charleston, WV, October 24-27 1988.

[Soft92]   *Classic-Ada User's Manual*, Software Productivity Solutions, 1992.

[Stee92]   Steer, B., Dunn, S., and Smith, S., *Advancing and Assessing Autonomy in Underwater Vehicle Technology Through Inter-Institutional Competitions and/or Cooperative Demonstrations*, Department of Ocean Engineering, Florida Atlantic University, Boca Raton, FL, May 1992.

[Stev93]   Stevens, C. D., *A Software Architecture for a Small Autonomous Underwater Vehicle Navigation System* (SANS), Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1993.

[Toml89]   Tomlinson, C., and Scheevel, M., "Concurrent Object-Oriented Programming Languages", *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F. H. Lochovsky, eds., pp. 79-124, ACM Press/ Addison-Wesley, New York, 1989.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center   2
Cameron Station
Alexandria, VA 22304-6145

Dudley Knox Library   2
Code 52
Naval Postgraduate School
Monterey, CA 93943-5002

Director, Training and Education   1
MCCDC, Code C46
1019 Elliot Road
Quantico, VA 22134-5027

Ted Lewis, Professor and Chairman   1
Department of Computer Science
Code CSLt
Naval Postgraduate School
Monterey, CA 93943-5118

Computer Technology Programs   1
Code 37
Naval Postgraduate School
Monterey, CA 93943-5119

Dr. S. H. Kwak, Code CS/Kw   5
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

Dr. R. B. McGhee, Code CS/Mz   1
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

LCDR M. K. Shields, Code EC/SL   1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

CAPT F. P. B. Thornton, Jr.
Director, MCOTEA
3035 Barnett Avenue
Quantico, VA 22134-5014

1