Copy No._____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Scenario and Tactics Development for OSPREY 4.3

S. Webb and J. Nickerson
MacDonald Dettwiler and Associates Ltd.

MacDonald Dettwiler and Associates Ltd.
Suite 60, 1000 Windmill Road
Dartmouth, NS
B3B 1L7

Project Manager: M.G. Hazen

Contract Number: W7707-04-2684

Contract Scientific Authority: M.G. Hazen, 902-426-3100 ext 176

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2005-196
September 2005

Canada

This page intentionally left blank.

Scenario and Tactics Development for OSPREY 4.3

S. Webb
J. Nickerson
MACDONALD DETTWILER AND ASSOCIATES LTD.

MACDONALD DETTWILER AND ASSOCIATES LTD.
Suite 60, 1000 Windmill Rd.
Dartmouth, NS B3B 1L7

Project Manager: M.G. Hazen
Contract number: W7707-04-2684
Contract Scientific Authority: M.G. Hazen, 902-426-3100 x176

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2005-196
September 2005

Principal Author

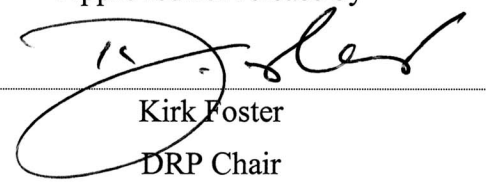
Sean Webb

Approved by

Mark G. Hazen

Scientific Authority

Approved for release by



Kirk Foster

DRP Chair

Work was completed under ARP 11BK and Bilateral CA/UK MOU Annex 1-02

Abstract

The OSPREY 4.3 simulation environment is a very complex tool that assumes the use of JAVA programming to create new tactics. In addition, a new scenario generation process was incorporated in Version 4.0 that is not documented in the previous user guides. This document describes a methodology for creating scenarios and new tactics as a guide to new users. It also documents maintenance work conducted on the software.

Résumé

L'environnement de simulation OSPREY 4.3 est un outil d'une grande complexité qui utilise la programmation JAVA pour créer de nouvelles tactiques. En outre, la version 4.0 intègre un nouveau processus de création de scénarios qui n'a pas été documenté dans les précédents guides de l'utilisateur. Ce document décrit une méthodologie de création de scénarios et de nouvelles tactiques sous la forme d'un guide à l'intention des nouveaux utilisateurs. On y traite également des tâches de mise à jour exécutées sur le logiciel.

This page intentionally left blank.

Executive summary

Scenario and Tactics Development for OSPREY 4.3

S. Webb

J. Nickerson; DRDC Atlantic CR 2005-196; Defence R&D Canada – Atlantic; September 2005.

Introduction

The OSPREY 4.3 simulation environment is a complex tool for modelling maritime airborne warfare that assumes the use of JAVA programming to create new tactics. In addition, a new scenario generation process was incorporated in Version 4.0 that is not documented in the previous user guides.

Principal Results

This document describes a methodology for creating scenarios and new tactics as a guide to new users. This process was developed as part of a tutorial for users and describes how to generate new tactics, platforms and sub-systems. It also documents maintenance work conducted on the software that is incorporated in the 4.3 release.

Significance of Results

The work in this report enhances the usability of OSPREY 4.3 for use in military studies. OSPREY has a well validated multistatic underwater acoustic environment and models scenarios from detection through tracking to attack.

Future plans

OSPREY is expected to be used in studies to develop multistatic and dipping sonar tactics.

Sommaire administratif

Titre du rapport

Webb, S. et Nickerson, J.; 2005; Développement de scénarios et de tactiques pour OSPREY 4.3; RDDC Atlantique CR 2005-196; Recherche et développement pour la défense Canada - Atlantique

Contexte

L'environnement de simulation OSPREY 4.3 est un outil complexe servant à modéliser des tactiques de guerre navale aéroportée, et qui utilise la programmation JAVA pour créer de nouvelles tactiques. En outre, la version 4.0 intègre un nouveau processus de création de scénarios qui n'a pas été documenté dans les précédents guides de l'utilisateur.

Principaux résultats

Ce document décrit une méthodologie de création de scénarios et de nouvelles tactiques sous la forme d'un guide à l'intention des nouveaux utilisateurs. Ce processus a été mis au point dans le cadre d'un tutoriel destiné aux utilisateurs, et décrit comment générer de nouvelles tactiques, de nouvelles plates-formes et de nouveaux sous-systèmes. On y documente également les tâches de mise à jour du logiciel de la version 4.3.

Importance des résultats

Les travaux qui figurent dans ce rapport mettent en relief l'utilisabilité de OSPREY 4.3 dans le cadre d'études militaires. OSPREY propose un environnement acoustique sous-marin multistatique correctement validé et modélise des scénarios évolutifs, depuis la détection jusqu'à l'attaque, en passant par la poursuite.

Travaux futurs

On prévoit utiliser OSPREY dans le cadre d'études visant à mettre au point des tactiques utilisant des sonars multistatiques et immergés.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1. Introduction

In 2000 Dstl the owner of the OSPREY simulation environment undertook at major upgrade, porting the software from a proprietary object-oriented language to JAVA. As part of this upgrade Canada had Macdonald Detwiler Associates (MDA) port the Underwater Acoustics Module (UAM) to JAVA. Subsequently, a standing offer contract was put in place with MDA to provide periodic maintenance and programming support for the software in Canada. Under contract W7707-03-2311/001/HAL, a simple High Level Architecture (HLA) interface was constructed so that Osprey could interact with Defence Research and Development Canada (DRDC) –Atlantic's other HLA federates.

Canada and the United Kingdom (UK) have begun using these new versions of Osprey, and are training personnel to work on them. As this work progresses, it is expected that the personnel will require assistance from the conversion team, both to configure the software correctly and make minor bug fixes. There is, therefore, a requirement for access to Osprey-knowledgeable personnel for short periods of time over a six-month period.

In the work reported in this contractor report, MDA personnel provided maintenance on OSPREY for Canada and the UK, in particular in the area of Airborne Dipping Sonars (ADS). In addition, MDA personnel documented the processes for creating OSPREY scenarios and tactics in the new JAVA language versions, and provided a tutorial for DRDC personnel.

2. OSPREY Overview

Osprey is a naval tactical simulation. It simulates the interaction of various platform types (e.g., aircraft, surface vessels, submarines, and weapons) and their onboard subsystems (e.g., body and propulsion, sonar, and communication systems). It will simulate the movement of the various platforms depending upon the tactics. A random element can be factored in to the initial conditions of a series of, essentially identical, simulation runs. This is called a Monte Carlo run and will determine the sensitivity of the simulation to small random fluctuations.

The objective of Osprey is to provide simulation software that closely represents the behaviour of various vessels (e.g., ships, submarines, aircraft, and sonobuoys) within a given environment (e.g., below the surface of sea for submarines, at the surface of sea for ships, and airborne for aircraft). A test scenario, running in the Osprey graphical user interface (GUI), is shown in Figure 1.

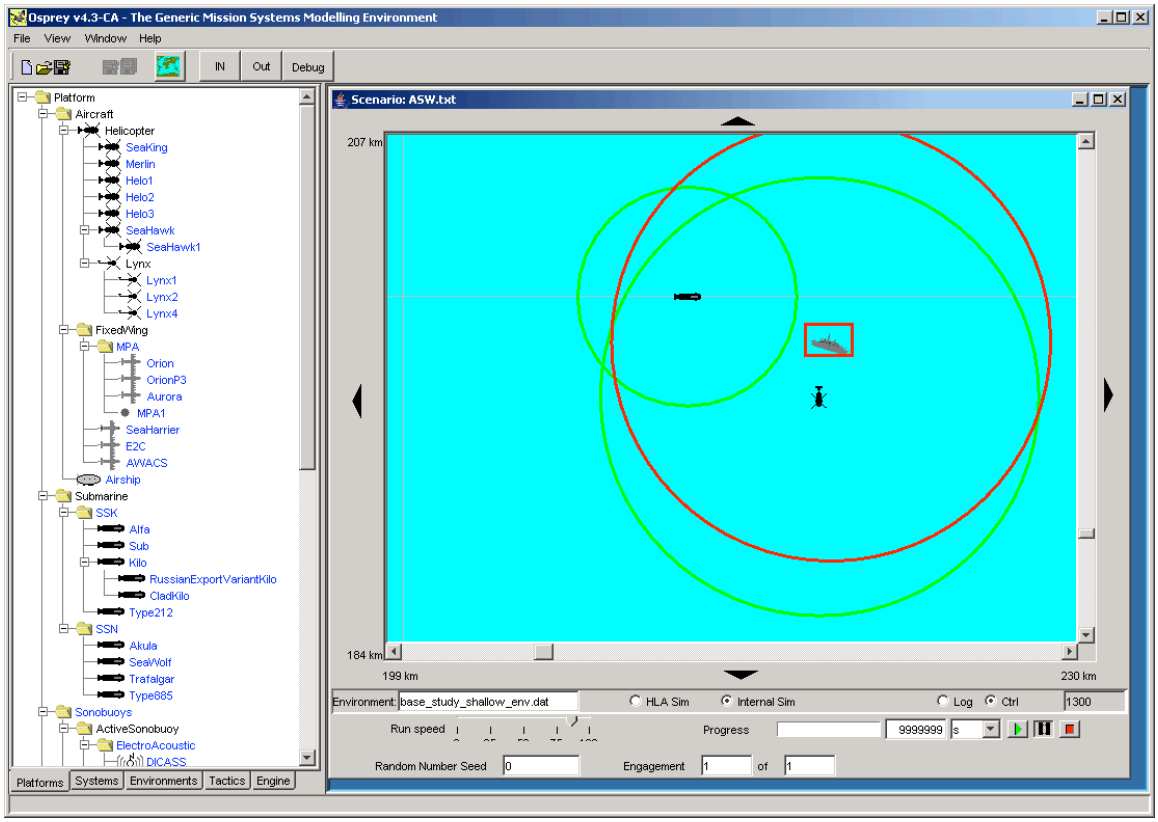


Figure 1: Osprey GUI Main Window

Files stored on disk control the various configuration and execution constraints of Osprey. For instance, one file specifies the way in which various objects are derived from parent objects (e.g., how a Type 22 frigate is derived from a Surface ship, the initial position of a ship, and the initial heading and speed for the ship).

Osprey is a suite of software programs that are designed to run on multiple platforms. This is achieved by writing the software in Java. Java provides platform independence for the GUI and any associated Java operating system calls.

The system can be accessed in two ways:

1. A GUI allows operator interactions and provides visual feedback to the operator.
2. A 'Command Line Interface' enables the system to run without any operator interface.

The design of the system takes a layered approach in that the underlying 'Command Line Interface' of Osprey is independent of the GUI. This is to enable the lower layered system to run from a command line interface with no GUI interaction. The GUI interrogates the lower layers to determine what to display. For instance, the movement of ships, airplanes, and other platforms is calculated by the lower layers, with the GUI non-intrusively 'eaves dropping' to determine the location, heading, and other representative information about the platforms needed to display them to the user.

3. Installation Notes

The paths contained in the following files must be re-configured to run Osprey:

Osprey.ini – (located in the top level directory) Set paths for Osprey to match the directories on your local machine.

Packages.txt – (located in the" \src\system" directory) Set paths for Osprey to match the directories on your local machine.

UserFile.txt – (located in the" \src\system" directory) Set paths for Osprey to match the directories on your local machine.

To run the Osprey GUI application, execute the "osprey.bat" file located in the Osprey "bin" directory.

To run the Osprey application in Monte Carlo mode, execute the "osprey_monte.bat" file located in the Osprey "bin" directory. This batch file passes the following command line parameters to Osprey:

- -username <username>
- -scenario <scenario file name>

For example, to run with user "osprey" and scenario file "monteActiveDetection.txt" use the following command line:

```
java Osprey -username system -scenario monteActiveDetection.txt >
montetestresults_1.txt
```

4. Creating New Tactics

This section describes how to add a new tactic to Osprey. The first subsection shows how to add a generic tactic to Osprey, including which files need to be created and modified. The remaining subsections explain how specific tactics (frigate, helicopter, submarine, and torpedo) were added to Osprey.

Before creating new tactics, a basic anti-submarine warfare scenario (ASW.txt) was created in Osprey. This scenario was created through the following steps:

1. Run the Osprey GUI.
2. Select "File | New Scenario…".
3. Type "ASW.txt" in the "File_name" box.
4. Click the "Create" button.
5. Place a Sub and Frigate on the Scenario map.
6. Give the Sub and Frigate an initial speed and heading (double-click their icons to display speed, heading, and other properties).
7. Save the scenario ("File | Save Scenario").

Note: To save a new version of the scenario with a different name, select "File | Save Scenario As …".

4.1 Generic Tactic Creation

To implement a new tactic in Osprey the following steps are taken:

1. Create a new tactics file in the OspreyHome/src/system/Tactics directory with a ".tac" extension. The filename should be the same as the class name that implements it. Edit the file and add the following lines:

   ```
   1 <ClassName> <MethodName>
   6 end
   ```

   The <ClassName> is the name of the Java class that will handle the event processing for this tactic and the <MethodName> is the name of the method from the class that will be called when Tactical Review events are generated.

2. Edit the file OspreyHome/src/system/ClassData/Tactics/hierarchy.idx. This is the file that lists all the tactics that will be available to operators when running the Osprey software. Find the line that begins with "Tactics=" and add the class name to the list. The names must be separated by a "*" character.

3. Create a Java class in the directory OspreyHome/src/sources/osprey/domain/tactic. This class must have the same name as the class name listed in the tactic file created in step 1. This new class will extend the Tactic class and should contain a constructor, an initialize method, and the method listed in the tactic file created in step 1. This method will have no return value and only one parameter (a tactical event object). This is the method that will be called whenever the platform that is associated with it generates a tactical event. The specific implementation

details of this method will depend on the tactics you are attempting to generate. The initialize method will be called when the platform to which the tactic is associated is first created, as well as when the platform is reset at the start of each iteration.

4. To add this tactic to a platform, open a scenario while running Osprey and double-click on the platform you want to add the tactic to. This will bring up a properties window for that platform, as shown in Figure 2. Click the button next to the edit box for tactics. This will bring up a list of available tactics, as shown in Figure 3. Select the desired tactic and press the "Open" button. Click the "OK" button to accept the change. Now that platform will use the tactic class selected to process its tactical events.

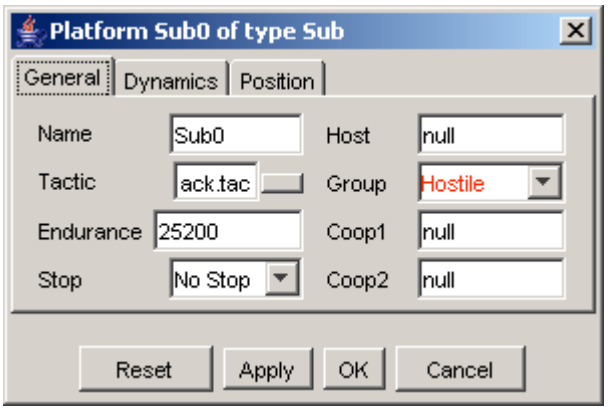


Figure 2: Platform Properties

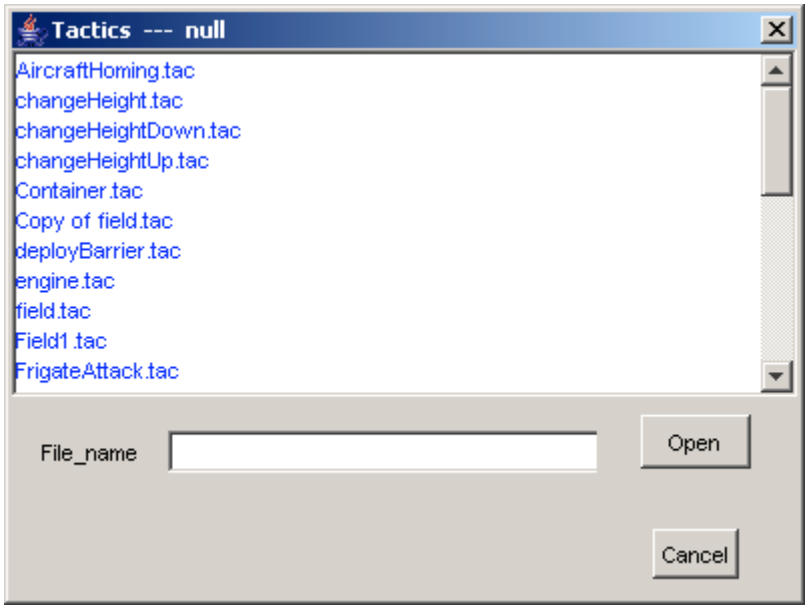


Figure 3: Tactics List

5. The main area to focus on when creating a new tactic is the tactical method created in the new Java class. This method receives tactical events and acts on those events. If a new contact is detected, the tactic may move the host platform toward or away from the

contact depending on the event. A new platform can be deployed from a tactic method, such as deploying a torpedo from a submarine. One way to write a new tactical method is to look at the other classes in the "OspreyHome/src/sources/osprey/domain/tactic" directory and use those classes and methods as example code.

4.2 Frigate Tactic

The frigate tactic was developed using the following information from Annex A of the Statement of Work (SOW) for Osprey Maintenance and User Support [Attachment to Ref 1]:

> *When the sub has been detected via passive or active sonar, it [the Frigate] turns off its active sonar and contacts the helicopter. If it was an active detection, it requests the helicopter to move to the detected location and begin dipping in the area. If it was a passive detection, it provides the helicopter with the line of bearing and asks it to begin searching along that bearing line. The ship manoeuvres and moves directly away from the sub. The ship continues to monitor the area with its passive sonar and avoids the sub.*

The following subsections describe how the frigate tactic was created and added to an existing Osprey platform.

4.2.1 Create Tactics File

A new text file named "LocateSub.tac" was created in the "OspreyHome/src/system/Tactics" directory. The contents of this file are:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    14 January 2005 at 10:09:56
// File name:        LocateSub.dat
//-------------------------------------------------------
// Description:
// LocateSub
//-------------------------------------------------------
1 LocateSub locateSub
6 end   // end of modifications...
```

4.2.2 Modify hierarchy.idx File

The file "OspreyHome/src/system/ClassData/Tactics/hierarchy.idx" was edited to add the new LocateSub tactic. After adding the LocateSub tactic, the hierarchy.idx file contained the following:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    23 August 2002 at 15:48:03
// File name:        hierarchy.idx
```

```
//-------------------------------------------------------
Tactic=TestDynamics*SurfaceAttack*AircraftHoming*LocateSub*SonarDe
tect*SimpleSonarDetect*MissileHomingTactic*Field*Container*Racetra
ck
Field=Field1
Racetrack=Racetrack1*Racetrack2*Racetrack3
```

4.2.3 Create Java Class for Tactic

The file "LocateSub.java" was created in the "OspreyHome/src/sources/osprey/domain/tactic" directory. The following code highlights the most significant parts of the LocateSub class:

```
public class LocateSub extends Tactic
{
  public void LocateSub()
  {
    // Set variables here that do not have to be reset each time
    // the platform is reset. For variables that need to be
    // reset each time the platform is reset, use the "initialise"
    // method.
  }
  public void initialise(Platform hostPlatform, ArrayList
tacticDefinition)
  {
    // Set variables here that have to be reset each time
    // the platform is reset.
  }
  public void locateSub(TacticalEvent event)
  {
    // Called whenever the platform receives a tactical event.
    // This is the method where you write the tactic logic.
  }
}
```

As the code shows, the locateSub method handles the tactical events as they arrive. Each tactical event has an event number. The locateSub method handles the following events: RESET, UPDATE, PING, ACTIVE_DETECTION, PASSIVE_BEARING_CONTACT, and PASSIVE_DETECTION.

The ACTIVE_DETECTION event is received when the active subsystem on the frigate detects an enemy contact. When an ACTIVE_DETECTION event is received, the tactic turns off the active

sonar, changes the ship heading to move away from the contact, deploys a helicopter (if not already deployed), and sends a tactical event of type ACTIVE_CONTACT to the helicopter.

The PASSIVE_BEARING_CONTACT or PASSIVE_DETECTION events are received when the passive subsystem on the frigate detects an enemy contact. When a PASSIVE_BEARING_CONTACT or PASSIVE_DETECTION event is received, the tactic changes the ship heading to move away from the contact, deploys a helicopter (if not already deployed), and sends a tactical event of type PASSIVE_BEARING_CONTACT to the helicopter.

A PING event is received from the locateSub method and is used to periodically trigger the locateSub method. This causes the locateSub method to be called on a regular basis, even when no external events have occurred. This is useful so that the tactic can do "housekeeping" tasks, such as changing heading/speed if moving in a pattern or activating the active sonar subsystem. When a PING event is received, the tactic generates another PING event on a 60 second delay and issues an OspreyEvent to the active subsystem to call the processSignal method.

Information can be stored within the class between calls to the locateSub method. In the frigate tactic, the number of helicopters on the ship is tracked. If this number is greater than 0 and an active or passive detection is made, a helicopter is deployed. As another example, if an active contact has been detected this information could be stored and used during a subsequent tactical event to determine if the contact is moving toward the frigate. If the contact is moving toward the frigate it could then start evasive manoeuvres.

4.2.4 Add Tactic To Platform

The LocateSub tactic was added to the Frigate platform using the following steps:

1. Start the Osprey GUI.
2. Open the ASW.txt scenario.
3. Double-click on the Frigate platform icon displayed in the Scenario window, which opens a properties window for that platform (similar to the one shown in Figure 2).
4. Click the button next to the edit box for tactics, which displays a list of available tactics (see Figure 3).
5. Select the "LocateSub" tactic and press the "Open" button.
6. Click the "OK" button to accept the change.

Now the Frigate platform in the ASW.txt scenario will use the LocateSub tactic class to process its tactical events.

4.3 Helicopter Tactic

The helicopter tactic was developed using the following information from Annex A of the SOW for Osprey Maintenance and User Support [Attachment to Ref 1]:

> *When directed by the ship to search for the sub, the helicopter goes to the location/bearing and begins searching using its dipping sonar. When detection is made, the helicopter moves to the detection location and dips again. When the helicopter is above the sub, it launches a torpedo. Three minutes after launching it begins dipping again in that area. If another detection is made it will again get itself above the submarine and launch another torpedo.*

The following subsections describe how the helicopter tactic was created and added to an existing Osprey platform.

4.3.1 Create Tactics File

A new text file named "SubAttack.tac" was created in the "OspreyHome/src/system/Tactics" directory. The contents of this file are:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    14 January 2005 at 10:09:56
// File name:        SubAttack.dat
//-------------------------------------------------------
// Description:
// SubAttack
//-------------------------------------------------------
1 SubAttack attackSub
6 end   // end of modifications...
```

4.3.2 Modify hierarchy.idx File

The file "OspreyHome/src/system/ClassData/Tactics/hierarchy.idx" was edited to add the new SubAttack tactic. After adding the SubAttack tactic, the hierarchy.idx file contained the following:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    23 August 2002 at 15:48:03
// File name:        hierarchy.idx
//-------------------------------------------------------
Tactic=TestDynamics*SurfaceAttack*AircraftHoming*LocateSub*SubAtta
ck*SonarDetect*SimpleSonarDetect*MissileHomingTactic*Field*Contain
er*Racetrack
Field=Field1
Racetrack=Racetrack1*Racetrack2*Racetrack3
```

4.3.3 Create Java Class for Tactic

The file "SubAttack.java" was created in the "OspreyHome/src/sources/osprey/domain/tactic" directory. The following code highlights the most significant parts of the SubAttack class:

```
public class SubAttack extends Tactic
```

```
{
  public void SubAttack()
  {
    // Set variables here that do not have to be reset each time
    // the platform is reset. For variables that need to be reset
    // each time the platform is reset, use the "initialise"
    // method.
  }
  public void initialise(Platform hostPlatform, ArrayList
tacticDefinition)
  {
    // Set variables here that have to be reset each time
    // the platform is reset.
  }
  public void attackSub(TacticalEvent event)
  {
    // Called whenever the platform receives a tactical event.
    // This is the method where you write the tactic logic.
  }
}
```

As the code shows, the attackSub method handles the tactical events as they arrive. Each tactical event has an event number. The attackSub method handles the following events: RESET, UPDATE, ACTIVE_CONTACT, ACTIVE_DETECTION, PASSIVE_BEARING_CONTACT, PASSIVE_DETECTION, and PING.

The PASSIVE_BEARING_CONTACT event is received from the frigate when the frigates sonar detects an enemy contact. When the helicopter tactic receives a PASSIVE_BEARING_CONTACT event it gets the bearing information. It should then fly to the frigate and begin flying along the contact bearing, periodically dipping to detect the enemy submarine. (Note: After discussion with the Scientific Authority [SA], this functionality was not included due to budget constraints. The code includes comments on how it should be implemented.)

When the helicopter tactic receives an ACTIVE_CONTACT event, it retrieves the target position, calls method moveToCoordinate of the Dynamics class to move the helicopter to the target position, and sets the current activity to MOVE_TO_TARGET. The currentActivity variable is a data member of the Tactic class that is used to keep track of what the platform is doing. Each time the attackSub method is called, it executes code specific to the current activity. The MOVE_TO_TARGET code checks if the helicopter has reached the given target position. When it has reached the target position it changes the current activity to SEARCH. The SEARCH code stops the helicopter, lowers and activates the dipping sonar, and generates a PING event that is sent back to the attackSub method.

The PING event sends a PING event back to the attackSub method and checks whether the Active Dipping Sonar (ADS) is deployed. If it is deployed it sends an event to call the processSignal method of the active transmitter on the ADS.

The attackSub method continues pinging until an ACTIVE_DETECTION event is received. When this event is received, the target bearing is calculated and the current activity is set to DESTROY. The DESTROY code gets the weapon deployment system (i.e., the torpedo), sets the torpedo heading to the target bearing, sets the torpedo speed, deploys the torpedo, and starts a wait timer. The current activity is set to WAIT, and each time the WAIT code is called it checks whether or not the timer has expired. When the timer expires, the WAIT code sets the current activity to SEARCH. If the submarine has not been destroyed within the specified time, and an active detection is made, another torpedo will be fired. The current activity will then be set back to WAIT. Once the submarine has been destroyed, the helicopter tactic will no longer receive active detections and will not fire additional torpedoes.

4.3.4 Add Tactic To Platform

Since the helicopter platform is created and deployed using Java code it is not initially part of the ASW.txt scenario shown in the Osprey GUI. Because of this, the SubAttack tactic was added to the Helicopter platform in the Java code as described in section 4.2.3.

4.4 Submarine Tactic

The submarine tactic was developed using the following information from Annex A of the SOW for Osprey Maintenance and User Support [Attachment to Ref 1]:

> *When the frigate is detected using passive sonar, the sub alters course and moves aggressively towards the frigate. It continues tracking the frigate with its passive sonar and modifying its course as required to keep on its trail.*
> *When the sub is within torpedo range of the ship, it fires a torpedo at the ship. If it misses, it continues to follow the ship and fires another torpedo after 1 minute.*
> *When dipping sonar is detected by the sub, or the sub is fired at, the sub goes deep, manoeuvres slightly and remains silent. After five minutes it decreases depth and begins to move again in the previous direction of the ship, changing direction based on any new passive detections.*

The following subsections describe how the submarine tactic was created and added to an existing Osprey platform.

4.4.1 Create Tactics File

A new text file named “FrigateAttack.tac” was created in the “OspreyHome/src/system/Tactics“ directory. The contents of this file are:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    14 January 2005 at 10:09:56
// File name:        FrigateAttack.dat
//-------------------------------------------------------
```

```
// Description:
// FrigateAttack
//-------------------------------------------------------------
1 FrigateAttack frigateAttack
6 end   // end of modifications...
```

4.4.2 Modify hierarchy.idx File

The file "OspreyHome/src/system/ClassData/Tactics/hierarchy.idx" was edited to add the new FrigateAttack tactic. After adding the FrigateAttack tactic, the hierarchy.idx file contained the following:

```
//-------------------------------------------------------------
// Last modified by: system
// Last modified:    23 August 2002 at 15:48:03
// File name:        hierarchy.idx
//-------------------------------------------------------------
Tactic=TestDynamics*SurfaceAttack*AircraftHoming*FrigateAttack*Loc
ateSub*SubAttack*SonarDetect*SimpleSonarDetect*MissileHomingTactic
*Field*Container*Racetrack
Field=Field1
Racetrack=Racetrack1*Racetrack2*Racetrack3
```

4.4.3 Create Java Class for Tactic

The file "FrigateAttack.java" was created in the "OspreyHome/src/sources/osprey/domain/tactic" directory. The following code highlights the most significant parts of the FrigateAttack class:

```
public class FrigateAttack extends Tactic
{
  public void FrigateAttack()
  {
    // Set variables here that do not have to be reset each time
    // the platform is reset. For variables that need to be reset
    // each time the platform is reset, use the "initialise"
    // method.
  }
  public void initialise(Platform hostPlatform, ArrayList
tacticDefinition)
  {
```

```
        // Set variables here that have to be reset each time
        // the platform is reset.
    }
    public void frigateAttack(TacticalEvent event)
    {
        // Called whenever the platform receives a tactical event.
        // This is the method where you write the tactic logic.
    }
}
```

As the code shows, the frigateAttack method handles the tactical events as they arrive. Each tactical event has an event number. The frigateAttack method handles the following events: RESET, UPDATE, SONAR_DETECTED, and PASSIVE_DETECTION.

The SONAR_DETECTED event is received when the submarine detects active sonar being used. The tactic should then make the submarine go deep, manoeuvre slightly and remain silent for five minutes. (Note: After discussion with the SA, this functionality was not included due to budget constraints.)

When the PASSIVE_DETECTION event is received the tactic gets the target position, sets the current activity to FOLLOW, and calculates the target bearing. The FOLLOW code sets the submarine heading to match the target bearing, checks if it is close enough to fire a torpedo, and if so, sets the current activity to DESTROY.

The DESTROY code gets the weapon deployment system (i.e., the torpedo), sets the torpedo heading to the target bearing, sets the torpedo speed, deploys the torpedo, and starts a wait timer. The current activity is set to WAIT, and each time the WAIT code is called it checks whether or not the timer has expired. When the timer expires, the WAIT code sets the current activity to SEARCH. If the target has not been destroyed within the specified time, and a passive detection is made, another torpedo will be fired. The current activity will then be set back to WAIT. Once the target has been destroyed, the submarine tactic will no longer receive passive detections and will not fire additional torpedoes.

4.4.4 Add Tactic To Platform

The FrigateAttack tactic was added to the submarine platform using the following steps:

1. Start the Osprey GUI.
2. Open the ASW.txt scenario.
3. Double-click on the Submarine platform icon displayed in the Scenario window, which opens a properties window for that platform (similar to the one shown in Figure 2).
4. Click the button next to the edit box for tactics, which displays a list of available tactics (see Figure 3).
5. Select the “FrigateAttack” tactic and press the “Open” button.
6. Click the “OK” button to accept the change.

Now the Submarine platform in the ASW.txt scenario will use the FrigateAttack tactic class to processes its tactical events.

4.5 Torpedo Tactic

The torpedo tactic was not explicitly specified, but was required to develop the scenario. The torpedo tactic was developed using the following information from the submarine tactic description found in Annex A of the SOW for Osprey Maintenance and User Support [Attachment to Ref.1]:

When the sub is within torpedo range of the ship, it fires a torpedo at the ship.

The following subsections describe how the torpedo tactic was created and added to an existing Osprey platform.

4.5.1 Create Tactics File

A new text file named "TorpedoHoming.tac" was created in the "OspreyHome/src/system/Tactics" directory. The contents of this file are:

```
////////////////////////////
1 TorpedoHoming nearestTarget


6 end   // end of modifications...
////////////////////////////////////////////
```

4.5.2 Modify hierarchy.idx File

The file "OspreyHome/src/system/ClassData/Tactics/hierarchy.idx" was edited to add the new TorpedoHoming tactic. After adding the TorpedoHoming tactic, the hierarchy.idx file contained the following:

```
//-------------------------------------------------------
// Last modified by: system
// Last modified:    23 August 2002 at 15:48:03
// File name:        hierarchy.idx
//-------------------------------------------------------
Tactic=TestDynamics*SurfaceAttack*AircraftHoming*FrigateAttack*Loc
ateSub*SubAttack*SonarDetect*SimpleSonarDetect*MissileHomingTactic
*TorpedoHoming*Field*Container*Racetrack
Field=Field1
Racetrack=Racetrack1*Racetrack2*Racetrack3
```

4.5.3 Create Java Class for Tactic

The file "TorpedoHoming.java" in the "OspreyHome/src/sources/osprey/domain/tactic" directory was created. The following code highlights the most significant parts of the TorpedoHoming class:

```
public class TorpedoHoming extends Tactic

{

  public void TorpedoHoming()

  {

// Set variables here that do not have to be reset each time
// the platform is reset. For variables that need to be reset each
time // the platform is reset, use the "initialise" method.

  }

  public void initialise(Platform hostPlatform, ArrayList
tacticDefinition)

  {

// Set variables here that have to be reset each time
// the platform is reset.

  }

  public void nearestTarget(TacticalEvent event)

  {

// Called whenever the platform receives a tactical event.

// This is the method where you write the tactic logic.

  }

}
```

As the code shows, the nearestTarget method handles the tactical events as they arrive. Each tactical event has an event number. The nearestTarget method handles the following events: RESET and PASSIVE_DETECTION.

When the PASSIVE_DETECTION event is received, the tactic gets the target position, sets the current activity to FOLLOW, if it was previously SEARCH, and calculates the target bearing. The FOLLOW code sets the torpedo heading to match the target bearing, calculates the distance between the torpedo and the target, and if it is less than the destination distance (i.e., the defined range to destroy the target), sets the torpedo and target alive state to false (i.e., target is destroyed) and the current activity to END.

4.5.4 Add Tactic To Platform

Since the torpedo platform is created and deployed using Java code, it is not initially part of the ASW.txt scenario shown in the Osprey GUI. Because of this, the TorpedoHoming tactic was added to the Torpedo platform in the Java code as described in sections 4.3.3 and 4.4.3.

5. Adding New Simulation Elements to OSPREY

5.1 Adding A New Platform

To add a new platform to Osprey, the following steps are taken:

1. Decide the new platform's hierarchical position relative to the existing "Java" platforms. These are located in the OSP_HOME\sources\osprey\domain\platform directory.
2. Create an object that "extends" the parent platform, in the osprey.domain.platform package. If the object has no addition methods, other than an empty constructor, it can now be compiled. However, if one were to run Osprey, the new platform would not be visible. This is because we also need to reference the new platform in the "system" areas platform hierarchy description file (hierarchy.idx) found in SYSTEM_PATH\ClassData\Platforms. If a new icon is desired for this platform, a cross-reference needs to be made in the icon mapping file (iconMap.idx), also found in SYSTEM_PATH\ClassData\Platforms.
3. To "customize" the platform, we need to state what sub-systems the platform can have. This is achieved by creating String instance variables for each required sub-system. For example, if the platform were to have an ElectroOptic system, one might create an instance variable electroOpticSys initialized as "null". The Platform object has already created instance variables: bodyPropSys, commandSys, deviceDeploymentSys, communicationSys, radar1Sys, radar2Sys, radar3Sys, and esmSys. Step 4 describes how these are subsequently used.
4. A method setupSubsystems() needs to be overridden, which assigns a String array to the arrayOfSystems variable. The array contains the names of the instance variables (i.e., subsystems) this platform will have. For example, in Helicopter this looks like:

```
protected void setupSubsystems()
{
    arrayOfSystems = new String[6];
    arrayOfSystems[0] = "bodyPropSys";
    arrayOfSystems[1] = "radar1Sys";
    arrayOfSystems[2] = "adsSys";
    arrayOfSystems[3] = "deviceDeploymentSys";
    arrayOfSystems[4] = "communicationSys";
    // command system should always be added last
    arrayOfSystems[5] = "commandSys";
}
```

   It should be stressed that Osprey expects ALL platforms to have a bodyPropSys, and a commandSys specified. This is because the bodyPropSys defines the location and movement of a platform, and the commandSys defines its behaviour (even if it is has no behaviour, it must still have a commandSys).

5. Finally, to make the GUI consistent with the arrayOfSystems, the method makeObjectGUI() needs to be overridden (otherwise, only the bodyPropSys can be modified). For example, this is the following method from the Helicopter object:

```
public void makeObjectGUI()

{
  completeUI.add(new DynamicCommentField(this, "Systems List"));

  completeUI.add(new DynamicSubsystemField(this, "Body and Propulsion
    System", "bodyPropSys"));

  completeUI.add(new DynamicSubsystemField(this, "ADS System",
    "adsSys"));

  completeUI.add(new DynamicSubsystemField(this, "Radar System",
    "radar1Sys"));

  completeUI.add(new DynamicSubsystemField(this, "Device Deployment
    System", "deviceDeploymentSys"));

  completeUI.add(new DynamicSubsystemField(this, "Communications
    System", "communicationSys"));

  completeUI.add(new DynamicSubsystemField(this, "Command System",
    "commandSys"));

}
```

5.2 Adding a New Subsystem To a Platform

To add a new subsystem to an Osprey platform, the following steps are taken:

1. Edit the file \OspreyHome\src\system\ClassData\Subsystems\hierarchy.idx. Add the name of the new subsystem to the file. For example, if you are creating a new passive sonar, find the line starting with "PassiveSensor=" and add the name of the new passive sensor to the end of the list. It should be preceded by a '*' character.

2. Edit the Java class for the platform you want to add the new subsystem to. In the setupSubsystem method, add a new string to arrayOfSubsystems and add the name of the new subsystem to the list. Then edit the method makeObjectGui and add a new completeUI line. Add a new public String member to the class to hold the name of the new subsystem. For example, to add a new sonar receiver subsystem, the following lines would be added to the platform source code:

   setupSubsystems method:

   ```
   arrayOfSystems[3] = "passiveReceiverSys";
   ```

   makeObjectGUI method:

   ```
   completeUI.add(new DynamicSubsystemField(this, "Passive Sensor
     System", "passiveReceiverSys"));
   ```

   public data member:

   ```
   public String passiveReceiverSys = "null";
   ```

   Note: The name of the public data member must be the same as the string name in the arrayOfSubsystems, and the same as the string for the third parameter of the completeUI.add method.

3. Run Osprey and select the “Systems” tab. Set any parameters for the new subsystem you are modifying.
4. Select the Platforms and add the new subsystem to the desired platform.

5.3 Adding New Tactical Event Types

To add a new type of Tactical Event that can be generated by Osprey edit the TacticalEvent.java class in directory “<OSPREY_HOME>\src\sources\osprey\scenario\simevent” and add a new data member as follows:

```
public static final int <event name> = <event number>;
```

where <event name> is the name of the new event you are adding and <event number> is the numerical value of the new event. Make sure the event number you chose is not already in use by another tactical event. Osprey currently generates the following tactical events:

```
public static final int RESET = 0;
public static final int MANOEUVRE_COMPLETE = 1;
public static final int MOTION_COMPLETE = 2;
public static final int MANOEUVRE_NOT_ACHIEVABLE = 3;
public static final int WAIT_COMPLETE = 10;
public static final int SONAR_DETECTED = 35;
public static final int PASSIVE_DETECTION = 84;
public static final int OUT_OF_ENDURANCE = 90;
public static final int INITIAL_RADAR_CONTACT = 100;
public static final int CPA_DETECTED = 105;
public static final int PASSIVE_BEARING_CONTACT = 108;
public static final int PASSIVE_VEL_SPEED_CONTACT = 109;
public static final int PING = 110;
public static final int ACTIVE_CONTACT = 120;
public static final int ACTIVE_DETECTION = 130;
public static final int UPDATE = 1000;
```

6. Osprey Code Changes

All changes and additions to Osprey Java code and supporting files are documented in Table 1.

Table 1: Changes to Osprey Code

Class	Method	Change Description
OspreyBWE1	processBelowWater	The list of valid targets was assumed to be all submarines. Fixed code to make it a generic list of platforms.
	triggerActiveReceivers	Added code to work around problems with incomplete ADS implementation. Added a workaround to check if there is a Helicopter, and if so, add the ADS for the helicopter as a valid receiver.
Submarine	SetupSubsystems, makeObjectGUI	Added a passive sensor system.
Platform	Public Data	Moved getNBData and getBBData flags from Submarine class to Platform class.
	initialise	For CompoundSubsystem code, changed call from "temp.initialise(data);" to "temp.initialise(this);" to work around problems with incomplete ADS implementation.
PassiveSonar	processPassiveSignal	Change aircraft to a generic platform.
	processActiveSignal	Added a workaround for ADS that sets destPlatform = ADS if the host is ADS.
Torpedo		Added a new platform based on the Missile platform.
TorpedoHoming		New tactic class to handle launched torpedoes.
FrigateAttack		New tactic for Submarine platform. Seek and Destroy enemy platforms while evading detection.
LocateSub		New tactic for Frigate platform. Seeks submarine and deploys helicopter to attack when one is located.
SubAttack		New tactic for Helicopter class. Seeks sub with ADS and fires torpedo when one is found.
SurfaceVessels	SetupSubsystems, makeObjectGUI	Added active, passive, signature, and communications subsystems to class.

Class	Method	Change Description
SimpleSonar	processPassiveSignal	Added tactical events for active detections.
SonicsProcessor	All	Made all sonobuoy specific platforms generic platforms so the Sonics Processor class would work for active and passive sonars attached to platforms other than sonobuoys.
ContactData		Changed all sonobuoy platforms to generic platforms.
TargetActiveData		Changed all sonobuoy platforms to generic platforms.
CompoundSubsystem	initialise	Added method "initialise", which takes a Platform instance as an argument. This change was made to work with a change made to the Platform class.
SonarTransmitter		Modified beamwidthVertical and coverageLimits data members to have different default values. The dipping sonar (ADS) was not picking up the submarine. The problem seemed to be that if the sonar and the target were not at the same depth, there would be no detection. Changing these values allowed the sub to be detected.
IterationManager		Set data member "simulationRunMode" to default to "Scenario.INTERNAL" to allow Monte Carlo batch runs to occur.
Scenario		Set data member "simulationRunMode" to default to "Scenario.INTERNAL" to allow Monte Carlo batch runs to occur.
osprey_monte.bat		New file showing how to run Osprey in Monte Carlo batch mode.
Torpedo.dat		New file used for ASW.txt scenario.
Flash.dat		Added the following systems: sonicsProcessorSys, sdhsSys, tdhsSys, fourStateKalmanSys, sixStateKalmanSys, communicationSys, and commandSys.

Class	Method	Change Description
FlashRx.dat		Added the following systems: startProcessingImmediately, selfNoiseFilename, updateRate, effectiveRange, isArrayGainFixed, arrayGain, arrayGainFilename, isHorizBeamwidthFixed, horizBeamwidth, horizBeamwidthFilename, isVertBeamwidthFixed, vertBeamwidth, vertBeamwidthFilename, NBlowerFreq, NBupperFreq, BBlowerFreq, BBupperFreq, rxMode, updateRate, bearingError, rangeError, speedError, pitchError, and integrationPeriod.
FlashTx.dat		Added the following systems: startProcessingImmediately, beamwidthVertical, numGroupsToTransmit, groupRepetitionInterval, randomlyFMCW, modeFlag, numPingSeconds, numCWTrainsPerGroup, priCW, pulseLengthNB, numFMTrainsPerGroup, priFM pulseLengthBB, updateRate, effectiveRange, activeFrequencyBB, activeBandwidthBB, activeFrequencyNB, txSourceLevelNB, txSourceLevelBB, and acousticMultiStaticTransmitter.
SubmarineBP.dat		Changed minAlt to -1000.0. Without this change the submarine cannot go below the surface.
SubAttack.dat		New file used for ASW.txt scenario.
ADS	highlightPlatform getInstanceData addEvent	Added code and new methods to workaround problems with incomplete ADS implementation. Added passiveReceiverSys, sonarTransmitterSys, activeTransmitterSys, commandSys, tdhsSys, sdhsSys, sonicsProcessorSys, and communicationSys. Added highlightPlatform, getInstanceData, and addEvent methods. The getInstanceData method clones host (helicopter) instance data and adds the cable length to the z position.

Class	Method	Change Description
ADSBP	winchADS updatePosition getADSDepth getADSHeight getAbsADSHeight processPassiveSignal getAbsDeployed	Fixed a bug in multiple methods that used incorrect code to get the subsystems. Was using helo_id.<nameofsystem>, instead of a string (i.e. - "commandSys"). Modified method “winchADS” to include cable length as part of the ads position. Fixed a bug in method "updatePosition" where the OspreyEvent being created specified "movePlatform" as the destination method instead of "updatePosition". Modified method “getADSDepth” to include cable length as part of the ads position. Fixed a bug in getADSDepth where values greater than zero were treated as being above water, when these values represent a depth below water. Modified method “getADSHeight” to include cable length as part of the ads position. Fixed a bug in method “getADSHeight” where values less than zero were treated as being below water when these values represent a height above water. Modified method “getAbsADSHeight” to include cable length as part of the ads position. Added “processPassiveSignal” method. Added “getAbsDeployed” method, which can be used by a tactic to determine if the ABS is fully deployed and ready to be used.
Helicopter	getSubsystem	Added “getSubsystem” method. Overrides super method. If a subsystem is not found on the Helicopter the ADS subsystem (if available) is also searched.

7. Maintenance

No bug fixes were requested during the contract, although some assistance was given to UK personnel in the development of dipping sonars.

7.1 Bugs and Deficiencies found during Callup

7.1.1 ADS

The ADS and Active Dipping Sonar Body and Propulsion (ADSBP) classes in Osprey are not fully implemented or tested. As documented in section 6, some workarounds were made to allow development of a scenario and tactic using a helicopter outfitted with ADS. Without these modifications, the ADS would not work at all. One of the problems found was that the z coordinate for the ADS was calculated incorrectly, resulting in the winch system of the ADSBP not lowering the ADS into the water. Another problem was that the sonar processing code was referencing the helicopter, instead of the ADS in some cases, resulting in no active transmissions and no reception of sonar information by the ADS.

These problems seem to stem from an incomplete design and implementation of the CompoundSubsystem class (ADS is a subclass of CompoundSubsystem). The CompoundSubsystem appears to have been added to Osprey as a compromise between a subsystem on a platform and a completely separate platform. The two subclasses of CompoundSubsystem in Osprey are ADS and TowedArray, both of which are tethered to a host platform by a cable. The problem with the current code is that the ADS is being treated in two different ways within Osprey. At times, it is treated as a subsystem and other times as a separate platform. In some cases, the code is expecting the ADS to have all the information a platform would have. In these instances, a special case was created in the code to check for an ADS system and point to the correct platform information stored in the helicopter object. The CompoundSubsystem, its subclasses, and all code that uses these classes should be reworked to treat CompoundSubsystem classes differently than subsystems and platforms.

7.1.2 Current Platform Position

Osprey does not show the current position information for a platform when you open its properties in the scenario window. To reproduce, check the position information in the properties of a platform before running the scenario, run the scenario for a while and note the platform moves, pause the scenario, check the position information again. The position is the same as when you first checked.

7.1.3 Platform Stopping Condition

The stopping condition for a platform is not used. For each platform in a scenario a “Stop” condition can be set. The stop conditions available are: "No Stop", “Initial Detect", “Wpn Deployed", “Wpn Hit", “Low Endurance", and “No Endurance”. This value is stored in the

"PlatformInstanceData" class in the "stopReason" field, but is not used within the Osprey code to stop the scenario as intended.

7.1.4 Crash with All Debugging On

With all debugging turned on (all levels for all classes) and running at full speed (i.e., much greater than real time) Osprey crashed. This occurred while running outside the debugger, so no additional information on the source of the crash was collected. This may be due to Osprey trying to log too much information at once to the same log file.

7.1.5 GUI Usability Issues

There are display and usability issues with the Osprey GUI. In some cases (such as platform properties in the scenario window), units for endurance, speed and position are not given. There are fields that can be filled in, such as "Host", which have no documentation indicating what value should be used. There are sections of the GUI where there is not enough room for some items to be displayed, so the text is partially cropped. This is especially noticeable in the scenario window.

7.2 Documentation Deficiencies

The documentation is minimal or completely missing for parts of Osprey. This section is intended to describe how some parts of Osprey work, as determined during the course of this call-up. It is meant to be a quick introduction to these topics and does not cover all details.

7.2.1 Batch (Monte-Carlo) Mode

Monte Carlo mode allows the user to run the same scenario multiple times without the user interface. To run Osprey in Monte Carlo mode pass the following command line parameters to Osprey:

- -username <username>
- -scenario <scenario file name>

For example, to run with user "osprey" and scenario file "monteActiveDetection.txt" use the following command line:

```
java Osprey -username system -scenario monteActiveDetection.txt > montetestresults_1.txt
```

The scenario file used can be created within the Osprey GUI, as described in section 4. For a Monte Carlo scenario, the values for random number seed, number of engagements, and the maximum engagement run time should be set to the values needed to obtain useful results. For example, if your scenario usually takes 10000 seconds to complete, the maximum engagement run time should not be less than 10000 or else you will not get useful results in the output files.

7.2.2 Debugging

The debug logging options can be set in Osprey by selecting "View | Debug Flags…". This opens a window that lists every class in Osprey. You can set the debug level for each class individually from L1 to L8. There is also a "Debugging Off/Debugging On" button that is initially set to "Debugging Off" (and coloured red). Click the button to set it to "Debugging On" (it will change colours to green).

Level 1 debug messages are general information messages. Level 8 messages are more specialized messages. Since debugging is specified on a per-class basis, you could implement your own scheme for what the debug levels mean in a new class. For example, you could use the first three levels for debug messages of low, medium, and high importance. The other five levels could be used for temporary debug messages, informational messages, etc.

7.3 Way Ahead

This section describes recommended tasks to improve and maintain Osprey.

7.3.1 Fix ADS

The ADS and ADSBP classes and associated code should be examined to determine a permanent fix for the problems listed in section 7.1.1. This work should be done in consultation with the Osprey maintainers in the UK to coordinate changes and fixes already made by that team. This would allow all ADS functionality to be used and enhancements made without the temporary workarounds that are now in place in the code. Issues, such as drift (if the helicopter is stationary, does the ADS drift through the water) and collision/targeting (can the ADS be destroyed by a torpedo), should be addressed for the CompoundSubsystem class and specifically the ADS.

7.3.2 Display Current Platform Information

The current position of a platform in a scenario should be available to the operator through the GUI. This could be displayed on the main GUI or by opening the platform properties window. As described in section 7.1.2, the current platform position is not currently displayed in the platform properties.

7.3.3 Complete "Stopping Condition" Implementation

The platform stopping condition set in the platform properties window should either be used in the Osprey code or removed. This feature should be re-examined to determine if it has value in scenario and tactic planning. If it is valuable, it should be fully implemented for all platforms.

7.3.4 Fix Crash with All Debug Logging On

The cause of a crash that was noted when all debugging was turned on with the simulation running at full speed should be determined and (if possible) fixed.

7.3.5 Improve GUI

The Osprey GUI should be investigated to determine deficiencies in its usability. Some known deficiencies are described in section 7.1.5 and include missing units and text which is hidden and cannot be read.

The debugging on/off button should also be marked more clearly to indicate the current state and what effect clicking the button will have on the debugging state. Currently, the button indicates what the current debugging state is. This could be misinterpreted to mean that when the button is clicked it will change the state to the displayed state. For example, the button may say "Debugging On". In the current system that means debugging is turned on. It would be clearer to change the button text to "Debugging Is On" or something similar.

7.3.6 Real World Data

Adding additional real-world maps would enhance the GUI and simulation detail of Osprey. There is currently a "test" map that is a vector-based map showing a small area of coastline. Additional investigation could be made in this area regarding incorporation of real-world environmental data, such as bathy, sound speed profile, etc., used for propagation loss and collision detection/avoidance. Currently, the map data is not used by Osprey to indicate that a platform has collided with the coast. Incorporating real-world data would allow Osprey to operate in a networked environment (such as an HLA federation) where the other applications are using real-world maps and environment data.

7.3.7 Security

The password system for Osprey should be enhanced, modified, or removed. Currently, the usernames and passwords for Osprey are kept in an unprotected text file. The current system was intended to allow multiple users to access different sets of features in Osprey depending on what they were required to do. The username/password dialog that appears when starting Osprey is currently a nuisance and should be removed until it is needed.

7.3.8 Convergence Zones

To assist in developing tactics, the concept of convergence zones could be added to Osprey. Basically, convergence zones can be used with sonar to predict where a target is most likely located (e.g., 20 km from source or 60 km from source but not 35 km from source). For the tactics generated under this call-up, the helicopter could have used convergence zone information to dip its sonar along the bearing line from the frigate toward the submarine. It would have dipped at the first convergence zone, then moved to the next zone if no detection was made. Currently, there is no way to access convergence zone data within tactic code.

7.3.9 Multiple Signature Modes

It would be useful to introduce the concept of multiple signature modes for platforms in Osprey. Currently, a platform has one signature that is used all the time within Osprey. In the real world,

platforms change their signatures depending on engine speed, machinery in use, and other factors. Osprey could be modified to allow each platform to have multiple modes/signatures associated with it, such as “full stop”, “running”, and “running quiet”. Modifying the Signature class data while “running” could do this. By modifying values, such as broadband radiated noise in a tactic to match a situation outlined in the scenario, the scenarios developed with Osprey can be closer to real-world situations.

7.4 Previous Recommendations

The following recommendations from section 7 of the Final Report for Osprey Maintenance and Testing (Document No. DN0511, Issue 1/0 dated 25 March 2004, Call-up Requisition No. W7707-3-2311), should be considered as part of the way ahead:

Figure 1. Publish all Osprey platforms to the HLA Federation. Currently, Osprey only publishes sub-surface composite entity types. By publishing all platforms, Osprey would be more useful as a red-force in HLA simulations.

Figure 2. Subscribe to more types of federates. Currently, Osprey only subscribes to sea surface composite entity types and any sea surface composite entity type it subscribes to will be inserted into its simulation as a destroyer. By adding additional federates, Osprey could be given an expanded role in an HLA federation and would be more useful as a red-force in HLA simulations.

1. Implement the dead reckoning algorithms. Osprey is not properly using the dead reckoning algorithms. These should be investigated and the algorithms should be modified to fit the intended design.

2. Remove the hard coded path names used in Osprey. These path names make it difficult to move Osprey from one machine to another. Use relative paths when possible or a run-time-configurable path, if that is required.

Figure 3. The height of subscribed sea surface platforms seems to be displaying incorrectly in some cases. The value of the height sometimes drops below or above sea level. This may be a problem with the conversion routine or it may be a problem with the values supplied by motion federate. Investigate this problem and implement a fix.

3. The GUI to control the simulation speed has no effect when running in HLA mode. This control should be disabled when running in this mode.

4. The GUI to stop or pause the simulation will kill the federation. These two buttons should be disabled then running in HLA mode.

7.5 Improved Documentation

As described in section 7.2, the documentation on using the Java version of Osprey is incomplete. The units for some GUI items, such as time units, are not specified. This leads the user to enter a

value for endurance without knowing whether it is in seconds, hours, or some other unit. By searching the source code, it was found that the unit for endurance is seconds, but this is not specified in the GUI or documentation. The scenario window settings are also not documented or intuitive, for example, does changing progress units from minutes to seconds affect other parts of the GUI or just the scenario? It is also not specified what happens after changing the random number seed and Engagement values.

The Tactics tab and Engine tab are also not documented. The Tactics tab seems to allow you to create a child of a tactic to specify parameters. For example, you could create a “locateSub” tactic to be used by a Helicopter platform with an “aggressiveness” parameter. Then you could create children of this tactic that you set the aggressiveness to 1, 2, 3, etc. The “locateSub” tactic code would then change the characteristics of the tactic based on the parameter. In this case, a higher value could correspond to the Helicopter moving faster, firing more torpedoes, etc.

All of these items, in addition to other ambiguous areas of Osprey, should be clearly documented for Osprey interface users and Osprey developers. This work should be coordinated with the UK Osprey maintainers to ensure the effort is not duplicated and to draw on their knowledge of the system.

7.6 Tactics System Redesign

The current tactics system in Osprey should be investigated to determine if it meets the user requirements. Depending on the results of the investigation, the Osprey tactics should be re-designed to be a more intuitive tactics definition system. The current system includes a partially implemented tactics scripting language for simple, non-reactive tactics and the use of Java for any complex, reactive tactics. While Java offers a high level of control over the tactics creation, it is not a user-friendly tactic development language. To make it easier to create new tactics, a reusable tactics Application Programming Interface (API) could be created based on existing tactics code. This API would provide some of the basic functionality that is common to many of the tactics, such as searching for another platform, following another platform, evading a platform, and manoeuvring in a specified pattern. The API would be developed to provide a balance between flexibility for software developers and usability for non-programmers.

Another approach to improving tactics within Osprey is to introduce tactics based on "subsumption architecture". This is a behaviour-based approach where there are low-level behaviours running concurrently with high-level behaviours. The low-level behaviours take precedence over the high-level ones. For example, in Osprey, a tactic could be created for a submarine based on subsumption architecture, where the low level behaviours include "Avoid Objects" and "Evade when Active Ping Detected", while the high level behaviours include "Detect Enemy Vessels" and “Move from Point A to Point B”. If the submarine is following the “Move from Point A to Point B” behaviour and it is about to hit the bottom, the "Avoid Objects" behaviour takes over to avoid the bottom. Once the submarine is out of harms way, the “Move from Point A to Point B” behaviour resumes. This approach is similar to how a tactic is currently written in Osprey. One of the main differences is that subsumption architecture makes the developer break down the behaviour into small components that can be reused. So in the previous example, the "Avoid Objects" behaviour would be reusable for multiple platforms. This approach

could be combined with the API approach to produce a flexible and reusable behaviour-based tactic definition system.

In addition to improving the low-level tactic generation capabilities with Java, it would be useful to tie the scripting language and Java code together. This would allow a non-programmer to specify most of their tactics with high-level scripts. If certain functionality were missing, they could then request a Java programmer to implement the missing functionality and make it available to use in the high-level scripts. Any redesign of the tactics system in Osprey should consider the role of the high-level scripts versus the Java code and how the two should be integrated.

8. Conclusion

This call-up revealed some usability issues and raised questions about how tactics are currently developed in Osprey. There is a large gap between what can be accomplished using the text-based scripts versus using the Java-based classes and methods to create tactics. As well, there are systems, such as the ADS, that would be useful for new scenarios but are not fully implemented in Osprey. These shortcomings have been documented in sections 7, and have been discussed with the SA, as well as Osprey representatives from the Ministry of Defence in the UK. There are interesting possibilities for future work with Osprey, including developing a new tactics definition system, continuing the development of Osprey as an HLA federate, and resolving some unresolved usability issues.

References

[1] Osprey Maintenance and User Support Call-up, Requisition No. W7707-04-2684 against Standing Offer No. W7707-032113/001/HAL

List of symbols/abbreviations/acronyms/initialisms

ADS	Active Dipping Sonar
ADSBP	Active Dipping Sonar Body and Propulsion
API	Application Programming Interface
DND	Department of National Defence
DRDC	Defence Research and Development Canada
GUI	Graphical User Interface
HLA	High Level Architecture
R&D	Research & Development
SOW	Statement of Work
UK	United Kingdom
UMA	Underwater Acoustics Module

This page intentionally left blank.

Distribution list

Document No.: DRDC Atlantic CR 2005-196

LIST PART 1: Internal Distribution by Centre:

6 DRDC Atlantic Library
1 M. Hazen
1 T. Wentzell

8 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 DRDKIM 3

1 TOTAL LIST PART 2

9 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

MacDonald Dettwiler and Associates

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

Unclassified

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

Scenario and Tactics Development for OSPREY 4.3

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

S. Webb and J. Nickerson

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
September 2005	43	1

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DRDC Atlantic/MICS/VCS

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
11BK and CA/UK MOU Annex 1-02	W7707-04-2684

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DN0614: 15 MARCH 2005	DRDC Atlantic CR 2005-196

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The OSPREY 4.3 simulation environment is a very complex tool that assumes the use of JAVA programming to create new tactics. In addition, a new scenario generation process was incorporated in Version 4.0 that is not documented in the previous user guides. This document describes a methodology for creating scenarios and new tactics as a guide to new users. It also documents maintenance work conducted on the software.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Maritime Air; Simulation

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca