2000-06

Demonstration of a concurrently programmed tactical level control software for autonomous vehicles and the interface to the execution level code

Carroll, William D.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEMONSTRATION OF A CONCURRENTLY PROGRAMMED TACTICAL LEVEL CONTROL SOFTWARE FOR AUTONOMOUS VEHICLES AND THE INTERFACE TO THE EXECUTION LEVEL CODE

by

William D. Carroll

June 2000

Thesis Advisor:	Man-Tak Shing
Second Reader:	Michael J. Holden

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

TITLE AND SUBTITLE : DEMONSTRATION OF A CONCURRENTLY PROGRAMMED TACTICAL LEVEL CONTROL SOFTWARE FOR AUTONOMOUS VEHICLES AND THE INTERFACE TO THE EXECUTION LEVEL CODE	**5. FUNDING NUMBERS**
6. AUTHOR(S) Carroll, William D.	
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) N/A	**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 U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

The desire for use of autonomous robotic vehicles has undergone tremendous growth in the past decade. One of the greatest challenges to the successful development of truly autonomous vehicles is the ability to link logically based high-level mission planning with low-level vehicle control software, without a labor intensive programming effort for each mission.

This challenge can be effectively achieved through the use of tri-level control software architecture, as described in the Rational Behavior Model. The control software (in the tactical level) must de-couple the high-level mission planning from the low-level vehicle control software to reduce the programming effort for each mission. This report describes an object-oriented, modular architecture for the middle (tactical) level that uses concurrent programming techniques and multi-language interfacing. This design enables the control software to handle the intense data management effort required to operate in an autonomous fashion and interface with code already perfected for use in the strategic (top) and execution (bottom) levels.

The design was evaluated by providing the tactical level with a simple execute order statement that was then used to drive the actions of the vehicle. The software package demonstrates the validity of the design and provides the framework for full implementation on an actual vehicle.

14. SUBJECT TERMS Autonomous vehicle, robot, AUV, Rational Behavior Model, RBM, concurrency, Ada 95, control software	**15. NUMBER OF PAGES** 82
	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	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

DEMONSTRATION OF A CONCURRENTLY PROGRAMMED TACTICAL LEVEL CONTROL SOFTWARE FOR AUTONOMOUS VEHICLES AND THE INTERFACE TO THE EXECUTION LEVEL CODE

William D. Carroll
Lieutenant, United States Navy
B.S., Oregon State University, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The desire for use of autonomous robotic vehicles has undergone tremendous growth in the past decade. One of the greatest challenges to the successful development of truly autonomous vehicles is the ability to link logically based high-level mission planning with low-level vehicle control software, without a labor intensive programming effort for each mission.

This challenge can be effectively achieved through the use of tri-level control software architecture, as described in the Rational Behavior Model. The control software (in the tactical level) must de-couple the high-level mission planning from the low-level vehicle control software to reduce the programming effort for each mission. This report describes an object-oriented, modular architecture for the middle (tactical) level that uses concurrent programming techniques and multi-language interfacing. This design enables the control software to handle the intense data management effort required to operate in an autonomous fashion and interface with code already perfected for use in the strategic (top) and execution (bottom) levels.

The design was evaluated by providing the tactical level with a simple execute order statement that was then used to drive the actions of the vehicle. The software package demonstrates the validity of the design and provides the framework for full implementation on an actual vehicle

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Most importantly, I would like to thank my wife, Sheilah, for her love, support, and encouragement through this work. Also, many thanks to Mike Holden and Professor Shing for without their effort and guidance this project would not have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Reliable robot vehicles, capable of safely performing complex actions without the need to place a human in harm's way, have become a top priority in today's world. To realize the greatest benefit, these robot vehicles must be able to operate autonomously in a rational manner in performance of their tasks. Autonomous vehicles are generally defined as vehicles that are capable of reasonably "intelligent" motion and action without requiring either a guide to follow or an operator to control them in real time [1].

Of particular interest to our naval forces is the deployment of such devices to reduce or eliminate the catastrophic effect that mine warfare has in today's littoral warfare. The following quote taken from the Naval Mine Warfare Vision 2010 emphasizes this point:

> Naval Mine Warfare comprises a critical part of our future warfighting capability. The proliferation of mines throughout the world as cheap means of sea control and the downsizing of our Naval forces dictate that mine countermeasures . . . become an integral part of our National Military Strategy. Naval Mine Warfare will perform an enabling role for Joint and Coalition forces [2].

That same document goes on to state that United States Naval forces must possess Autonomous Underwater Vehicles (AUVs) to facilitate rapid and thorough clearance of any mined sea lanes [2].

One of the greatest challenges to the successful development of truly autonomous mine hunting vehicles is the ability to link logically based high-level mission planning software with low-level vehicle control software. Control software for these systems exists at the highly abstract “logical” level and at the extremely low "hardware operations” level [3]. The implementation of these two levels results in specific top-to-bottom software interaction that are hard-coded for a particular application and task. A change in implementation, and even the simple addition of a new capability, results in a need to re-work the code at both ends. This code rework invites the introduction of new errors into the code as well as increasing the overall code complexity.

What is required is an intermediate level, a generic framework that can be both mission and platform independent. This intermediate level would provide standard Application Programming Interfaces (API’s) for low level components while having the ability to accept a wide range of high-level mission commands and tasking. An API is the software that is used to support system-level integration of software products or newly developed software into existing or new applications. APIs provide for interoperability across different platforms; this is an important feature when developing new or upgrading existing [distributed] systems [4].

An approach to implement this intermediate level is to utilize a Rational Behavior Model (RBM), developed in detail by Byrnes [5] and implemented by Kwak [6], Holden [3], and Leonhardt [7] for the Naval Postgraduate School (NPS) Phoenix AUV. The RBM is a three-level software architecture consisting of Strategic, Tactical and Execution levels with respective emphasis on mission planning, programmed vehicle responses labeled "behaviors," and efficient real-time execution of vehicle hardware control programming. The RBM is described in Chapter II.

B. APPROACH

This work builds on those completed by Kwak, Holden and Leonhardt by continuing to enhance the design of a Tactical level control software package for an autonomous robotic vehicle. This enhancement of the Tactical level is accomplished by incorporating object oriented software design and implementing it using concurrent tasking techniques and the multilanguage interfacing capabilities available using the Ada 95 programming language [8]. The design was demonstrated on a single processor Personal Computer highlighting the benefits of concurrent tasking and the advantages of multiple processes "sharing" a single Central Processing Unit (CPU). These design enhancements move the promise of rationally-behaving autonomous vehicles further toward the goal of rapidly deployable vehicle control software, without a labor intensive programming effort for each mission.

C. SCOPE

A representative Tactical level software package for an Autonomous Underwater Vehicle was developed using the Ada 95 programming language. Use of Ada 95 enabled the design to incorporate multiple tasks (processes) and a multilanguage interface to the execution level software. The Execution level software used for this work was taken from the A.R.I.E.S. AUV developed by the Center for AUV Research at the Naval Postgraduate School [9]. The A.R.I.E.S. is described in Chapter II.

Within the Tactical level software individual Ada tasks were used to modularize code into separate concurrently operating processes synonyms with the delegation of responsibility performed by a human submarine crew [3]. For the scope of this thesis the main controlling process is referred to as the Officer of the Deck (OOD). The OOD process and its related function packages will perform the mission planing and coordinate the efforts of the entire vehicle. Navigator, Engineer, and Deck Log processes were incorporated to perform the individual tasks of vehicle navigation, propeller motor and control surface actuation, and event recorder respectively.

This thesis will demonstrate the validity of the design by providing the Tactical level software package with a simple high-level execute order statement. That order will then be used to initiate the required actions to perform the mission. The interface package will enable the Tactical level to make calls to the Execution level code to drive the propeller motors and to position the control surfaces of the autonomous vehicle. This software package demonstrates asynchronous control transfer between tasks running concurrently, interaction (communication) between tasks, and function calls to existing Execution level software. This provides the framework for full implementation on an actual vehicle.

D. THESIS ORGANIZATION

Chapter I: Introduction. This chapter gives a general outline of the work, including motivation, approach, scope of the work, and the thesis organization.

Chapter II: Background. This chapter contains pertinent background information on Unmanned Underwater Vehicle (UUV) programs, the Rational Behavior Model

(RBM), Software Engineering, Concurrent Programming, and the NPS A.R.I.E.S. Underwater Autonomous Vehicle.

Chapter III: Software Architecture. This chapter describes the Tactical level software architecture and the interface to the Execution level.

Chapter IV: Implementation. This chapter describes the implementation and execution of the code. It provides necessary information and program code to conduct the experiment.

Chapter V: Conclusions and Recommendations. Includes theoretical improvements and future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. US NAVY UNMANNED UNDERWATER VEHICLE PROGRAM

To meet the requirement for developing Autonomous Underwater Vehicles (AUVs) for mine reconnaissance the Director of the Navy's Expeditionary Warfare Division (N85) has been given the responsibility for establishing the Navy's Unmanned Underwater Vehicle (UUV) Program. The Navy's first priority in its UUV plan is rapid development of a covert mine reconnaissance capability [11]. A two tiered approach was implemented to develop the systems needed to provide both near term and long term systems to meet the requirements set forth in the UUV Program Plan [10].

The first was understandably labeled the Near Term Mine Reconnaissance System (NMRS) program. This program capitalizes on existing technologies for rapid deployment of a mine reconnaissance system. The NMRS will utilize a vehicle controlled via fiber-optic cable connected to the launch platform [12]. This approach highlighted the fact that true autonomy was not yet achievable for the deployment of the NMRS. The second program labeled the Long Term Mine Reconnaissance System (LMRS), was directed to develop the system that would eventually replace the NMRS. This program concentrated on investigating emerging technologies and developing new ones that would provide significantly improved capability over the NMRS, namely autonomous operations endurance of more than 40 hours [11]. This thesis seeks to further the development of truly autonomous vehicles by providing a framework for a robust software architecture capable of controlling an Autonomous Underwater Vehicle (AUV).

B. RATIONAL BEHAVIOR MODEL (RBM)

The Rational Behavior Model (RBM) develops an approach to linking high level logical mission planning for autonomous vehicles with low-level vehicle control programming. The result is a three-level software architecture consisting of the Strategic, Tactical and Execution levels, each to be implemented in a way perfected or better suited for use at that level. The Strategic level is programmed with emphasis on mission planning, the Tactical level is programmed for vehicle responses ("behaviors"), and the Execution level uses efficient real-time execution of vehicle hardware control programming [3]. Figure 1 illustrates the relationships between the three levels of the RBM.

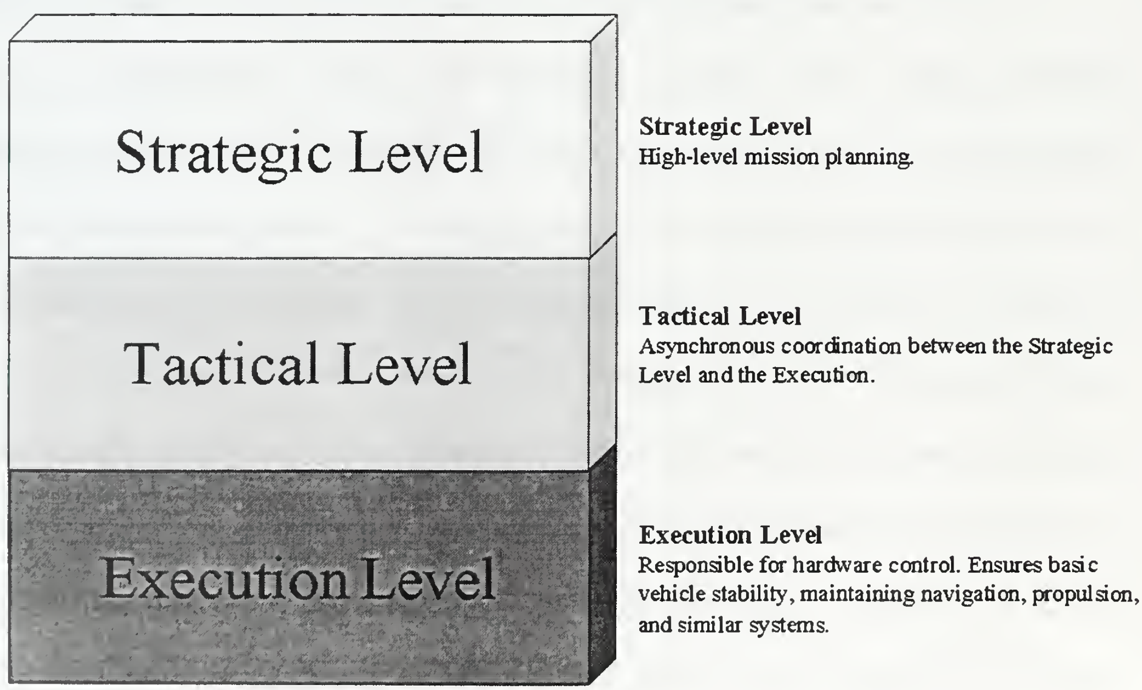

Figure 1. Rational Behavior Model

1. Strategic Level

The Strategic level is comprised of essential mission planning software. It uses high-level mission logic and provides for the deterministic sequencing of the underlying behaviors implemented for that particular autonomous vehicle [5].

2. Tactical Level

The Tactical level includes programmed vehicle responses and implements the behaviors capable of satisfying the goals assigned by the Strategic level. It acts as the intermediary under the Strategic level direction and provides an interface for issuing the commands necessary to direct the performance of the Execution level. The Tactical level must also interact with the Strategic level either explicitly, as answers to specific queries, or to simply respond upon the completion of a commanded behavior [5].

Behaviors contained within this level are non-logic-based executed processes being performed by one or more entities within the Tactical level. The use of more than one entity will enable asynchronous control of necessary functions to enable the vehicle to operate in an autonomous fashion [3].

3. Execution Level

The Execution level provides efficient real-time execution of vehicle hardware control programming. Responsible for all of the physical actions of the vehicle, this is the software intermediary between the Tactical level and the actual hardware of the vehicle, and must meet all the hard real-time scheduling requirements to ensure basic vehicle stability, maintaining navigation, propulsion, and similar systems [5].

C. SOFTWARE ENGINEERING

The Software Engineering approach to developing software applications or systems is one of forethought rather than afterthought. Traditional engineering practices such as requirement documentation, analysis of design, modeling, component testing, and incremental inspections are common place in electrical, mechanical and civil engineering projects. All of these practices serve to prevent design changes during construction (which are often physically impossible to do), or failure of the completed project during its useful life span. All too often software projects are kicked off before any of these critical issues are considered.

In addition to the use of the engineering design philosophies mentioned above during the development phase of designing software systems, the following principles are also considered when taking a Software Engineering approach:

Maintainability: the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a change.

Reusability: the degree to which a software module or other work product can be used in more than one computing program or software system.

Flexibility: the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Scalability: the ease with which a system or component can be modified to fit the problem area [13].

The software package for control of an autonomous vehicle described in this thesis was developed with these Software Engineering principles in mind. It incorporates object oriented software design for modeling the application domain. The model used is based on human operators in a manned submarine for modularity. It is implemented using concurrent tasking techniques for performance, flexibility, and scalability, and it uses multilanguage interfacing capabilities to take advantage of code reuse.

D. CONCURRENT PROGRAMMING

Traditional programming techniques involve a sequence of actions performed one after another. Concurrent programming entails two or more traditional sequences of actions to be performed concurrently within the same program. Concurrent programming enables asynchronous control transfer, meaning a process can initiate the task to perform some other action and then can continue its own sequence while the other process (task) is busy fulfilling the request [14].

1. Single processors

The multitask program that is running on a single central processor unit (CPU) computer will share that computer's CPU between tasks. This is called interleaved concurrency. The benefit to multitask programs running on a single CPU computer are realized when a wait, on some external event such as the completion of an input operation, or delay occurs in a task that is accessing the CPU. While a task is delayed the other task(s) can access the CPU. Very short, 1/100 sec, delays can be preprogrammed into the sequence of tasks to force time sharing of the CPU by the various tasks.

2. Multiple processors

If the multitask program is compiled to run on a multiple processor computer then different processors will actually execute different tasks at the same time. This is called overlapped concurrency. The compiler handles the scheduling of multitasked programs, enabling the same program that is implemented on a single CPU computer to be re-compiled for use on a multiple CPU machine.

Concurrent programming techniques are used for many different reasons. Programs designed to monitor or control several devices are most easily written with one task managing each individual activity. The use of tasks can allow programs to finish more quickly by sharing the CPU or through the use of multple CPU's. Simulation programming can benefit by using tasks designed to run within the rules of each entity modeled for the simulation [14].

E. NPS A.R.I.E.S AUV

The Center for Autonomous Underwater Vehicle (AUV) Research at the Naval Postgraduate School (NPS) designed and built the Acoustic Radio Interoperative Exploratory Server (A.R.I.E.S.) AUV for research and development of AUV systems. The A.R.I.E.S. is the replacement vehicle for the NPS Phoenix AUV described in the work done by Kwak [6], Holden [3], and Leonhardt [7]. The Phoenix has been decommissioned and now sits as a display in the NPS research museum. The A.R.I.E.S. is shown in figure 2.

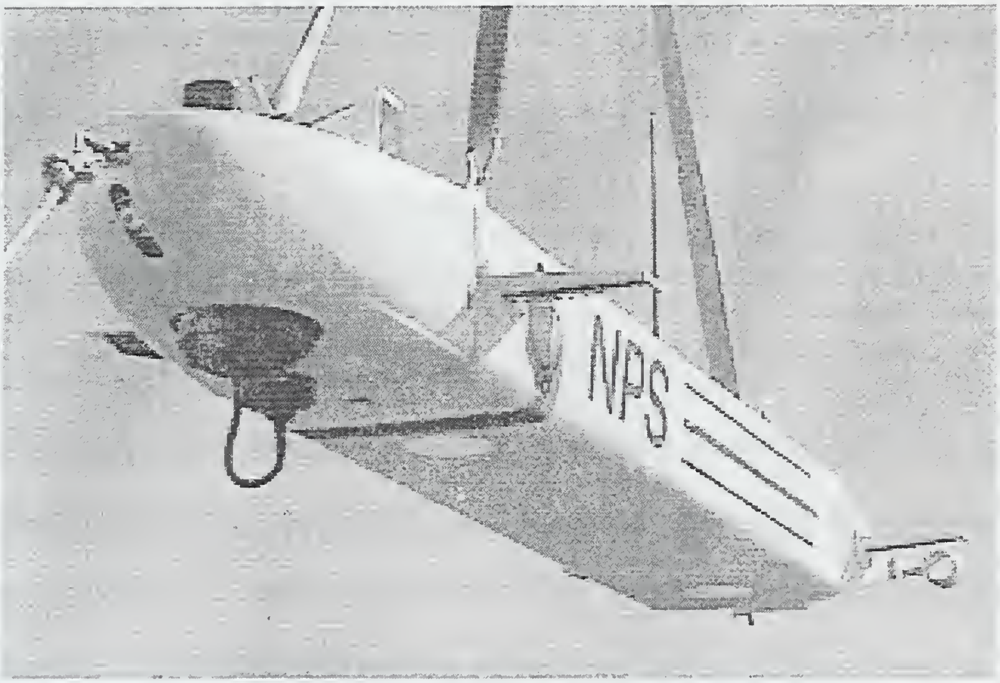

Figure 2. NPS A.R.I.E.S. Autonomous Underwater Vehicle

The term "Server," used in the acronym describing the latest NPS AUV comes from research in the use of multi-vehicle fleets of AUVs linked to a supervisor vehicle, or server, for minesweeping operations [15]. The A.R.I.E.S. design incorporates an acoustic modem to facilitate data links between AUVs while under water. The A.R.I.E.S. uses dual computer architecture with each computer dedicated to perform specific vehicle software and hardware functions. It uses a modular multi-rate, multi-process configuration for semi-autonomous and autonomous underwater vehicle operation. The two computers communicate over standard TCP/IP network sockets. Other computers can be logged into the vehicles network either by cable or wireless connection. The dual computer implementation uses one system for data gathering and running navigation filters, while the second computer uses the output from the first computer to operate the various auto-pilots for servo level control. The A.R.I.E.S. performs its mission in accordance with a sequential mission script file that is preloaded onto the vehicle, or can be downloaded/modified via an external computer logged into the vehicle's network [9].

The only relation between A.R.I.E.S. and this thesis was the partial use of A.R.I.E.S. execution level software code that drives the propeller motors and positions the control surfaces of the vehicle in response to the auto-pilots direction.

The file named Execf.c, for execution functions, was written in C programming language by Dr. Dave Marco, Dept. of Mechanical Engineering, Naval Postgraduate School, Monterey California. Only the functions related to driving the propeller motors and positioning the control surfaces were adopted from Dr. Marco's original code. Other lines of code within the borrowed functions that were not pertinent to this work were deleted. The shell of the actual code used onboard an operating AUV was used to highlight the capability of the design. The functions that were selected to interact with the Tactical level code were used to simulate control of the following hardware components onboard the A.R.I.E.S. AUV: left propeller, right propeller, left bow plane, right bow plane, left stern plane, right stern plane bow rudder, and a stern rudder. The A.R.I.E.S. AUV also incorporates bow and stern lateral thrusters and bow and stern vertical thrusters [9]. The control of these last four components was not addressed in this thesis.

III. SOFTWARE ARCHITECTURE

A. INTRODUCTION

This chapter describes the Software Architecture applied to the design of the representative tactical level software package for an Autonomous Underwater Vehicle. The architecture was designed for the Ada 95 programming language and includes the components required for the interface to the Execution level software. Figure 3 illustrates how the Tactical level architecture fits within the framework of the RBM.

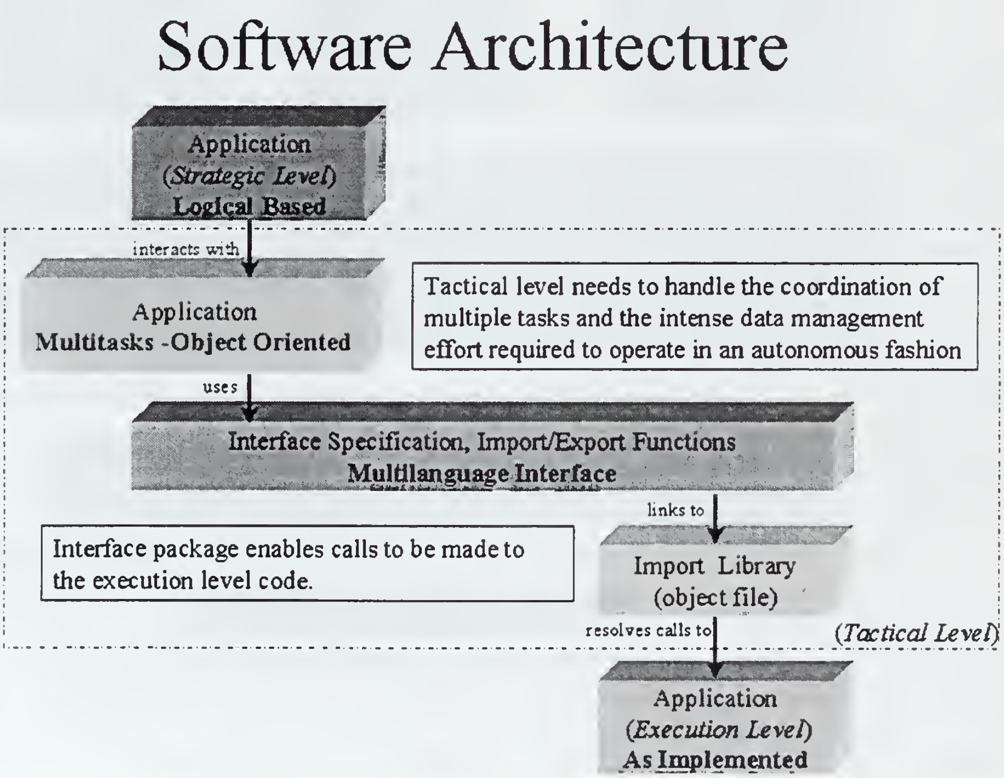

Figure 3. Autonomous Vehicle Software Architecture

B. APPROACH

This work's ultimate goal was to provide a robust software architecture capable of performing the intense data management required for a robot vehicle to operate autonomously in the performance of its mission. To accomplish this, Ada tasks were used to provide concurrency among functions modeled after human submarine operators. This approach also served to modularize functions in a logical manner. Figure 4 illustrates the major components within the Tactical level software architecture.

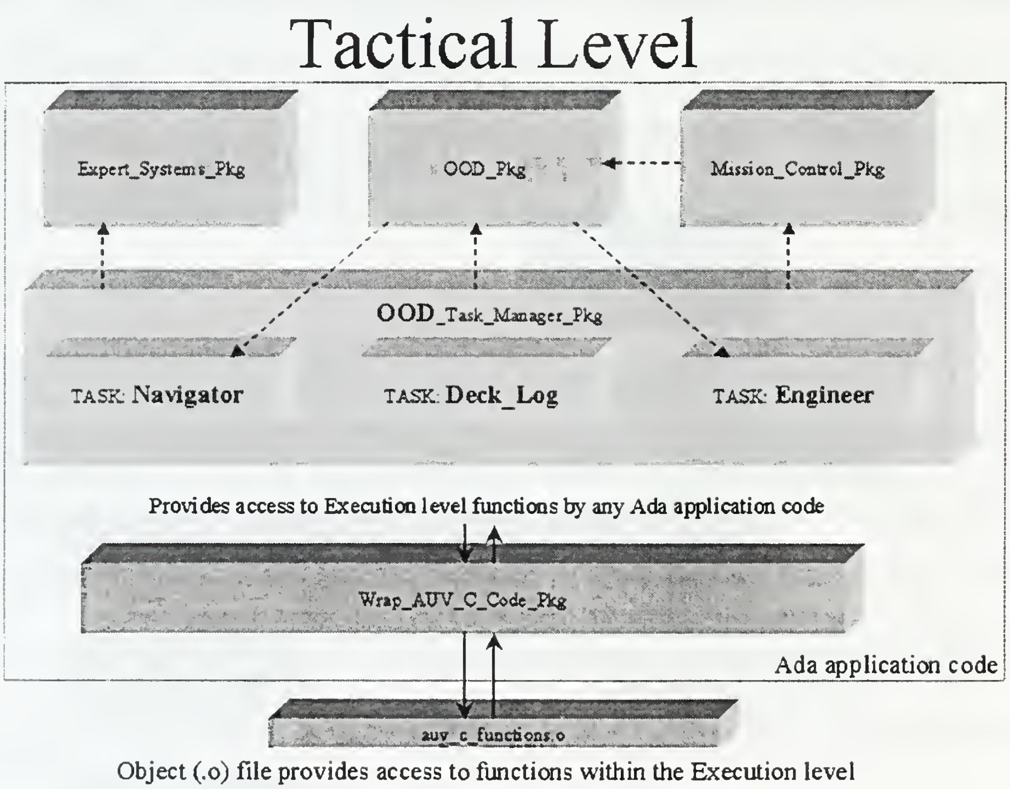

Figure 4. Tactical Level Components

1. Tactical Level Application Components

The main controlling process for the Tactical level is referred to as the Officer of the Deck (OOD) analogous to the human watch stander in charge of all operations aboard a naval vessel. The OOD will perform the mission planing and coordinate the efforts of the Ada Tasks utilized within this Tactical level software architecture. Ada Tasks are spawned, concurrently, to perform specific actions or for continuos control of critical parts of the robot vehicle to maintain stable operation. The Navigator, Engineer, and Deck Log tasks were incorporated in this demonstration to perform the individual tasks of vehicle navigation, propeller motors and control surface actuation, and event recorder respectively. The use of Ada tasks enables the tasks' sequential procedures to be performed independent of the operation of the OOD, or any other task unless specifically programmed rendezvous are required by the software design [14]. The major packages and procedures utilized for the demonstration of this software architecture are described below. A more detailed description is found in chapter IV, Implementation.

The OOD Task Manager package receives the simple high-level execute order statement from the Strategic level via its Receive Orders procedure. That order will then be used to invoke the Officer Of The Deck procedure in the body of the OOD Task Manager package to control the rest of the required actions to perform the mission. When the Officer Of The Deck procedure is finished, the Tactical level is exited and control is returned back to the Strategic level. The Officer Of The Deck procedure calls the Mission Planner procedure to carryout the orders received. The Mission Planner contains the sequential mini-missions, which make up the complete operation directed by the simple high-level execute order statement sent from the Strategic level. The mini-missions are accessed with the appropriate call to the Mission Control package. The Mission Control Package contains the detailed sequence of events for performing the mini-missions. This method provides for rapid modification or addition of new missions on a robotic vehicle by simply modifying the existing functions or adding new ones.

The Officer Of The Deck procedure and the procedures within the Mission Control package all utilize the OOD package to perform their respective mission sequences. The OOD package modularizes the repeatable actions performed by the OOD. Removing these functions from the OOD Task Manager package reduces complexity and enhances the readability of the code.

The Expert Systems package contains functions that would utilize specialized algorithms, access to database information, and input sources necessary to return the appropriate information/data back to the requesting Tactical level entity. They can be used by the Officer of the Deck procedure itself or any of the tasks as required to complete their function. This method supports upgrades and expandability by providing standard interfacing specifications at the time of design. This implementation simulates two expert system functions. The first is called to determine the next course to station. The second is called to determine a course for which to begin the mine-hunting mission.

2. Interface to the Execution Level

The Wrap AUV C Code package contains the wrapper functions required for the Tactical level to make calls to and receive calls back from the Execution level. A wrapper function in Ada contains the standard Ada function interfaces to interact with the rest of the Ada program code. For each of these functions an import or export pragma is used to provide the required interface information for access to/by the other language function [14]. The use of this package enables the Tactical level to link to the vehicles Execution level functions which control the hardware, input/output devices, and sensors.

The Execution level code is written in a language decided on by the vehicle hardware developers, and is platform specific. An Ada interface can be provided for a variety of software languages and could support many different platforms [8].

In order to interface the Execution level code an object file (.o) must be included in the linker options when the Ada code is compiled. An object file is created when compiling the execution level code. The object file (.o) enables the Ada code to be linked to the Execution level functions during the compilation of the main Tactical level application. An interface package using wrapper functions as described above is then written in Ada to handle the code interaction between the Ada application and the other programming language functions.

A concern, which is not addressed in this thesis, is the requirement to account for compiler, code, and operating system compatibility. There are many combinations that will work and many new methods and compilers are becoming available on a continuing basis.

The architecture utilized for this thesis contains both import and export pragma functions. These functions enable two way interactions between the Ada application program and the execution level code written in C programming language and are described in detail in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

A. STRATEGIC LEVEL IMPLEMENTATION

The Strategic level was interfaced as a "black box" for the purpose of this thesis. A single Ada program with a procedure named CO_Strategic_Level was used. This procedure initiates the high level command that would be given by a logic based Strategic level program. For this demonstration the direction given to the Tactical level was simply what to do and where to do it. The complete code can be found in Appendix A, Section 1.

B. TACTICAL LEVEL IMPLEMENTATION

The Tactical level is comprised of six Ada software packages. Each Ada package is comprised of a specification file and a like named body file. The specification file contains the interface descriptions for the procedures and functions that are implemented in the package body. The packages used in this demonstration are described below. The complete code can be found in Appendix A, Section 2.

1. OOD Task Manager Package

The OOD_Task_Manager_Pkg controls and directs the actions of the AUV to meet the assigned mission. The Officer_Of_The_Deck procedure within the OOD_Task_Manager_Pkg is the sequential series of statements and function calls that culminate in the completion of the assigned mission. The procedure begins when the order is sent from the Strategic level code to the Tactical level via a call to the procedure Receive_Orders. With the call to this procedure comes the pertinent information on what to do and where to do it. The Receive_Orders procedure is the only connection from the Strategic to the Tactical level. Subsequent interaction back to a Strategic level is not addressed in this demonstration.

The Mission_Planner procedure, when invoked by the Officer_Of_The_Deck procedure, makes the call to the appropriate procedure within the Mission_Control_Pkg. This enables the OOD to call various Mission Control procedures multiple times in order to complete a larger mission goal.

The OOD_Task_Manager_Pkg also contains several Ada Tasks to concurrently perform specific actions or for continuous control of critical parts of the AUV to maintain stable operation. The Navigator (NAV), Engineer (ENG), and Deck Log (LOG) tasks will immediately be spawned upon initialization of the main program. These tasks will be blocked at their accept entry point and become available to act as directed by the Officer_Of_The_Deck procedure and also by procedues from within the Mission_Control_Pkg. The Navigator, Engineer, and Deck Log tasks were incorporated in this demonstration to perform the individual tasks of vehicle navigation, propeller motor and control surface actuation, and event recorder respectively.

a. Ada Tasks

Both the Navigator (NAV) and the Engineer (ENG) tasks utilize three Ada task accept statements as entry points. The three accept statements are Taking_Action, Making_Report, and NAV_Aye or ENG_Aye. The accept statements Taking_Action and Making_Report facilitate communication among procedures and tasks. The NAV/ENG_Aye allows for an action order to be sent to the appropriate task.

The Navigator (NAV) task provides for functions regarding ship's position and course to station. A case selection is used based on an order type sent to the accept statement NAV_Aye. The order types CourseToStation and GivePosition perform the function as the names apply

The Engineer (ENG) task interacts with the Execution level code to drive the propeller motors and position the control surfaces. A case selection is used based on an order type sent to the accept statement ENG_Aye. The order types AllStop, AllAhead, PortStop, PortAhead, PortBack, StbdStop, StbdAhead, and StbdBack provide for propeller motor control. The order types RightRudder, LeftRudder, UpPlanes, and DownPlanes provide for positioning the control surfaces. The order type EmergencySurface is the abort mission call and sets the propeller motors and control surfaces to return the AUV to the surface of the ocean.

The Data Logger (LOG) takes all communications that utilize the communicate procedure within the utilities package and logs them in a text file.

2. OOD Function Package

The OOD_Pkg provides for modularization of OOD actions. The procedures Taking_Action and Roger_Out facilitate communication among procedures and tasks. The procedure Give_Order allows for an action order to be sent to the appropriate task using the case selection described above.

3. Mission Control Package

The Mission_Control_Pkg contains detailed sequences for performing specific mini-missions. The mini-missions are pieced together to complete the requirements of the high-level mission order statement.

4. Expert Systems Package

The Expert_Systems_Pkg contains functions that would utilize specialized algorithms, access to database information, and input sources necessary to return the appropriate information/data back to the requesting Tactical level entity.

5. C Code interface

The Wrap_AUV_C_Code_Pkg provides access to Execution level functions. This interface package enables the Tactical level to make calls to the Execution level code and, in this case, simulate driving the propeller motors and positioning the control surfaces of the AUV.

The key to the interface is the object file created when compiling the Execution level code. The Object File (.o) enables the Ada code to be linked to the Execution level functions during the compilation of the main Tactical level Application.

Three procedures, Text_From_C_Function, End_Text_From_C_Function, and Double_From_C_Function utilize the pragma Export to enable the Execution level to communicate back to the Ada program for simulated response by the vehicle propellers and control surfaces.

The remaining procedures all utilize pragma Import to give the Tactical level code access to the Execution level functions. They are: Stop_Screw_Motors, Rudder_Angle, Planes_Angle, Zero_Fins, Abort_Mission, Left_Screw_Speed_Control, and Right_Screw_Speed_Control. They all give Execution level control access to the Tactical level as each procedure name applies.

6. Utilities Package

The Utilities_Pkg provides for screen output formatting and system clock functions. A Communicate procedure is used to provide a way for all information exchanges to be logged in the vehicle deck log and to provide the screen output for use in code development and debugging efforts.

C. EXECUTION LEVEL IMPLEMENTATION

The Execution level code used for this thesis was written in C programming language. The code used is based on program functions written by Dr. Dave Marco [19] for the NPS A.R.I.E.S. AUV and was modified by the author as indicated within the code. The complete code can be found in Appendix A, Section 3.

The functions used for this thesis are used to drive the propeller motors and position the control surfaces of A.R.I.E.S. The NPS A.R.I.E.S. AUV has a left and right propeller motor and the following control surfaces: left bow plane, right bow plane, left stern plane, right stern plane. bow rudder, and a stern rudder. The NPS A.R.I.E.S. AUV also incorporates bow and stern lateral thrusters and bow and stern vertical thrusters. The control of these components was not included in this thesis. The interface to the propeller motors and the control surfaces functions are:

StopScrewMotors( ) - sets motor control voltage to zero for both motors.

ScrewMotor(int Motor, double ControlVolt) - sets the indicated motor control voltage to the designated voltage.

ControlSurface(int Surface, double Angle) - sets the indicated control surface to the desired angle.

Rudder(double Angle) - sets the rudders to the desired angle.

Planes(double BowAngle, double SternAngle) - sets the planes to the desired angle.

ZeroFins( ) - sets all control surfaces to zero angle.

Abort( ) - sets the motor control voltage for both the left and the right propeller to ahead propulsion, and sets the control surfaces to bring the vehicle to the surface of the water.

LeftScrewSpeedControl(double n-com) - sets the control voltage sent to the left propeller motor to the desired level.

RightScrewSpeedControl(double n-com) - sets the control voltage sent to the right propeller motor to the desired level.

D. CODE TEST SCENARIO

The text in Appendix B is from the screen output during code execution. The high-level order statement from the Strategic level directs the AUV to hunt for mines at a specific Latitude and Longitude. The first step in completing this mission is to transit to the indicated position. The NAV task is accessed to give a course to station. The NAV task accesses the appropriate Expert System function, which will compute the course station. When the NAV task returns the course to station to the OOD, the OOD then gives the order to the ENG task to make way and gives a rudder order to come to that course. When on the appropriate course the order is given for rudders amidships. A full implementation can have the NAV task and the ENG task interact to maintain on track as current and sea state act on the vehicle. When on course the OOD gives the order to the ENG task to dive the Vehicle underwater. When at the desired depth the OOD orders zero planes. The OOD queries the NAV task for the current location and is informed that they are at the directed position to begin hunting for mines. The OOD orders the ENG task to come to all stop. At this point the transit operations are complete and control transfers to the Hunt Mines procedure. The OOD requests a course to hunt mines for the NAV task. The NAV task accesses the appropriate Expert System function to compute the course to hunt mines. When the NAV task returns the course to hunt mines, the OOD then gives the order to the ENG task to make way and gives a rudder order to come to that course. When on the appropriate course the order is given for rudders amidships. The report then comes saying that they have completed the Mine-Hunt operation. The OOD gives the ENG task the order to surface and the AUV is recovered.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

The Tactical level software architecture design described in this thesis has been implemented and was successfully demonstrated on a personal computer running under Windows NT 4.0 service pack 5. The success of this partial implementation of a concurrent Tactical level working within the proven design of the Rational Behavior Model provides the framework needed for full implementation and testing of the design on an actual robotic vehicle.

B. IMPROVEMENTS OVER PREVIOUS DESIGNS

This design provides the flexibility required for a robotic vehicle to perform multiple missions without the need to re-work the code at both ends. This is accomplished through the use of the Mission Control Package. Multiple mission profiles can be preprogrammed into the vehicle and accessed as required to perform a specific mission.

The robustness required for a robot vehicle to handle the intense data management needed to operate autonomously is gained through the use of Ada tasks. These tasks allow for concurrent program sequences to perform specific actions independent of each other.

The design enables the use of the Tactical level program to be portable to other platforms. Only the Interface Package needs to be modified to facilitate a new vehicles Execution level code interface.

C. RECOMMENDATIONS

The results of this research are promising. Full development of this software design would improve existing AUV operational capabilities and provide a valuable source of research in the field of mobile robotics. Effort should be made to incorporate this design in building a Tactical level software module that would address all the required procedures and functions needed for a robotic vehicle to autonomously complete a mission.

D. FUTURE WORK

This work shows the framework of a Tactical level using Ada tasks and the interface to the Execution level code written in a different programming language. There is more work required to develop the complete design and incorporate all the necessary functions and procedures required for autonomous operation of a robotic vehicle. A few specific areas of further research needed are listed below:

1. Full implementation using a software simulated vehicle

Develop the complete design and incorporate all the necessary functions, procedures, and tasks required for autonomous operation of an AUV. This development should proceed using a software simulation of an actual operating AUV. This would enable the Tactical level software development to occur concurrently with the development of the vehicle and its Execution level software.

2. Expert systems within the Tactical Level

The key to total robot vehicle autonomy lies in the ability to relate experienced-based knowledge to vehicle control software.

Expert systems will be required to enhance the operational capability of the robot vehicle. Interfaces can be established even if a particular Expert System technology has yet to mature. When the system matures and becomes available it can easily be incorporated into the desired vehicle.

3. Porting to Multiprocessor Platform

This thesis incorporated the use of a single CPU computer. Further research into using multitasked control software on multiple CPU computers promises some distinct advantages. With the addition of multiple processors the compiler will be able to distribute the load evenly between tasks and enhance system performance.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CODE

1. STRATEGIC LEVEL CODE

```
--|--------------------------------------------------------------------
--| FileName   : CO_Strategic_Level.ada
--| Author     : LT William D. Carroll, USN
--| Date       : June 2000
--| Course     : N/A
--| Project    : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: This program provides the Tactical level with a simple
--|              order statement: what to do and where to do it.
--|--------------------------------------------------------------------

WITH Ada.Text_IO;
WITH Utilities;
WITH OOD_Task_Manager_Pkg;

PROCEDURE CO_Strategic_Level IS

BEGIN

--North latitudes, and WEST longitudes are entered as positive
--numbers, but it is not necessary to use a "+" sign.
--For example, 45.00° North would be entered as 45.00
--South latitudes and EAST longitudes are entered as negative
--numbers using a "-" sign.
--For example, -125.00 represents 125.00° East Longitude.

  OOD_Task_Manager_Pkg.Receive_Orders(Orders    => "Hunt mines",
                                      Latitude  => 36.7,
                                      Longitude => 121.85);

END CO_Strategic_Level;
```

2. TACTICAL LEVEL CODE

```
------------------------------------------------------------------------
--| FileName    : OOD_Task_Manager_Pkg.ads
--| Author      : LT William D. Carroll, USN
--| Date        : June 2000
--| Project     : Thesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: This Package receives the direction from the Stratigic
--|               level through the Receive_Orders procedure. The code
--|               for the NAV, ENG, & LOG tasks are located here.
------------------------------------------------------------------------

PACKAGE OOD_Task_Manager_Pkg IS

  SUBTYPE Name_String_Type IS String (1..10);
  SUBTYPE Order_String_Type IS String (1..10);
  SUBTYPE Report_String_Type IS String (1..20);
  SUBTYPE Course_String_Type IS String (1..3);
  TYPE Name_Type IS (OfficerOfTheDeck, Navigator, Engineer);
  TYPE Order_Type IS (HuntMines, MineHuntCourse,
                      CourseToStation, GivePosition,
                      AllStop, AllAhead,
                      PortStop, PortAhead, PortBack,
                      StbdStop, StbdAhead, StbdBack,
                      EmergencySurface,
                      RudderAmidship,
                      RightRudder, LeftRudder,
                      ZeroPlanes,
                      UpPlanes, DownPlanes, None);


------------------------------------------------------------------------
--| TASK NAV(Navigaton Officer)
------------------------------------------------------------------------
TASK NAV IS
  ENTRY Taking_Action;
  ENTRY Making_Report(Name   : IN Name_Type;
                      Report : Report_String_Type);
  ENTRY NAV_Aye(Name      : IN OUT Name_String_Type;
                NavOrder  : Order_Type;
                Latitude  : Float := 0.0;
                Longitude : Float := 0.0);
END; --NAV

------------------------------------------------------------------------
--| TASK ENG(Engineering Officer)
------------------------------------------------------------------------
TASK ENG IS
  ENTRY Taking_Action;
  ENTRY Making_Report(Name : IN Name_Type);
  ENTRY ENG_Aye(Name     : IN OUT Name_String_Type;
                EngOrder : Order_Type);
```

```
END; --ENG

----------------------------------------------------------------------
--| TASK LOG(Deck Log Entries)
----------------------------------------------------------------------
TASK LOG IS
  ENTRY Log_It(Item    : IN String;
               Hour    : IN Integer;
               Minute  : IN Integer;
               Seconds: IN Float);
  ENTRY Close_Log;
END; --LOG


----------------------------------------------------------------------
--| PROCEDURE   Receive_Orders
----------------------------------------------------------------------
PROCEDURE Receive_Orders(Orders      : Order_Type;
                         Latitude    : Float;
                         Longitude : Float);



END OOD_Task_Manager_Pkg;


----------------------------------------------------------------------
--| Filename    : OOD_Pkg.ads
--| Author      : LT William D. Carroll, USN
--| Date        : 30 May 2000
--| Project     : Thesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: OOD Procedures
----------------------------------------------------------------------
WITH OOD_Task_Manager_Pkg;

PACKAGE OOD_Pkg IS


----------------------------------------------------------------------
--| PROCEDURE Give_Order
----------------------------------------------------------------------
PROCEDURE Give_Order(Name : IN OOD_Task_Manager_Pkg.Name_Type;
                     Order: OOD_Task_Manager_Pkg.Order_Type;
                     Latitude  : Float := 0.0;
                     Longitude : Float := 0.0);

----------------------------------------------------------------------
--| PROCEDURE Taking_Action
----------------------------------------------------------------------
PROCEDURE Taking_Action;

----------------------------------------------------------------------
--| PROCEDURE Roger_Out
----------------------------------------------------------------------
```

```
PROCEDURE Roger_Out(Name : IN OUT
OOD_Task_Manager_Pkg.Name_String_Type);


END OOD_Pkg;


------------------------------------------------------------------------
--| FileName   : Mission_Control_Pkg.ads
--| Author     : LT William D. Carroll, USN
--| Date       : June 2000
--| Project    : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: This package provides Mission Procedures
------------------------------------------------------------------------

PACKAGE Mission_Control_Pkg IS


------------------------------------------------------------------------
--| PROCEDURE   Transit_To_Location
------------------------------------------------------------------------
PROCEDURE Transit_To_Location(Latitude  : Float;
                              Longitude : Float);

------------------------------------------------------------------------
--| PROCEDURE   Hunt_For_Mines
------------------------------------------------------------------------
PROCEDURE Hunt_For_Mines;


END Mission_Control_Pkg;


------------------------------------------------------------------------
--| FileName   : Expert_Systems_Pkg.ads
--| Author     : LT William D. Carroll, USN
--| Date       : June 2000
--| Project    : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Expert Systems functions
------------------------------------------------------------------------
WITH OOD_Task_Manager_Pkg;


PACKAGE Expert_Systems_Pkg IS

------------------------------------------------------------------------
--| FUNCTION   Get_Course_To_Station
------------------------------------------------------------------------
FUNCTION Get_Course_To_Station(Latitude  : Float;
                               Longitude : Float) RETURN Integer;

------------------------------------------------------------------------
--| FUNCTION   Get_Mine_Hunt_Course
```

```
------------------------------------------------------------------------
FUNCTION Get_Mine_Hunt_Course(Latitude  : Float;
                              Longitude : Float) RETURN Integer;


END Expert_Systems_Pkg;


------------------------------------------------------------------------
--| Filename   : Wrap_AUV_C_Code_Pkg.ads
--| Author     : LT William D. Carroll, USN
--| Date       : 30 May 2000
--| Course     : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Provides for the interfacing to auv_c_functions.c
------------------------------------------------------------------------

WITH Interfaces;
USE Interfaces;
WITH Interfaces.C;
USE  Interfaces.C;
WITH Interfaces.C.Strings;
USE  Interfaces.C.Strings;


PACKAGE Wrap_AUV_C_Code_PKG IS


------------------------------------------------------------------------
--| PROCEDURE   Text_From_C_Function
------------------------------------------------------------------------
PROCEDURE Text_From_C_Function(A_String : chars_ptr);
   pragma Export(Convention => C,
                 Entity => Text_From_C_Function,
                 External_Name => "CStringBack");

------------------------------------------------------------------------
--| PROCEDURE   End_Text_From_C_Function
------------------------------------------------------------------------
PROCEDURE End_Text_From_C_Function(A_String : chars_ptr);
   pragma Export(Convention => C,
                 Entity => End_Text_From_C_Function,
                 External_Name => "EndCStringBack");

------------------------------------------------------------------------
--| PROCEDURE   Double_From_C_Function
------------------------------------------------------------------------
PROCEDURE Double_From_C_Function(C_Double : C.double);
   pragma Export(Convention => C,
                 Entity => Double_From_C_Function,
                 External_Name => "CDoubleBack");

------------------------------------------------------------------------
--| PROCEDURE Stop_Screw_Motors
------------------------------------------------------------------------
```

```
PROCEDURE Stop_Screw_Motors;

--------------------------------------------------------------------------
--| PROCEDURE Rudder_Angle
--------------------------------------------------------------------------
PROCEDURE Rudder_Angle(Angle : double);

--------------------------------------------------------------------------
--| PROCEDURE Planes_Angle
--------------------------------------------------------------------------
PROCEDURE Planes_Angle(BowAngle : double; SternAngle : double);

--------------------------------------------------------------------------
--| PROCEDURE Zero_Fins
--------------------------------------------------------------------------
PROCEDURE Zero_Fins;

--------------------------------------------------------------------------
--| PROCEDURE Abort_Mission
--------------------------------------------------------------------------
PROCEDURE Abort_Mission;

--------------------------------------------------------------------------
--| PROCEDURE Left_Screw_Speed_Control
--------------------------------------------------------------------------
PROCEDURE Left_Screw_Speed_Control(n_com : double);

--------------------------------------------------------------------------
--| PROCEDURE Right_Screw_Speed_Control
--------------------------------------------------------------------------
PROCEDURE Right_Screw_Speed_Control(n_com : Double);


END Wrap_AUV_C_Code_PKG;


-----------------------------------------------------------------------
--| FileName    : Utilities.ads
--| Author      : Michael J. Holden, modified by William D. Carroll
--| Date        : July 1999 - May 2000
--| Project     : Thesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Package Specification for Utilities.
-----------------------------------------------------------------------

WITH Ada.Text_IO;
WITH Ada.Calendar;

PACKAGE Utilities IS

--------------------------------------------------------------------------
--| PROCEDURE   Get_Current_Time
--------------------------------------------------------------------------
PROCEDURE Get_Current_Time(Hour     : OUT Integer;
                           Minute   : OUT Integer;
```

```
                              Seconds : OUT Float);

----------------------------------------------------------------------
--| PROCEDURE   Communicate
----------------------------------------------------------------------
PROCEDURE Communicate (Item: IN String);

--------------------------------------------------------------------
--| PROCEDURE Display_Message
--------------------------------------------------------------------
PROCEDURE Display_Message(Message_Text : IN String);

-----------------------------------------------------------------
--| PROCEDURE Print_Symbol
--| Post: Displays a symbol on the same line a number of times.
--|   Symbol is the character to be repeated
--|   HowMany is the number of times to repeat the character
-----------------------------------------------------------------
PROCEDURE Print_Symbol (Symbol  : IN Character;
                        HowMany : IN Natural);


END Utilities;


----------------------------------------------------------------------
--| FileName   : OOD_Task_Manager_Pkg.adb
--| Author     : LT William D. Carroll, USN
--| Date       : 09 NOV 1999 - June 2000
--| Project    : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: This Package receives the direction from the Stratigic
--|              level through the Receive_Orders procedure. The code
--|              for the NAV, ENG, & LOG tasks are located here.
----------------------------------------------------------------------

WITH Ada.Text_IO;
WITH Ada.Integer_Text_IO;
WITH Ada.Float_Text_IO;
WITH Utilities;
WITH Mission_Control_Pkg;
WITH Expert_Systems_Pkg;
WITH Wrap_AUV_C_Code_Pkg;
WITH OOD_Pkg;
use Ada;

PACKAGE BODY OOD_Task_Manager_Pkg IS



-- Task NAV, ENG, LOG, will start executing as soon as the project
--   program is started.
-- Each task will block on its ACCEPT until the entry is called.
-- The tasks will end when this program is no longer active.
```

```
----------------------------------------------------------------------
--| TASK NAV(Navigator)
----------------------------------------------------------------------
TASK BODY NAV IS

NavName : Name_String_Type := "Navigator ";
Course : Course_String_Type := "000";
Recomended_Course : Integer := 0;

BEGIN -- NAV

  Utilities.Communicate ("Navigator: ""Standing by""
");

  LOOP
    SELECT
      ACCEPT Taking_Action;
        Utilities.Communicate ("Navigator: ""Taking action""");
    OR
      ACCEPT Making_Report(Name   : IN Name_Type;
                           Report : Report_String_Type)DO
        CASE Name IS
          WHEN Navigator => NULL;
          WHEN Engineer =>
            Utilities.Communicate ("Navigator makes report to Engineer
");
            ENG.ENG_Aye(Name => NavName, EngOrder => None);
          WHEN OfficerOfTheDeck =>
            Utilities.Communicate ("Navigator makes report to OOD
");
            OOD_Pkg.Roger_Out(Name => NavName);
        END CASE;
      END Making_Report;
    OR
      ACCEPT NAV_Aye(Name      : IN OUT Name_String_Type;
                     NavOrder  : Order_Type;
                     Latitude  : Float := 0.0;
                     Longitude : Float := 0.0) DO
        CASE NavOrder IS
          WHEN None =>
            Utilities.Communicate ("NAV: ""Navigator Aye"": Ack. " &
Name & "         ");
          WHEN CourseToStation =>
            Utilities.Communicate ("NAV: ""Get Course Aye"": Ack. " &
Name & "        ");

            Recomended_Course :=
Expert_Systems_Pkg.Get_Course_To_Station(Latitude  => Latitude,

Longitude => Longitude);
            Course := Integer'Image(Recomended_Course);
            Utilities.Communicate ("Navigator Recommends Course of " &
Course & " to OOD     ");
            OOD_Pkg.Roger_Out(Name => NavName);
```

```
          WHEN MineHuntCourse =>
           Utilities.Communicate ("NAV: ""Mine Hunt Course Aye"": Ack.
" & Name & "  ");

           Recomended_Course :=
Expert_Systems_Pkg.Get_Mine_Hunt_Course(Latitude  => Latitude,

Longitude => Longitude);
           Course := Integer'Image(Recomended_Course);
           Utilities.Communicate ("Navigator Recomends Course of " &
Course & " to OOD      ");
           OOD_Pkg.Roger_Out(Name => NavName);

          WHEN GivePosition =>
           Utilities.Communicate ("NAV: ""Give Position Aye"": Ack. "
&  Name & "     ");
           Utilities.Communicate ("Current position:  36.70 Deg North
Latitude   ");
           Utilities.Communicate ("                  121.85 Deg West
Longitude   ");
          WHEN others => NULL;
        END CASE;
      END  NAV_Aye;
    OR
      TERMINATE;
    END SELECT;
  END LOOP;
END NAV;

-----------------------------------------------------------------------
--| TASK ENG(Engineering Officer)
-----------------------------------------------------------------------
TASK BODY ENG IS

EngName     : Name_String_Type := "Engineer   ";

BEGIN -- ENG

  Utilities.Communicate ("Engineer: ""Standing by""
");

  LOOP
    SELECT
      ACCEPT Taking_Action;
        Utilities.Communicate ("Engineer: ""Taking action""");
    OR
      ACCEPT Making_Report(Name : IN Name_Type) DO
        CASE Name IS
          WHEN Navigator =>
            Utilities.Communicate ("Engineer makes report to Navigator
");
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN Engineer =>  NULL;
          WHEN OfficerOfTheDeck =>
```

```
            Utilities.Communicate ("Engineer makes report to OOD
");
            OOD_Pkg.Roger_Out(Name => EngName);
          END CASE;
        END Making_Report;
      OR
        ACCEPT ENG_Aye(Name : IN OUT Name_String_Type;
                       EngOrder : Order_Type) DO
          CASE EngOrder IS
            WHEN None =>
              Utilities.Communicate ("ENG: ""Engineer Aye"": Ack. " &
Name & "           ");
            WHEN AllStop =>
              Utilities.Communicate ("ENG: ""All Engines Stop Aye"": Ack.
" & Name & "  ");
              Wrap_AUV_C_Code_PKG.Stop_Screw_Motors;
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN AllAhead =>
              Utilities.Communicate ("ENG: ""All Engines Ahead Aye"":
Ack. " & Name & " ");
              Wrap_AUV_C_Code_PKG.Left_Screw_Speed_Control(5.0);
              Wrap_AUV_C_Code_PKG.Right_Screw_Speed_Control(5.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN PortStop =>
              Utilities.Communicate ("ENG: ""Port Engine Stop Aye"": Ack.
" & Name & "  ");
              Wrap_AUV_C_Code_PKG.Left_Screw_Speed_Control(0.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN PortAhead =>
              Utilities.Communicate ("ENG: ""Port Engine Ahead Aye"":
Ack. " & Name & " ");
              Wrap_AUV_C_Code_PKG.Left_Screw_Speed_Control(5.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN PortBack =>
              Utilities.Communicate ("ENG: ""Port Engine Back Aye"": Ack.
" & Name & "  ");
              Wrap_AUV_C_Code_PKG.Left_Screw_Speed_Control(5.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN StbdStop =>
              Utilities.Communicate ("ENG: ""Starboard Engine Stop Aye"":
Ack. " & Name & "  ");
              Wrap_AUV_C_Code_PKG.Right_Screw_Speed_Control(0.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN StbdAhead =>
              Utilities.Communicate ("ENG: ""Starboard Engine Ahead
Aye"": Ack. " & Name );
              Wrap_AUV_C_Code_PKG.Right_Screw_Speed_Control(5.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN StbdBack =>
              Utilities.Communicate ("ENG: ""Starboard Engine Back Aye"":
Ack. " & Name & " ");
              Wrap_AUV_C_Code_PKG.Right_Screw_Speed_Control(5.0);
              NAV.NAV_Aye(Name => EngName, NavOrder => None);
            WHEN EmergencySurface =>
```

```
            Utilities.Communicate ("ENG: ""Emergency Surface Aye"":
Ack. " & Name & " ");
            Wrap_AUV_C_Code_PKG.Abort_Mission;
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
         WHEN RudderAmidship =>
            Utilities.Communicate ("ENG: ""Rudder Amidship Aye"": Ack.
" & Name & "   ");
            Wrap_AUV_C_Code_PKG.Rudder_Angle(0.0);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN RightRudder =>
            Utilities.Communicate ("ENG: ""Right Rudder Aye"": Ack. " &
Name & "      ");
            Wrap_AUV_C_Code_PKG.Rudder_Angle(0.4);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN LeftRudder =>
            Utilities.Communicate ("ENG: ""Left Rudder Aye"": Ack. " &
Name & "       ");
            Wrap_AUV_C_Code_PKG.Rudder_Angle(0.4);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN ZeroPlanes =>
            Utilities.Communicate ("ENG: ""Zero Planes Aye"": Ack " &
Name & "        ");
            Wrap_AUV_C_Code_PKG.Planes_Angle(0.0, 0.0);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN UpPlanes =>
            Utilities.Communicate ("ENG: ""Up Planes Aye"": Ack " &
Name );
            Wrap_AUV_C_Code_PKG.Planes_Angle(0.4, 0.4);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN DownPlanes =>
            Utilities.Communicate ("ENG: ""Down Planes Aye"": Ack. " &
Name & "       ");
            Wrap_AUV_C_Code_PKG.Planes_Angle(0.4, 0.4);
            NAV.NAV_Aye(Name => EngName, NavOrder => None);
          WHEN others => NULL;
        END CASE;
      END  ENG_Aye;
    OR
      TERMINATE;
    END SELECT;
  END LOOP;
END ENG;

----------------------------------------------------------------------
--| TASK LOG(Deck Log Entries)
----------------------------------------------------------------------
TASK BODY LOG IS

DeckLogName : Name_String_Type := "Deck Log  ";
Deck_Log :  Ada.Text_IO.File_Type;
FileName : String := "DeckLog.txt";

BEGIN -- LOG

Ada.Text_IO.Create(File => Deck_Log,
```

```
                    Mode => Ada.Text_IO.Out_File,
                    Name => FileName);

Utilities.Display_Message(Message_Text => "Deck Log is open.");

  LOOP
    SELECT
      ACCEPT Log_It(Item    : IN String;
                    Hour    : IN Integer;
                    Minute  : IN Integer;
                    Seconds: IN Float) DO

        Ada.Text_IO.Put(File => Deck_Log,
                        Item => Item & "At: ");
        Ada.Integer_Text_IO.Put(File => Deck_Log,Item => Hour,   Width
=> 2);
        Ada.Text_IO.Put(File => Deck_Log,Item => ":");
        Ada.Integer_Text_IO.Put(File => Deck_Log,Item => Minute, Width
=> 2);
        Ada.Text_IO.Put(File => Deck_Log,Item => ":");
        Ada.Float_Text_IO.Put(File => Deck_Log,
                          Item => Seconds, Fore => 2, Aft =>10, Exp =>
0);
        Ada.Text_IO.Put_Line(File => Deck_Log,
                             Item => " ");
        END Log_It;
    OR
      ACCEPT Close_Log;
        Ada.Text_IO.Close(File => Deck_Log);
        Utilities.Display_Message(Message_Text => "Deck Log is
closed.");
    OR
      TERMINATE;
    END SELECT;
  END LOOP;
END LOG;

------------------------------------------------------------------------
-----------------------| END of TASKS |---------------------------------
------------------------------------------------------------------------

------------------------------------------------------------------------
--| PROCEDURE Mission_Planner
------------------------------------------------------------------------
PROCEDURE Mission_Planner(Orders    : Order_Type;
                         Latitude  : Float;
                         Longitude : Float) IS

BEGIN
  CASE Orders IS
    WHEN None =>
      Utilities.Communicate ("No Mission received");
    WHEN HuntMines =>
      Utilities.Communicate ("Commence Mine Hunt Mission
");
```

```
      Mission_Control_Pkg.Transit_To_Location(Latitude, Longitude);
      Mission_Control_Pkg.Hunt_For_Mines;
    WHEN others => NULL;
  END CASE;

END Mission_Planner;

------------------------------------------------------------------------
--| PROCEDURE Officer_Of_The_Deck
------------------------------------------------------------------------
PROCEDURE Officer_Of_The_Deck (Orders    : Order_Type;
                               Latitude  : Float;
                               Longitude : Float) IS


BEGIN -- Officer_Of_The_Deck

  Utilities.Communicate ("OOD: ""I have the Deck""
");

  Mission_Planner(Orders, Latitude, Longitude);

  Utilities.Communicate ("Surface the AUV for recovery
");
  OOD_Pkg.Give_Order(Name => Engineer, Order => EmergencySurface);

  LOG.Close_Log;

END Officer_Of_The_Deck;

------------------------------------------------------------------------
--| PROCEDURE Receive_Orders
------------------------------------------------------------------------
PROCEDURE Receive_Orders(Orders    : Order_Type;
                         Latitude  : Float;
                         Longitude : Float) IS

BEGIN
  Officer_Of_The_Deck(Orders, Latitude, Longitude);
END Receive_Orders;


END OOD_Task_Manager_Pkg;


------------------------------------------------------------------------
--| FileName    : OOD_Pkg.adb
--| Author      : LT William D. Carroll, USN
--| Date        : June 2000
--| Project     : Thesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: OOD Procedures
------------------------------------------------------------------------

WITH Utilities;
```

```
PACKAGE BODY OOD_Pkg IS

OodName : OOD_Task_Manager_Pkg.Name_String_Type := "OOD           ";

---------------------------------------------------------------------
--| PROCEDURE Give_Order
---------------------------------------------------------------------
PROCEDURE Give_Order(Name : IN OOD_Task_Manager_Pkg.Name_Type;
                     Order: OOD_Task_Manager_Pkg.Order_Type;
                     Latitude  : Float := 0.0;
                     Longitude : Float := 0.0) IS
   BEGIN
      CASE Name IS
         WHEN OOD_Task_Manager_Pkg.Navigator =>
              Utilities.Communicate ("OOD gives order to Navigator
");
              OOD_Task_Manager_Pkg.NAV.NAV_Aye(Name      => OodName,
                                               NavOrder  => Order,
                                               Latitude  => Latitude,
                                               Longitude => Longitude);
         WHEN OOD_Task_Manager_Pkg.Engineer =>
              Utilities.Communicate ("OOD gives order to Engineer
");
              OOD_Task_Manager_Pkg.ENG.ENG_Aye(Name => OodName,
EngOrder => Order);
         WHEN OOD_Task_Manager_Pkg.OfficerOfTheDeck => NULL;
      END CASE;
   END Give_Order;

---------------------------------------------------------------------
--| PROCEDURE Taking_Action
---------------------------------------------------------------------
PROCEDURE Taking_Action IS

   BEGIN
      Utilities.Communicate ("OOD: ""Taking action""
");
      DELAY 0.1;      -- lets another task have the CPU
   END Taking_Action;

---------------------------------------------------------------------
--| PROCEDURE Roger_Out
---------------------------------------------------------------------
PROCEDURE Roger_Out(Name : IN OUT
OOD_Task_Manager_Pkg.Name_String_Type) IS

   BEGIN
      Utilities.Communicate ("OOD: ""Roger Out"": Acknowledge " & Name
& "      ");
   END  Roger_Out;


END OOD_Pkg;
```

```
---------------------------------------------------------------------
--| FileName    : Mission_Control_Pkg.adb
--| Author      : LT William D. Carroll, USN
--| Date        : June 2000
--| Project     : Thesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: This package provides Mission Procedures
---------------------------------------------------------------------
WITH OOD_Task_Manager_Pkg;
USE  OOD_Task_Manager_Pkg;
WITH OOD_Pkg;
WITH Utilities;

PACKAGE BODY Mission_Control_Pkg IS


---------------------------------------------------------------------
--| PROCEDURE   Transit_To_Location
---------------------------------------------------------------------
PROCEDURE Transit_To_Location(Latitude  : Float;
                              Longitude : Float) IS
BEGIN
  Utilities.Communicate ("Commence Transit_To_Location Mission
");
  OOD_Pkg.Give_Order(Name => Navigator, Order => CourseToStation,
                  Latitude => Latitude, Longitude => Longitude);
  OOD_Pkg.Give_Order(Name => Engineer, Order => AllAhead);
  OOD_Pkg.Give_Order(Name => Engineer, Order => RightRudder);
  OOD_Pkg.Give_Order(Name => Engineer, Order => RudderAmidship);
  OOD_Pkg.Give_Order(Name => Engineer, Order => DownPlanes);
  OOD_Pkg.Give_Order(Name => Engineer, Order => ZeroPlanes);
  OOD_Pkg.Give_Order(Name => Navigator, Order => GivePosition);
  OOD_Pkg.Give_Order(Name => Engineer, Order => AllStop);
  Utilities.Communicate ("Transit_To_Location Mission Complete
");

END Transit_To_Location;

---------------------------------------------------------------------
--| PROCEDURE   Hunt_For_Mines
---------------------------------------------------------------------
PROCEDURE Hunt_For_Mines IS

BEGIN
  Utilities.Communicate ("Commence Hunt_For_Mines Mission
");
  OOD_Pkg.Give_Order(Name => Navigator, Order => MineHuntCourse);
  OOD_Pkg.Give_Order(Name => Engineer, Order => AllAhead);
  OOD_Pkg.Give_Order(Name => Engineer, Order => RightRudder);
  OOD_Pkg.Give_Order(Name => Engineer, Order => RudderAmidship);
  Utilities.Communicate ("Hunt_For_Mines Mission Complete
");

END Hunt_For_Mines;
```

```
END Mission_Control_Pkg;

------------------------------------------------------------------------
--| FileName   : Expert_Systems_Pkg.adb
--| Author     : LT William D. Carroll, USN
--| Date       : June 2000
--| Project    : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Expert Systems functions
------------------------------------------------------------------------

PACKAGE BODY Expert_Systems_Pkg IS

------------------------------------------------------------------------
--| FUNCTION   Get_Course_To_Station
------------------------------------------------------------------------
FUNCTION Get_Course_To_Station(Latitude  : Float;
                               Longitude : Float) RETURN Integer IS
 Recomended_Course : Integer := 90;
BEGIN
  RETURN Recomended_Course;
END Get_Course_To_Station;

------------------------------------------------------------------------
--| FUNCTION   Get_Mine_Hunt_Course
------------------------------------------------------------------------
FUNCTION Get_Mine_Hunt_Course(Latitude  : Float;
                              Longitude : Float) RETURN Integer IS
 Recomended_Course : Integer := 95;
BEGIN
  RETURN Recomended_Course;
END Get_Mine_Hunt_Course;

END Expert_Systems_Pkg;

------------------------------------------------------------------------
--| Filename   : Wrap_AUV_C_Code_Pkg.adb
--| Author     : LT William D. Carroll, USN
--| Date       : 30 May 2000
--| Course     : Thesis
--| Compiler   : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Provides for the interfacing to auv_c_functions.c
------------------------------------------------------------------------

WITH Ada.Text_IO;
WITH Ada.Integer_Text_IO;
WITH Ada.Float_Text_IO;
WITH Utilities;
```

```
PACKAGE BODY Wrap_AUV_C_Code_Pkg IS


-----------------------------------------------------------------------
--| FUNCTION   Value_Without_Exception
--|            Lovelace Ada tutorial - David A. Wheeler
-----------------------------------------------------------------------
FUNCTION Value_Without_Exception(S : chars_ptr) RETURN String IS
   pragma Inline(Value_Without_Exception);

   -- Translate S from a C-style char* into an Ada String.
   -- If S is Null_Ptr, return "", does raise an exception.
   BEGIN
      IF S = Null_Ptr THEN RETURN "Null Ptr";
         ELSE RETURN Value(S);
      END IF;
   END Value_Without_Exception;

-----------------------------------------------------------------------
--| PROCEDURE   Text_From_C_Function
-----------------------------------------------------------------------
PROCEDURE Text_From_C_Function(A_String : chars_ptr) IS

   -- Convert the sent C chars_ptr to an Ada String value without
   --   getting an exception if the returned is a Null_ptr.
   Report : String := Value_Without_Exception(A_String);

   BEGIN
      Ada.Text_IO.Put(Item => Report & " ");
   END Text_From_C_Function;

-----------------------------------------------------------------------
--| PROCEDURE   End_Text_From_C_Function
-----------------------------------------------------------------------
PROCEDURE End_Text_From_C_Function(A_String : chars_ptr) IS

   -- Convert the sent C chars_ptr to an Ada String value without
getting
   --   an exception if the returned is a Null_ptr.
   Report : String := Value_Without_Exception(A_String);

   BEGIN
      Ada.Text_IO.Put_Line(Report);
   END End_Text_From_C_Function;

-----------------------------------------------------------------------
--| PROCEDURE   Double_From_C_Function
-----------------------------------------------------------------------
PROCEDURE Double_From_C_Function(C_Double : C.double) IS

  -- Cast to a Ada Float value for manipulation within Ada
  Ada_Float :  Float := Float(C_Double);

  BEGIN
```

```
    -- Output to the screen followed by a space
    Ada.Float_Text_IO.Put(Ada_Float, 2, 4, 0);
    Ada.Text_IO.Put(Item => " ");

  END Double_From_C_Function;

------------------------------------------------------------------------
--| PROCEDURE   Stop_Screw_Motors
--| The function is called "StopScrewMotors" in C.
--| The function is found in the object file "auv_c_functions.o"
------------------------------------------------------------------------
   procedure StopScrewMotors;
   pragma Import(Convention => C,
                 Entity => StopScrewMotors,
                 External_Name => "StopScrewMotors");
-- Wrapper function
PROCEDURE Stop_Screw_Motors IS
   BEGIN
      StopScrewMotors;
      --Ada.Text_IO.Put_Line(Item => "Inside - Stop_Screw_Motors");
   END Stop_Screw_Motors;


------------------------------------------------------------------------
--| PROCEDURE   Rudder_Angle
--| The function is called "Rudder" in C.
--| The function is found in the object file "auv_c_functions.o"
------------------------------------------------------------------------
   procedure Rudder(Angle : C.double);
   pragma Import(Convention => C,
                 Entity => Rudder,
                 External_Name => "Rudder");
-- Wrapper function
PROCEDURE Rudder_Angle(Angle : double) IS
   BEGIN
      Rudder(Angle);
   END Rudder_Angle;

------------------------------------------------------------------------
--| PROCEDURE   Planes_Angle
--| The function is called "Planes" in C.
--| The function is found in the object file "auv_c_functions.o"
------------------------------------------------------------------------
   procedure Planes(BowAngle : C.double; SternAngle: C.double);
   pragma Import(Convention => C,
                 Entity => Planes,
                 External_Name => "Planes");
-- Wrapper function
PROCEDURE Planes_Angle(BowAngle : double; SternAngle : double)IS
   BEGIN
      Planes(BowAngle,SternAngle);
   END Planes_Angle;

------------------------------------------------------------------------
```

```
--| PROCEDURE   Zero_Fins
--| The function is called "ZeroFins" in C.
--| The function is found in the object file "auv_c_functions.o"
----------------------------------------------------------------------
   procedure ZeroFins;
   pragma Import(Convention => C,
                 Entity => ZeroFins,
                 External_Name => "ZeroFins");
-- Wrapper function
PROCEDURE Zero_Fins IS
    BEGIN
      ZeroFins;
    END Zero_Fins;

----------------------------------------------------------------------
--| PROCEDURE Abort_Mission
--| The function is called "AbortMission" in C.
--| The function is found in the object file "auv_c_functions.o"
----------------------------------------------------------------------
   procedure AbortMission;
   pragma Import(Convention => C,
                 Entity => AbortMission,
                 External_Name => "Abort");
-- Wrapper function
PROCEDURE Abort_Mission IS
    BEGIN
      AbortMission;
      --Ada.Text_IO.Put_Line(Item => "Inside - Abort_Mission");
    END Abort_Mission;

----------------------------------------------------------------------
--| PROCEDURE Left_Screw_Speed_Control
--| The function is called "LeftScrewSpeedControl" in C.
--| This function is found in the object file "auv_c_functions.o"
----------------------------------------------------------------------
   procedure LeftScrewSpeedControl(n_com : C.double);
   pragma Import(Convention => C,
                 Entity => LeftScrewSpeedControl,
                 External_Name => "LeftScrewSpeedControl");
-- Wrapper function
PROCEDURE Left_Screw_Speed_Control(n_com : double) IS
    BEGIN
      LeftScrewSpeedControl(n_com);
    END Left_Screw_Speed_Control;

----------------------------------------------------------------------
--| PROCEDURE Right_Screw_Speed_Control
--| The function is called "RightScrewSpeedControl" in C.
--| This function is found in the object file "auv_c_functions.o"
----------------------------------------------------------------------
   procedure RightScrewSpeedControl(n_com : C.double);
   pragma Import(Convention => C,
                 Entity => RightScrewSpeedControl,
                 External_Name => "RightScrewSpeedControl");
-- Wrapper function
```

```
PROCEDURE Right_Screw_Speed_Control(n_com : double) IS
    BEGIN
      RightScrewSpeedControl(n_com);
    END Right_Screw_Speed_Control;

END Wrap_AUV_C_Code_Pkg;


------------------------------------------------------------------------
--| FileName    : Utilities.adb
--| Author      : Michael J. Holden, modified by William D. Carroll
--| Date        : July 1999 - May 2000
--| Project     : THesis
--| Compiler    : Aonix ObjectAda 7.1.2.205 (Professional)
--| Description: Package Body for Utilities.
------------------------------------------------------------------------

WITH Ada.Text_IO;
WITH Ada.Float_Text_IO;
WITH Ada.Integer_Text_IO;
WITH Ada.IO_Exceptions;
WITH OOD_Task_Manager_Pkg;

PACKAGE BODY Utilities IS

  PROCEDURE Get_Current_Time(Hour    : OUT Integer;
                             Minute  : OUT Integer;
                             Seconds : OUT Float) IS
      Now           : Ada.Calendar.Time:= Ada.Calendar.Clock;
     -- Hour          : Integer;
    --  Minute        : Integer;
    --  Seconds       : Float;
      Second        : Ada.Calendar.Day_Duration;
      FloatSecond   : Float;
      FloatMinute   : Float;
      FloatHour     : Float;

   BEGIN -- Display_Current_Time
      Second := Ada.Calendar.Seconds(Now);
      FloatSecond := Float(Second);
      FloatMinute := FloatSecond/60.0;
      FloatHour   := FloatMinute/60.0;
      Hour   := Integer(FloatHour - 0.5);
      Minute := Integer( ( ( FloatHour - Float(Hour) ) * 60.0) - 0.5 );
      Seconds := (FloatSecond-((Float(Minute) * 60.0)+(Float(Hour) *
3600.0)));
      Ada.Integer_Text_IO.Put(Item => Hour,   Width => 2);
      Ada.Text_IO.Put(Item => ":");
      Ada.Integer_Text_IO.Put(Item => Minute, Width => 2);
      Ada.Text_IO.Put(Item => ":");
      --Ada.Integer_Text_IO.Put(Item => Seconds, Width => 2);
      Ada.Float_Text_IO.Put(Item => Seconds, Fore => 2, Aft =>4, Exp =>
0);

--      Ada.Text_IO.New_Line;
```

```
   END Get_Current_Time;

----------------------------------------------------------------------
--| PROCEDURE Communicate
----------------------------------------------------------------------
PROCEDURE Communicate (Item: String) IS

  Log_String : String := Item;
  Hour       : Integer;
  Minute     : Integer;
  Seconds    : Float;

BEGIN -- Communicate
  Ada.Text_IO.New_Line;
  Ada.Text_IO.Put(Item => Item & " At: ");
  Get_Current_Time(Hour, Minute, Seconds);
  Ada.Text_IO.New_Line;
  OOD_Task_Manager_Pkg.LOG.Log_It(Item    => Log_String,
                                  Hour    => Hour,
                                  Minute  => Minute ,
                                  Seconds => Seconds);
END Communicate;

----------------------------------------------------------------------
--| PROCEDURE  Display_Message
----------------------------------------------------------------------

   PROCEDURE Display_Message(Message_Text : IN String) IS

   LineWidth : Natural := 70;

   BEGIN -- Display_Message
      Ada.Text_IO.New_Line(Spacing =>2);
      Print_Symbol (Symbol => '-',
                    HowMany => (LineWidth-Message_Text'Length)/2 - 1);
      IF Message_Text = "" OR ELSE Message_Text = " " THEN
         Ada.Text_IO.Put(Item => '-' & '-');
      ELSE
         Ada.Text_IO.Put(Item => " " & Message_Text & " ");
      END IF;
      Print_Symbol (Symbol => '-',
                    HowMany => (LineWidth-Message_Text'Length)/2 - 1);
      Ada.Text_IO.New_Line(spacing =>2);


END Display_Message;


-------------------------------------------------------------------
--| PROCEDURE Print_Symbol
--| Pre : None
--| Post: Displays a symbol on the same line a number of times.
--| Exceptions Raised: None.
--| Parameters:
--|   Symbol is the character to be repeated
```

```
--|   HowMany is the number of times to repeat the character
--| Complexity: O( )
-------------------------------------------------------------------

      PROCEDURE Print_Symbol (Symbol  : IN Character;
                              HowMany : IN Natural) IS
      BEGIN -- Print_Symbol
      FOR Count IN 1..HowMany LOOP
          Ada.Text_IO.Put(Item => Symbol);
        END LOOP;
    END Print_Symbol;


END Utilities;
```

3. EXECUTION LEVEL CODE

```
// ---------------------------------------------------------------------
//  FileName    : auv_c_functions.c
//  Author      : Dr Dave Marco - NPS, modified by William D. Carroll
//  Date        : May 2000
//  Project     : Thesis
//  Compiler    : GNAT Version 3.12p, used gcc for .o file output
//  Description: Modified from "Execf.c" C code written by Dr. D. Marco
//               for the Naval Postgraduate School A.R.I.E.S.
//               Autonomous Underwater Vehicle (AUV). Changes made by
//               LT are denoted by //** in the right margin. Code
//               by Dr. Marco that is not required for this
//               demonstration has been either commented out but
//               retained for clarity or has been deleted.
// ---------------------------------------------------------------------

#include <stdio.h>
#include <string.h>

#define LEFT_BOW_PLANE          1
#define RIGHT_BOW_PLANE         2
#define LEFT_STERN_PLANE        3
#define RIGHT_STERN_PLANE       4
#define BOW_RUDDER              5
#define STERN_RUDDER            6

//---------------------------------------------------------------------
//  NAME           StopScrewMotors
//---------------------------------------------------------------------
StopScrewMotors()
{
   CStringBack("Screw Motors Stoped");                              //**
   EndCStringBack("");                                              //**
   //RubyDac(0,0.0); /* Left Screw  */
```

```
   //RubyDac(1,0.0); /* Right Screw */
} // end StopScrewMotors()

//------------------------------------------------------------------------
//  NAME          ScrewMotor
//------------------------------------------------------------------------
ScrewMotor(int Motor, double ControlVolt)
{
   char*  retchar  = "No Action ScrewMotor";

   if (Motor == 0){                                                    //**
      CStringBack("From ScrewMotor: Left Screw at:");                  //**
      CDoubleBack(ControlVolt);                                        //**
      EndCStringBack("VDC");                                           //**
   }                                                                   //**
   if (Motor == 1){                                                    //**
      CStringBack("From ScrewMotor: Right Screw at:");                 //**
      CDoubleBack(ControlVolt);                                        //**
      EndCStringBack("VDC");                                           //**
   }

   /* Motor = Motor number,  0  Left Screw   Ch = 0,  Pin 2 and BO 2
                             1  Right Screw  Ch = 1,  Pin 4 and BO 4

      Volt = Control Voltage Sent to Servo Amplifier +-5 VDC
   */

}//end ScrewMotor()

//------------------------------------------------------------------------
//  NAME          ControlSurface
//------------------------------------------------------------------------
void ControlSurface(int Surface, double Angle)
{
  /*  This function sends the desired ANGLE (radians) to the specified
      control SURFACE */

  switch(Surface)
  {
     case 1:
        CStringBack("Left Bow Plane set to:");                         //**
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     case 2:
        CStringBack("Right Bow Plane set to:");                        //**
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     case 3:
        CStringBack("Left Stern Plane set to:");                       //**
```

```
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     case 4:
        CStringBack("Right Stern Plane set to:");                      //**
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     case 5:
        CStringBack("Bow Rudder set to:");                             //**
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     case 6:        /* This Uses the Second 9513 Chip */
        CStringBack("Stern Rudder set to:");                           //**
        CDoubleBack(Angle);                                            //**
        EndCStringBack("radians");                                     //**
        // code deleted
        break;

     default:
        CStringBack("No Action to ControlSurface");                    //**
        EndCStringBack("");                                            //**
        //printf("Invalid surface code\n");
        break;
   }
} // end ControlSurface()

//-----------------------------------------------------------------------
//  NAME          Rudder
//-----------------------------------------------------------------------
Rudder(double Angle)
{
  /* Send Angular Deflection (RADIANS) to Rudders.
     Convention: (+) Angle Right-Hand Rule about z-axis */

   ControlSurface(BOW_RUDDER,-Angle);
   ControlSurface(STERN_RUDDER,Angle);

} /* Rudder */

//-----------------------------------------------------------------------
//  NAME          Planes
//-----------------------------------------------------------------------
Planes(double BowAngle, double SternAngle)
{
  /* Send Angular Deflection (RADIANS) to Planes.
     Convention: (+) angle Right-Hand Rule about y-axis */
```

```
   ControlSurface(LEFT_BOW_PLANE,-BowAngle);
   ControlSurface(RIGHT_BOW_PLANE,BowAngle);
   ControlSurface(LEFT_STERN_PLANE,SternAngle);
   ControlSurface(RIGHT_STERN_PLANE,-SternAngle);

} /* Planes */

//----------------------------------------------------------------------
//  NAME           ZeroFins
//----------------------------------------------------------------------
ZeroFins()
{
   CStringBack("Fins at zero");                                      //**
   EndCStringBack("");                                               //**

   Rudder(0.0);
   Planes(0.0,0.0);
}

//----------------------------------------------------------------------
//  NAME           Abort
//----------------------------------------------------------------------
Abort()
{
   CStringBack("*** Inside Abort *** EMERGENCY SURFACE!!!!!");     //**
   EndCStringBack("");                                               //**
   //printf("Inside Abort\n");
   Rudder(-0.4);
   Planes(0.4,-0.4);
   ScrewMotor(0,5.0);
   ScrewMotor(1,5.0);
}

//----------------------------------------------------------------------
//  NAME           LeftScrewSpeedControl
//----------------------------------------------------------------------
LeftScrewSpeedControl(double n_com)
{
   double Limit;      // parameter required for the algorithm in the
   double e_n,v_spc;  //   original code.

   //code deleted, v_spc is not made equal to n_com in the original
code.
   v_spc = n_com;

   ScrewMotor(0,v_spc);
}

//----------------------------------------------------------------------
//  NAME           RightScrewSpeedControl
//----------------------------------------------------------------------
RightScrewSpeedControl(double n_com)
{
   double Limit;      // parameter required for the algorithm in the
   double e_n,v_spc;  //   original code.
```

```
   //code deleted, v_spc is not made equal to n_com in the original
code.
   v_spc = n_com;

   ScrewMotor(1,v_spc);
}
```

APPENDIX B. OUTPUT

1. OUTPUT

.

```
Navigator: "Standing by"                          At:  1:51:28.0771

Engineer: "Standing by"                           At:  1:51:28.0771

OOD: "I have the Deck"                            At:  1:51:28.0771

------------------------- Deck Log is open. -------------------------

Commence Mine Hunt Mission                        At:  1:51:28.0869

Commence Transit_To_Location Mission              At:  1:51:28.0869

OOD gives order to Navigator                      At:  1:51:28.0869

NAV: "Get Course Aye": Ack. OOD                   At:  1:51:28.0869

Navigator Recommends Course of  90 to OOD         At:  1:51:28.0869

OOD: "Roger Out": Acknowledge Navigator           At:  1:51:28.0972

OOD gives order to Engineer                       At:  1:51:28.0972

ENG: "All Engines Ahead Aye": Ack. OOD            At:  1:51:28.0972
From ScrewMotor: Left Screw at:   5.0000 VDC
From ScrewMotor: Right Screw at:  5.0000 VDC

NAV: "Navigator Aye": Ack. Engineer               At:  1:51:28.0972

OOD gives order to Engineer                       At:  1:51:28.0972

ENG: "Right Rudder Aye": Ack. OOD                 At:  1:51:28.0972
Bow Rudder set to: -0.4000 radians
Stern Rudder set to:  0.4000 radians

NAV: "Navigator Aye": Ack. Engineer               At:  1:51:28.1069

OOD gives order to Engineer                       At:  1:51:28.1069

ENG: "Rudder Amidship Aye": Ack. OOD              At:  1:51:28.1069
Bow Rudder set to:  0.0000 radians
Stern Rudder set to:  0.0000 radians

NAV: "Navigator Aye": Ack. Engineer               At:  1:51:28.1069

OOD gives order to Engineer                       At:  1:51:28.1069
```

```
ENG: "Down Planes Aye": Ack. OOD                At:  1:51:28.1069
Left Bow Plane set to: -0.4000 radians
Right Bow Plane set to:  0.4000 radians
Left Stern Plane set to:  0.4000 radians
Right Stern Plane set to: -0.4000 radians

NAV: "Navigator Aye": Ack. Engineer             At:  1:51:28.1968

OOD gives order to Engineer                     At:  1:51:28.2368

ENG: "Zero Planes Aye": Ack OOD                 At:  1:51:28.2671
Left Bow Plane set to:  0.0000 radians
Right Bow Plane set to:  0.0000 radians
Left Stern Plane set to:  0.0000 radians
Right Stern Plane set to:  0.0000 radians

NAV: "Navigator Aye": Ack. Engineer             At:  1:51:28.3579

OOD gives order to Navigator                    At:  1:51:28.3877

NAV: "Give Position Aye": Ack. OOD              At:  1:51:28.4180

Current position:  36.70 Deg North Latitude     At:  1:51:28.4482

                  121.85 Deg West Longitude     At:  1:51:28.4780

OOD gives order to Engineer                     At:  1:51:28.5078

ENG: "All Engines Stop Aye": Ack. OOD           At:  1:51:28.5381
Screw Motors Stoped

NAV: "Navigator Aye": Ack. Engineer             At:  1:51:28.5781

Transit_To_Location Mission Complete            At:  1:51:28.6079

Commence Hunt_For_Mines Mission                 At:  1:51:28.6377

OOD gives order to Navigator                    At:  1:51:28.6680

NAV: "Mine Hunt Course Aye": Ack. OOD           At:  1:51:28.6982

Navigator Recomends Course of  95 to OOD        At:  1:51:28.7280

OOD: "Roger Out": Acknowledge Navigator         At:  6:19:32.6055

OOD gives order to Engineer                     At:  6:19:32.6465

ENG: "All Engines Ahead Aye": Ack. OOD          At:  6:19:32.6758
From ScrewMotor: Left Screw at:   5.0000 VDC
From ScrewMotor: Right Screw at:  5.0000 VDC

NAV: "Navigator Aye": Ack. Engineer             At:  6:19:32.7363

OOD gives order to Engineer                     At:  6:19:32.7656
```

```
ENG: "Right Rudder Aye": Ack. OOD            At:  6:19:32.7969
Bow Rudder set to: -0.4000 radians
Stern Rudder set to:  0.4000 radians

NAV: "Navigator Aye": Ack. Engineer         At:  6:19:32.8555

OOD gives order to Engineer                 At:  6:19:32.8867

ENG: "Rudder Amidship Aye": Ack. OOD        At:  6:19:32.9160
Bow Rudder set to:  0.0000 radians
Stern Rudder set to:  0.0000 radians

NAV: "Navigator Aye": Ack. Engineer         At:  6:19:32.9766

Hunt_For_Mines Mission Complete             At:  6:19:33.0059

Surface the AUV for recovery                At:  6:19:33.0352

OOD gives order to Engineer                 At:  6:19:33.0664

ENG: "Emergency Surface Aye": Ack. OOD      At:  6:19:33.0957
*** Inside Abort *** EMERGENCY SURFACE!!!!!
Bow Rudder set to:  0.4000 radians
Stern Rudder set to: -0.4000 radians
Left Bow Plane set to: -0.4000 radians
Right Bow Plane set to:  0.4000 radians
Left Stern Plane set to: -0.4000 radians
Right Stern Plane set to:  0.4000 radians
From ScrewMotor: Left Screw at:   5.0000 VDC
From ScrewMotor: Right Screw at:  5.0000 VDC

NAV: "Navigator Aye": Ack. Engineer         At:  6:19:33.2559


------------------------ Deck Log is closed. ------------------------
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. *Autonomous Robot Vehicles*, I. J. Cox and G.T. Wilfong, eds., Spring-Verlang, New York, 1990.

2. Crute, Danial A., *Naval Mine Warfare Vision 2010 A View Toward the Future*, [http://www.ncsc.navy.mil/CCS/papers/vision.htm]. May 2000.

3. Holden, Michael J., *Ada Implementation of Concurrent Execution for Multiple Tasks in the Strategic and Tactical Levels of the Rational Behavior Model for the NPS AUV*, Masters Thesis, Naval Postgraduate School, Monterey, CA, September 1995.

4. Krechmer, K. "Interface APIs for Wide Area Networks." *Business Communications Review 22,* pp.72-4, November 1992.

5. Byrns, R.B., *The Rational Behavior Model: A multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*, PH.D. Dissertation, Naval Postgraduate School, Monterey, CA, March 1993.

6. Kwak, Se-Hung, Thornton, F.P.B., Jr., "A concurrent, object-oriented implementation for the tactical level of the rational behavior model software architecture for UUV control," Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology, AUV '94, pp. 54-60, 1994.,

7. Leonhardt, Bradley J., *Mission Planning and Mission Control Software for the Phoenix Autonomous Underwater Vehicle (AUV): Implementation and Experimental Study*, Masters Thesis, Naval Postgraduate School, Monterey, CA, March 1996.

8. Ada Reference Manual, International Standard ANSI/ISO/IEC 8652:1995(E), January 1995.

9. Interview between Dr. Dave Marco, Dept. of Mechanical Engineering, Naval Postgraduate School, Monterey California, and the author, 17 May 2000.

10. "Unmanned Underwater Vehicle (UUV) Program Plan." [http://www.contracts.hq.navsea.navy.mil/pms403/lmrs/uuvprogu.doc]. Aug 1996.

11. Navy Expeditionary Warfare Division (N85), Web Site. Mine Warfare (N852), "UUVs Unmanned Underwater Vehicles." [http://www.exwar.org/What'snew/uuvs.htm]. May 2000.

12. Navy Expeditionary Warfare Division (N85), Web Site. Mine Warfare (N852), "U.S. Naval Mine Warfare Plan." [http://www.exwar.org/What'snew/mwp/contents.htm]. May 2000.

13. IEEE standard glossary of software engineering terminology, IEEE Std 610.12-1990, 10 Dec 1990.

14. Cohen, Norman H., *Ada as a Second Language*, Second Edition, 979-998. McGraw-Hill Companies, Inc.,1996.

15. Ludwig, Peter M., *Formation Control for Multi-Vehicle Robotic Minesweeping*, Masters Thesis, Naval Postgraduate School, Monterey, CA, June 2000.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr. Teresa McMullen 1
   Office of Naval Research
   800 N. Quincy St. Tower 1
   Arlington, VA 22217-5660

4. Dr. Dan Boger 1
   Chairman, Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5118

5. Dr. Man-Tak Shing 2
   Computer Science Department, Code CS/Sh
   Naval Postgraduate School
   Monterey, CA 93943-5118

6. Dr. Dave Marco 1
   Mechanical Engineering Department, Code ME/Hy
   Naval Postgraduate School
   Monterey, CA 93943-5146

7. CDR Michael J. Holden, USN(Ret) 2
   1004 P.G. Lane #7
   Pacific Grove, CA 93950

8. LT William D. Carroll, USN 3
   PCS 476 Box 1213
   FPO AP 96322-1213