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

DRDC | RDDC
technologysciencetechnologie

Integration of Adversarial Behaviour Prediction in COA-T

Laurence Olivier Marion-Ouellet
Michel Mayrand
OODA Technologies Inc.

Prepared by:
OODA Technologies Inc.
4710 rue St-Ambroise, suite 226
Montreal, QC H4C 2C7

PSPC Contract Number: W7707-145677 TA19
Technical Authority: Allan Gillis, DRDC – Atlantic Research Centre
Contractor's date of publication: February 2018

Defence Research and Development Canada

Contract Report

DRDC-RDDC-2018-C058
March 2018

Canada

IMPORTANT INFORMATIVE STATEMENTS

This document was reviewed for Controlled Goods by Defence Research and Development Canada (DRDC) using the Schedule to the *Defence Production Act*.

RISOMIA Call-up 19: Integration of Adversarial Behaviour Prediction in COA-T

Prepared by

OODA Technologies
Laurence Olivier Marion-Ouellet and Michel Mayrand

February 23, 2018

OODA Technologies Inc.
4710 rue St-Ambroise, suite 226
Montréal, QC, Canada, H4C 2C7
Tel: 514-903-4747
www.ooda.ca

Table of Content

1 Introduction

This Call-up investigates the recently acquired BANDIT tool and how it can be integrated into the COA-T environment. BANDIT is an agent-based computational platform, which is designed to evaluate scenarios with an emphasis on modeling different types of illegal behaviour and the interaction between agents. The platform consists of an agent behaviour modelling system and a multi-agent maritime simulator. The platform allows defining a number of scenarios through a simple configuration JSON input in the web interface and it offers the means to run these scenarios in a single or batch mode and evaluate the results as single or aggregate data sets respectively.

1.1 Document Purpose

The objective of this document is to provide information on how the BANDIT platform works and how it can be incorporated it in the COA-T environment. Section 2 focuses on the architecture of both the original BANDIT product and the COAT version created during this call-up. Section 3 covers BANDIT only, its installation, notes, etc. Section 4 performs an analysis of our COAT module for BANDIT. Section 5 is a general user guide for COAT-BANDIT and Section 6 concludes the document.

1.2 Product Scope

The platform can be used to model various contemporary phenomena, primarily focusing on maritime drug smuggling in the Caribbean and in the Eastern Pacific. It can also model the migrant activity in the Mediterranean and estimate areas of high probability of migrant transits.

Here is a partial list of the features (claimed by Blindspot) for BANDIT [Blindspot - 2017]:

- The BANDIT is able to simulate not only water surface vessels and submarines, but also various aerial assets, such as UAVs or reconnaissance airplanes including their endurance.
- The user can also model equipped sensors with their range and probability of object detection.
- The BANDIT platform is able to work with various geographical features such as shore lines, navigable water bodies, etc. The platform also integrates support for meteorological and oceanographic data from various sources such as NOAA or Copernicus.
- The BANDIT platform is designed to simulate up to thousands of agents in one simulation instance. Each agent is simulated individually and inter-agents interactions are possible.
- Once you have prepared your agents in the BANDIT platform, you can execute the simulation in a batch mode (Monte-Carlo).

1.3 Definitions, Acronyms and Abbreviations

AIS: Automatic Identification System
BANDIT: Behavioural Agents for Drug Interdiction
BSM: Behaviour System Model
DB: Database
DRDC: Defence Research and Development Canada
IDE: Integrated Development Environment
MSA: Maritime Situational Awareness
NRL: Naval Research Laboratory

2 Architecture

2.1 Architecture scope

Defence Research and Development Canada (DRDC) Atlantic Research Centre is developing a prototyping framework called the Course of Action Testbed (COA-T) to assist with research into naval planning tools. The CoA-T framework has been architected around a client/server message based communications backbone using elements of the Simulation Interoperability Standards Organization (SISO)/ North Atlantic Treaty Organization (NATO) Coalition Battle Management Language (CBML) and Military Scenario Definition Language (MSDL) standards to define message formats.

The intent of the CoA-T is to facilitate research and demonstration of technologies to support command teams in the development and assessment of courses of action. As such it is expected that the plug and play architecture will support a wide variety of possible tools.

DRDC has access to a simulation of pirate and smuggler behaviour, developed from work initially done by NATO, called BANDIT (Behavioural Agents for Drug Interdiction). BANDIT is configured to work as a server responding to client requests. The scope of this project is to support the implementation of BANDIT on the COA-T local area network and configure COA-T to generate and display BANDIT related data.

2.2 BANDIT Architecture

BANDIT is an agent-oriented time-stepped simulation framework that was created to model maritime piracy and smuggling.

Simulations in this model are run in three steps. First, the scenarios are either randomly sampled from given parameter distributions or directly set as the input. Second, the scenarios are executed while logging important events and trajectories of the agents. Finally, the events are post-processed and output data is created.

The process of simulation execution in BANDIT uses two main components, the environment and the agents. In the environment, the state variables are stored and changed by the commands from the agents. The agents obtain information concerning their surroundings from the environment. The connection between the agents and the environment is mediated by the controller interface.

The BANDIT platform is intended to be run as a web service on the simulation server. It awaits a request to execute a simulation given by the scenario specification. A typical scenario describes Monte Carlo simulation by specifying a number of iterations and which parameters of the simulation are specified as random distributions. From a single scenario multiple simulation instances are sampled and executed. Results are then statistically processed.

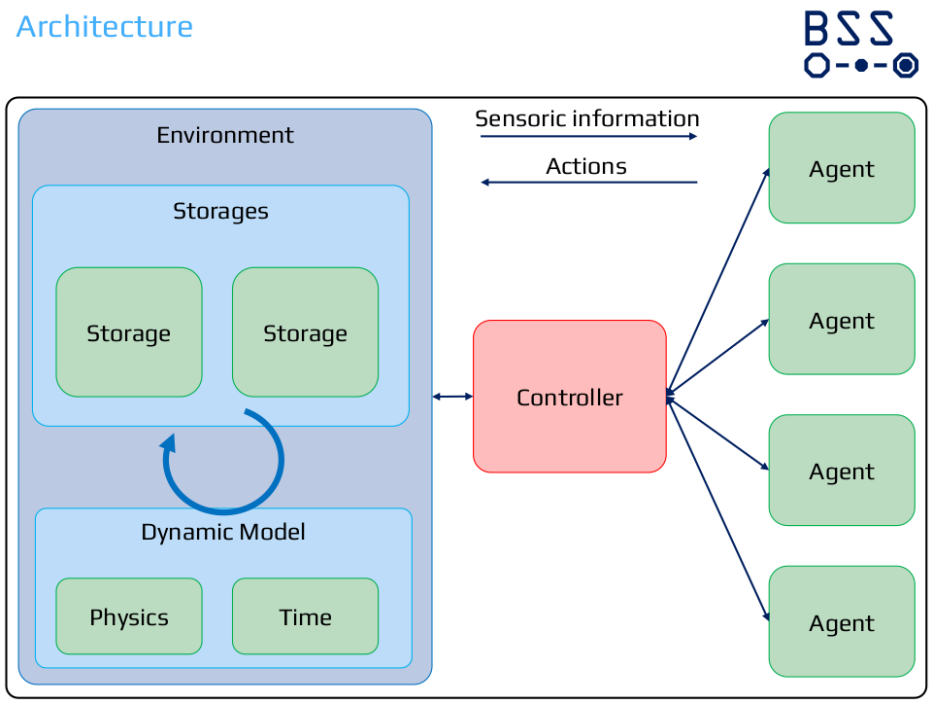

Figure 1 - Bandit platform architecture.

The BANDIT platform consists of two main components: the simulator and the agent behaviour models. The simulator is responsible for simulation of the virtual world, whereas the behaviour models control simulated vessels using artificial intelligence – planning the actions, reacting to the inputs, etc. These components are strictly separated and all communication is through a specified interface. This approach allows a component to be replaced, should the need arise, and to test the behaviour models independent of the simulation.

Figure 1 depicts the main modules of the BANDIT platform. The environment represents physical items in the simulated world. It consists of two components: storages and the model:

1. **Storage**: The storage is an aggregate data structure responsible for keeping the actual state variables of the simulated entities. The state variables describe the physical part of the virtual simulated world, i.e., the environment and the parameters describing the modeled agents.
2. **Model**: The model describes the dynamics of the system. At each simulation step, each model is called with the update of the simulation time. The model is then responsible for updating the storage with new state variables. The variables are updated based on the modeled behaviour, i.e., on the actions of the agent.
3. **Controller**: The controller is an interface between the agent and the environment. It forwards sensory information from the environment to the agent and propagates action calls from the agent to the environment.

For more detailed information on Bandit architecture see [Hrstka-2015A].

2.3 BANDIT/COA-T Architecture

This section describes the steps needed to achieve the following two goals of integrating BANDIT into COAT.

1. Make BANDIT services accessible from the COAT system.
2. Create a proof of concept to check if the integration of external systems is possible with COAT.

Transforming BANDIT from a web service to an ActiveMQ service required the building of a new module, *DRDC-Coat.* Built with a Spring architecture, its main application has essentially a single responsibility at start up: get a connection to ActiveMQ and establish a listener on a topic.

This listener then consumes the messages published to its topic and when it spots a Bandit Request, calls the *server* module to perform a simulation. To accommodate this difference of input types between a JSON sequence containing the whole scenario and an ActiveMQ request, new functions were added to some classes in *server*, performing the same actions as before but using inputs coming from our listener and filling the rest with defaults.

Due to BANDIT's structure, getting the simulation results is a bit harder than simply setting our information in a function's structured output. Blindspot's "modular structure" rather creates threads which (slowly) produce the output. During testing, code that accessed this expected output right at the end of the simulation resulted in error (Since the output was not ready yet).

As a workaround, the consumer starts a simulation and gets an accompanying UUID. It then starts a loop that continuously checks if an output has been produced with a corresponding UUID in our *SimulationRunner*. When it does, this output is sent to a producer as data to be given to the appropriate topic. This loop also has a timeout feature, in case errors happening during the simulation prevent the creation of an output.

As a final note, while it was possible to get BANDIT working through the COAT interface, it is possible that BANDIT itself may not be the best choice. The original kmz outputs it generates are corrupted, its structure is sibylline, and its working scenarios do not use its advertised capabilities. The difference between BANDIT and simply interpolating a smuggler's route (Since *startDate*, waypoints and smuggler's speed all need to be inputted in BANDIT) with a margin of error is the addition of weather data and its effect on smugglers' behaviour. Without a working METOC provider however, BANDIT does not seem to possess any advantage over simple dead reckoning with margins of error, for now.

DRDC might realize that the BANDIT services that are now accessible from COAT may seem narrow when compared to the claims from Blindspot concerning BANDIT (modular, allows simulation of complex behaviours between agents, etc.). However, the product delivered to DRDC does not allow for everything listed.

After compiling and playing around with BANDIT, out of ten or so scenario types, only a single one was found to be in working order. The web service that was delivered to us only allows for a

very specific "nrl_example.json" scenario for a smuggler following waypoints in the Caribbean Sea. All effects of the weather on its route are disabled. Its behaviour or "tarping" (Hiding in the day, full speed during the night) is less appropriate for North American smugglers. Considering those limitations which were discovered during development, it was decided to move away from JSON inputs and to create an easier to understand *BanditRequest* object holding the customisable parameters.

All other examples were tested, with or without modifications to their structure, which resulted in errors and program crashes. It seems that Blindspot was contracted by NRL to provide a very specific type of scenario and modified BANDIT for the specific purpose of their partnership. Anything else was left off and could be obsolete in terms of compatibility.

3 BANDIT User Guide

3.1 Installation

This section contains the instructions for installing and running the original BANDIT service. These instructions were tested in a Linux (Ubuntu) environment but can be replicated on Windows 7, 8 or 10. One can use the Cygwin environment on Windows for executing these instructions.

3.1.1 Prerequisites

The following tools are required before installing BANDIT:

- Java 8:
  - On Linux Ubuntu:
    ```
    sudo apt-get install software-properties-common
    sudo apt-add-repository ppa:webupd8team/java
    sudo apt-get update
    sudo apt install oracle-java8-installer
    export JAVA_HOME=/usr/lib/jvm/java-8-oracle
    ```
  - On Windows: see https://java.com/en/download/faq/java_win64bit.xml
- Maven 3.3.9 or higher
  - On Linux Ubuntu: `sudo apt-get install maven`
  - On Windows: see https://maven.apache.org/install.html
- Scala compiler 2.11.8 or higher
  - On Linux Ubuntu: `sudo apt-get install scala`
  - On Windows: see https://www.scala-lang.org/download/
- Eclipse or IntelliJ
  - **IMPORTANT**: As BANDIT is a mix of Java AND Scala, you will need to add the Scala plugin for Eclipse or IntelliJ.

3.1.2 Unpacking the original BANDIT bundle

First, unzip Bandit:

```
unzip DRDC-bandit.zip
```

BANDIT is composed of three directories:

- **common**: a series of utility libraries created by Blindspot.
- **metoc**: data layer code for retrieving Meteorological and Oceanographic (METOC) data from the US Navy Fleet Numerical Meteorology and Oceanography Center.
- **bandit**: The main webservice application.

Bandit uses some of the Blindspot utility libraries (directory common) and the METOC data layer (directory metoc). These are separate projects, upon which BANDIT depends.

The language content of all the modules from this package can be described as follows:

Table 1: Analysis of Bandit code in terms of coding languages and lines of code

Language	Number of Files	Lines of comment	Lines of code
Java	232	3191	13432
Scala	36	103	1273
Maven	25	101	1673
HTML	64	80	13721
XML	89	0	3557
CSS	5	1	317
Javascript	1	0	31
Total	**452**	**3476**	**34004**

3.1.3 Setting up your Maven environment

All the dependencies (including Scala) can be downloaded through the Maven build process which requires Internet access. If you do not have Internet access and want to compile offline, you will have to copy in your home the entire tree layout of the Maven repository (~/.m2) from a user who had access to Internet and has compiled the code. For the COA-T/BANDIT version, we provided a copy of the M2 Java/Maven libraries necessary for running BANDIT should the instructions below does not work.

BANDIT needs a library for generating KML outputs. This library is forked from the official version (BSD license), which supports Google's track element (trajectories with timestamped points, that Google Earth is able to animate). This library is on the Czech Technical University Maven repository. To access it, we need to use the settings.xml file provided within the BANDIT bundle, which you place into .m2 directory in your home directory:

```
cp settings.xml ~/.m2
```

3.1.4 Compiling the code

To compile the code, use the maven command:

```
cd common
mvn install
```

```
cd ../metoc
mvn install
cd ../bandit
mvn install
```

The compilation will download any Java/Scala library dependencies. At the end of each compilation, all sub-sections in the summary output at the end should say SUCCESS. Before going further, any error should be addressed.

3.1.5 Troubleshooting

Because of the dependency of the Czech Technical University Maven repository, it is recommended keeping a back up (.m2/repository/cz part) in case the repository is not maintained anymore.

In such a case or if you have problems in establishing an internet connection with the Czech Technical University Maven repository, you may encounter the following error:

```
[ERROR] Failed to execute goal on project planners: Could not resolve
dependencies for project cz.blindspot.bandit:planners:jar:1.0-SNAPSHOT:
Failed to collect dependencies at
de.micromata.jak:CustomJavaAPIforKml:jar:2.2.0-SNAPSHOT: Failed to read
artifact descriptor for de.micromata.jak:CustomJavaAPIforKml:jar:2.2.0-
SNAPSHOT: Could not transfer artifact
de.micromata.jak:CustomJavaAPIforKml:pom:2.2.0-SNAPSHOT from/to atg-repo
(http://jones.felk.cvut.cz/artifactory/repo): Not authorized ,
ReasonPhrase:Unauthorized. -> [Help 1]
```

That specific library can be downloaded from:
https://github.com/agents4its/mobilitytestbed/blob/master/mobilitytestbed/lib/CustomJavaAPIforKml-2.2.0.jar

Create the following directory path:

```
~/.m2/repository/de/micromata/jak/CustomJavaAPIforKml/2.2.0-SNAPSHOT
```

and manually install the library in your local Maven repository. Copy the JAR file in that location, rename it CustomJavaAPIforKml-2.2.0-SNAPSHOT.jar and create in the same directory a metadata file which contains the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata modelVersion="1.1.0">
  <groupId>de.micromata.jak</groupId>
  <artifactId>CustomJavaAPIforKml</artifactId>
  <version>2.2.0-SNAPSHOT</version>
  <versioning>
    <snapshot>
      <localCopy>true</localCopy>
    </snapshot>
    <lastUpdated>20171015215114</lastUpdated>
    <snapshotVersions>
      <snapshotVersion>
```

```
        <extension>jar</extension>
        <value>2.2.0-SNAPSHOT</value>
        <updated>20171015215114</updated>
      </snapshotVersion>
      <snapshotVersion>
        <extension>pom</extension>
        <value>2.2.0-SNAPSHOT</value>
        <updated>20171015215114</updated>
      </snapshotVersion>
    </snapshotVersions>
  </versioning>
</metadata>
```

3.2 Running BANDIT

3.2.1 Prerequisites

To run BANDIT, it is recommended to use a development tool such as Eclipse or IntelliJ. In both cases, you will need to install the Scala plugin in order to process the Scala code. Using an IDE creates an easier process when trying to catch unsatisfied dependencies, but it is also possible to take advantage of Maven's lifecycle to build an executable jar file. To do so, maven commands "mvn clean package" will need to be run in the server subfolder. The approach taken by OODA was rather to use the IDE, which grants more insight into BANDIT's structure.

For IntelliJ: In the Configure Menu, select Plugins, then, click on Browse JetBrains Plugins. In the search field, type Scala, in the result list, select Scala and click the Install button on the right side. Click the Restart IntelliJ IDEA button.

For Eclipse: Download the fully configured IDE for Scala at http://scala-ide.org/

3.2.2 Import the Maven code in the IDE

For IntelliJ: Create a project form existing sources. Select the *DRDC-Bandit/bandit* directory. Select the radio button for *Import project from external model* and below, select *Maven*. Click Next. In addition to the default selections, select the *Search for projects recursively* and *Import Maven projects automatically*. Select Next three times. In the SDK panel, select one (JDK 1.8) if not already done. Click Next. Click Next and Finish.

3.2.3 Starting the BANDIT server

For IntelliJ: Select the menu Run->Run…->Edit Configurations. Deploy the left Default list and select Application. In the right panel, input the following entries:

- Main class: cz.blindspot.bandit.server.Webservice
- VM options: -Dcz.blindspot.bandit.configuration="/$YOUR_INSTALLATION_DIR/DRDC-bandit/bandit/local.conf"
- Working directory: $YOUR_INSTALLTION_DIR/DRDC-bandit/bandit
- Use classpath of module: select the server directory
- JRE: Select the Default

Above the bottom box, click the "+" sign and add "Build".

Finally, click "Run". The output of the run will be displayed at the bottom of the IntelliJ window. Resolve any errors that are displayed in that area.

If an error says that the port number is already in use, modify line 21 of file:
`server/src/main/java/cz/blindspot/bandit/server/Webservice.java`
and change the port number 8080 for another which is free and restart the application.

Uncomment line 14 of files:
`server/src/main/java/cz/blindspot/bandit/server/IssueSimulationServlet.java`
`server/src/main/java/cz/blindspot/bandit/server/DownloadResultServlet.java`
and add
`import javax.servlet.annotation.WebServlet;`
in the top import section of each file and restart the server.

3.2.4 How to test the server

The BANDIT web service is used in two different ways. The first is to serve as an API interface using the GET command. The second is to provide a simple web application for inputting the information.

To test the server, open a browser and enter the link:
http://0.0.0.0:8080/simulation/json-input.html

You should now see the BANDIT interface for inputting simulation parameters.

3.3 Running the BANDIT examples

Note that this section concerns the unmodified original BANDIT program. Some features have been modified or disabled in order to accommodate delivery of a jar for simulation in the North Atlantic zone.

The first modification is to disable the WaterPolygons. While they provide useful information about the shorelines, necessary to make accurate simulations, their data is absent for zones outside the Mediterranean and Caribbean seas and cause errors for examples in the North Atlantic.

To fix this issue, calls to the class *WaterPolygonProvider*'s *isInWater* function now simply return true no matter the position instead of performing a check against its data, which resulted in *NullException* for positions outside of its zones. Therefore, due to its absence of data for zones out of the Carribean and Medditerenean seas, we had to disable the *WaterPolygon* service.

It is also necessary to allow outputs to be created in North America. In the post-processing module, /src/main/java/cz/blindspot/bandit/postprocessing/HeatGrid, a new bounding box for North America can be created (private static final BoundingBox NORTHAMERICA_BOUNDING_BOX = BoundingBox.fromEsriCoords(38, 285, 40, 66, 0.5, 0.5);) and used in the HeatGrid.create function.

3.3.1 Scenario List

The current BANDIT module responsible for the web service, "server", only works for scenarios following the general structure of an NRLScenario: a boat, typically of type "GoFast", follows an itinerary. The boat goes at full speed during the night and stops during the day, covered by a blue tarp in order to avoid detection. Using BANDIT to predict the movements of drug smugglers using such a strategy made sense; due to wind and current effects, the position of the smugglers can be estimated. The scenarios were built by accessing structures of shorelines describing where boats could go in the Caribbean (A boat going through an island does not make much sense). Some possible features were the addition of fishermen on the sea and regular patrols by the coast guard or police. The trajectory of the smugglers would be modified by these other boats.

A complex example of this would be where a smuggler's boat sees a fisherman and becomes attracted to it in order to query information regarding the police's whereabouts. The smuggler would then avoid the area where the police is present.

However, the only working NRL scenario that was implemented in the server seems to only focus on a single boat following a series of waypoints. The weather information is also not considered due to the deactivation of the METOC provider by Blindspot Solutions.

The other BANDIT scenarios (Migrants going through the Mediterranean or smuggler being tracked by a surveillance drone) do not work on the web interface. It seems that the web interface could also accept scenarios of type "Transshipment", where two smugglers exchange cargo, but traces of how to build such a scenario are absent.

3.3.2 How to Run an example

After starting the web server (`server/src/main/java/cz/blindspot/bandit/server/Webservice.java`, main function), an input text box appears at address http://0.0.0.0:8080/simulation/json-input.html, where you can copy/paste the nrl_example (The only one working) scenario itself. The parameters of a NRLScenario do not follow BANDIT's own template (available at `DRDC-bandit\bandit\scenarios\src\main\resources\schema\scenario-schema.json`), meaning the exact nature of every input parameters stays nebulous, as described by BANDIT's phase 1 report: "*Unfortunately the specifications are not released to public, hence we cannot include them in this report*.". This is probably due to a desired compatibility with inputs from NRL, which could be restricted/classified.

Notwithstanding these problems, a simple nrl_example for a *GoFast* boat was present, in folder `DRDC-bandit\bandit\scenarios\src\main\resources`. The "GF" in *targetType* indicates what type of vessel is simulated.

Copy-pasting it into the text box of the web page and pressing the start button will begin the simulations. When they are done, the page displays a clickable link to download the final results (this feature is broken in the version of Bandit supplied). To access the results, one needs to go in `DRDC-bandit\results\` and find the appropriate folder according to the simulation timestamp. In this folder, an `output.kmz` file is present (Which will be corrupted, possibly due to changes in

the kmz structure/libraries since the release of bandit) and an `output.json` in zipped format. If the simulation is unsuccessful, both these files will be absent, replaced by an error text file.

It is also possible that a simulation can execute correctly but with an output full of missing data (Described by a -999 value). This problem was encountered when trying to run waypoints in the Northern Atlantic, since the expected area of operations was the Caribbean. The software therefore tried to input North Atlantic data into a totally incompatible grid resulting in the missing data in a grid. The final version delivered by OODA replaced this inadequate Caribbean zone by one covering the North Atlantic, much more useful for Canadian applications.

3.3.3 Issues with BANDIT

Following the reading of the phase 1 report by the BANDIT developers [Hrstka-2015B], it seems BANDIT is intentionally obfuscated. No real user guide, no readme, no comments in the code and only a single runnable example. It is possible that such difficulties are due to an agreement between Blindspot and NRL or because Blindspot is used to building the scenarios as a service to clients rather than delivering BANDIT as an actual product ready to be used.

3.4 The METOC Data Layer

3.4.1 Description

METOC was implemented in BANDIT to give accurate information about the weather, in order to influence the behaviour of smugglers or other agents. However, due to deactivation of the provider, a "dummy" METOC provider is currently used, which seems to offer no wind and no wave without regard to the actual climate.

3.4.2 Development status

The web API was tailored for the need of the first user of BANDIT, namely NRL. However BANDIT was designed as a modular application, so the server module is actually quite thin. So if we need our own web API, we can create a new module with an API that would suit our use case (using for example *Dropwizard* framework to set up the web layer). Some examples can be found in the demo-scenarios module.

Since the primary customer for Bandit was NRL, the *MetocProvider* was designed to be used with METOC server. The METOC server is an application that was running on the Czech Technical University side and was periodically downloading METOC forecast from the NRL. BANDIT used this information to run the simulation. Because the project is now closed, the METOC server is now useless. We have also connected to Copernicus service, but it was not fully integrated.

The creators of BANDIT have extracted an interface *MetocProvider* and created *DummyMetocProvider* which allows the code to compile and run without the METOC data (no METOC output from BANDIT). In order to make BANDIT work with METOC data, you need to implement your own *MetocProvider*.

4 BANDIT/COA-T Design

4.1 Modifications to BANDIT

In order to accommodate scenarios where the waypoints were located in the North Atlantic, some modifications had to be made. One the first problems encountered was the absence of *WaterPolygons* (Shoreline data). These therefore had to be disabled. Avoiding land masses therefore becomes the job of wisely choosing waypoints. If one wishes, it could be possible to try to recreate WaterPolygon data from OpenStreetMap map files, which seems to be the original source. The process Blindspot used to transform the OSM files into a code compatible structure was however not investigated in this callup.

Some modifications also had to be applied to the dummy METOC provider. The original service was built in order to check different sources depending on the position of the simulation. Since the North Atlantic was never considered as a zone of operations, the dummy provider did not cover it, which has been fixed.

Of course, one of the main modifications brought to BANDIT was the addition of a module, responsible for integration in the COAT system. Its main systems are a consumer and a producer.

The consumer is responsible for listening to any BANDIT request message on an ActiveMQ topic and to start the simulation runs with the inputs provided in the message's body. The producer publishes the answers that are extracted from the simulation results to ActiveMQ.

In order to provide for more efficient testing, an input system for scenarios has been created. The expected input message is a JSON-serialized *BanditRequest* object. This means that both JSON strings and Java Objects will be accepted as inputs, allowing for greater versatility.

4.2 Data model for the input

Table 1: ActiveMQ Data Model for Inputs

JMS Part	Name *(String)*	Value *(Object)*	Value Type	Comment
Header	**JMSType**	**"**BANDIT_REQUEST**"**	**String**	
Properties	**RequestId**	**Unique ID of the request**	**String**	
Body (JSON serialized BanditRequest java object)	**speed**	**Speed in knots**	**double**	
	speedUncertainty	**Speed uncertainty in knots**	**double**	
	startDate	**TimeStamp of the start of the simulation**	**String**	**Format: YYYY-MM-DDTHH:MM:SSZ**
	startUncertainty	**Uncertainty on the startDate, in hours**	**int**	

	missionLength	**Unknown, in hours**	**int**	
	missionLengthUncertainty	**Unknown, in hours**	**int**	
	waypoints	**Expected route of the smuggler**	**List<LatLon>**	**An array of the waypoints**
	uncertainties	**Uncertainty surrounding the exact position of the waypoints, in nautical miles(?)**	**List<double>**	**An array of the uncertainties**

Table 1 features the input parameters for a *BANDIT_REQUEST* message, published on the topic *C2.REQUESTS*. While some parameters are obvious, others like *missionLength* and its uncertainty are not clear at first glance and could very well simply not be considered in an *NRLScenario*, despite their presence in the original example.

Deeper digging through the code shows that *missionLength* is at least not used as an upper bound on the simulation's timetable. It is automatically set to its default of 10 days. Another possibility (hard to explore due to the code's structure) is that this parameter is supposed to represent the vessel's autonomy. Indeed, in some scenarios on Blindspot's website, BANDIT's advertised capabilities include the ability to simulate the behaviour of migrant's boats without enough fuel trying to traverse the Mediterranean Sea. Necessary to these simulations is the presence of a factor describing how long a boat can run before running out of fuel which could possibly be our *missionLength*. This is, however, only a hypothesis.

4.3 Data model for the output

Table 2: ActiveMQ Data Model for Output

JMS Part	Name *(String)*	Value *(Object)*	Value Type	Comment
Header	**JMSType**	**"**BANDIT_ANSWER**"**	**String**	
Properties	**RequestId**	**Unique ID of the request**	**String**	**This RequestId comes from the Request message**
Body (JSON formatted String)	**"0"**	**Probability grid at time 0**	**JSON heat grid**	**Represents the startDate**
	"3"	**Probability grid at time 0 + 3 hours**	**JSON heat grid**	
	"t"	**Probability grid at final time t**	**JSON heat grid**	**t being a multiple of 3.**

The output *BANDIT_ANSWER* messages, visible at Table 2, are currently being published on topic *C2.REQUESTS_DATA* and are built as an array of heatgrid elements. These elements possess the following data model:

- A description of the grid's location (*LowerLat*, *LowerLon*, and number of Lat and Lon elements and their size)

- The time it covers, as a timestamp
- The grid itself (Currently 66 longitudes per 40 latitudes) filled with a probability.

It is important to know that the probabilities being talked about here are not probabilities in a statistical sense and do not sum to 1 (A decision made by Blindspot). Therefore, this value should me more used as a general relative reference in order to evaluate where a boat might be.

This HeatGrid is built from the results of a batch of simulations which vary the parameters taken as inputs according to their uncertainties. As time advances, the smugglers in some of these simulations will have reached their destination and are therefore removed from the batch. When all of them have arrived, BANDIT outputs its equivalent of *missingData*: -999, rather than probabilities.

4.4 Monitoring

According to the ActiveMQ Design Document for COAT, each system incorporated into this Testbed must possess a HeartBeat service, in order to allow monitoring. COAT-BANDIT therefore has a `ScheduledExecutorService` scheduler. Its role, at startup, is to make sure the task "*beat*" is done every 10 seconds. This task is defined as a call to the function `Producer.SendHeartBeat()`. It is in this function that the heartbeat `MapMessage` itself is created and sent to the topic *SYSTEM.STATUS*.

Its properties are the following:

Table 3: ActiveMQ Data Model for HeartBeat

JMS Part	Name *(String)*	Value *(Object)*	Value Type	Comment
Header	**JMSType**	**"HEARTBEAT"**	**String**	
Contents	**UPTIME**	How long has the app been running (In seconds)	**Double**	
	APP_NAME"	"DRDC-BANDIT"	**String**	
	CLIENT_ID	"d55d2e20-0285-4f1e-9a08-713438b76c26"	**String**	Currently hardcoded in the function
	HOST_NAME	Name of the machine running the application	**String**	
	HOST_IP	IP of the machine running the application	**String**	
	APP_INFO	""	**String**	
	PUBLISHED	Expected value: "C2.REQUESTS"	**String**	
	SUBSCRIBED	Expected value: "C2.REQUESTS_DATA"	**String**	
	STATUS	"NORMAL"	**String**	No error messages have been implemented, yet

5 BANDIT/COA-T User Guide

5.1 Requirements

Since the deliverable is a single executable JAR containing all dependencies, the only requirement is java 1.8. To start the COAT-BANDIT:

1. Make sure your ActiveMQ service is active and that the port 8079 is free. (Can be modified in DRDC-bandit/bandit/server/src/main/java/cz/blindspot/bandit/server/ Webservice.java)
2. The dependencies needed for the project are the same as the ones described in section 3.1.1.
3. If you don't have ActiveMQ installed, a simple implementation is a docker instance (if you have Docker installed):
    a. docker pull webcenter/activemq:latest (to be done only once)
    b. docker run --name='activemq' -it --rm -e 'ACTIVEMQ_MIN_MEMORY=512' -e 'ACTIVEMQ_MAX_MEMORY=2048' -P webcenter/activemq:5.14.3

5.2 Starting the COAT-BANDIT

5.2.1 By recompiling

1. Go to DRDC-Bandit/bandit/DRDC-Coat/src/main/resources and open the *application.properties* file. Modify the values in order to reflect the topics used, ActiveMQ IP + port and its login information.
2. Compile the following:
    a. In `DRDC-Bandit/common: mvn clean install`
    b. In `DRDC-Bandit/metoc: mvn clean install`
    c. In `DRDC-Bandit/bandit: mvn clean install`
    d. In `DRDC-Bandit/bandit/DRDC-Coat: mvn clean package`
3. Launch the program: `java -jar DRDC-Coat/target/DRDC-Coat-1.0-SNAPSHOT.jar`
4. COAT-BANDIT is now waiting for an input request.
5. Use the BanditRequester Eclipse project to send an input. Outputs will be sent to the topic specified in application.properties.

5.2.2 Using the delivered jar

1. Go to the directory containing the .jar file in a command window.
2. Type `java -jar DRDC-Coat/target/DRDC-Coat-1.0-SNAPSHOT.jar` and press enter.
3. In order to use different settings than the default values (Concerning ActiveMQ IP:Port, login information and the topic to use), it is possible to do so without recompiling. Please check Annex B for such instructions.

5.3 Developer guide

5.3.1 BanditRequester

The BanditRequester project has been created to facilitate the sending of ActiveMQ requests to BANDIT. It contains a *BanditRequest* class, which is the same as the one present in the DRDC-BANDIT module. In the main function, such a *BanditRequest* is generated and filled with inputs. The final instruction is then to send it to the producer which serializes this java object before sending it to the topic.

The BanditRequester also features a consumer, which listens on the expected topic of the BANDIT outputs. This allows us to confirm that the simulation is complete and has outputted valid results on the appropriate topic. A second consumer also exists, which listens to the exact requests the BanditRequester sends out. This can prove useful if one wishes to emulate *BanditRequest* objects without going through the actual creation of a java object. Annex A has more details on the structure.

5.3.2 Final note about COAT-BANDIT

BANDIT cannot currently be considered a finished product. While it promises a modular architecture and the ability to simulate different scenarios, actually making these run without any information appears to be a time-consuming task, made harder by the absence of help from the original developers.

6 Conclusion

6.1 Bandit as a Coat integration test

While the obstacles to make BANDIT work were many, which therefore had an impact on the COAT integration, this code delivery and report should not discourage DRDC from trying to add other services into COAT.

Java applications with great documentation on the necessary structures to run it are the perfect candidates to be integrated into COAT. BANDIT did not fit this description. Another difficulty encountered was the continued use of JSON objects in the code. Rather than parse the JSON once and create workable java objects, the JSON objects are kept, manipulated and often transformed into *InputStreams* making it hard to modify or insert values. This decision was probably made to keep BANDIT flexible, but when only a single example is working this can be considered a moot point.

In conclusion, BANDIT has been successfully integrated into the COAT structure, but with possibly disappointing results due to the original condition of BANDIT. Nonetheless, this call-up can be used as a learning experience for choosing future services to include in COAT.

Annex A: Bibliography

[Hrstka-2015A] HRSTKA, O., et al. Agent-based Approach to Illegal Maritime Behavior Modeling. Scientific Journals of the Maritime University of Szczecin. 2015, 42(114), 101-111. ISSN 2392-0378. Available from: http://repository.am.szczecin.pl/handle/123456789/754

[Hrstka-2015B] HRSTKA, O., et al. BANDIT (Behavioral Agents for Drug Interdiction)
Phase 1 Report. Internal Report for NRL. 2015. CAGE: 3D97G, DUNS: 361049704

[DRDC Halifax-2017] DRDC Halifax, COAT ActiveMQ Design – v1.4, 01DB-PSA Design Document – ActiveMQ, Internal Report for COAT Development, 2017.

[Blindspot-2017] Blindspot-Solutions, Bandit – Platform Overview, Retrieved from http://blindspot-solutions.com/bandit/

Annex B: Application.properties sample and configuration for COAT-BANDIT

Here an example for the application.properties file of COAT-BANDIT is available. It is located in `DRDC-BANDIT/bandit/DRDC-Coat/src/main/ressources`

```
jsa.activemq.topic=C2.REQUESTS
jsa.activemq.broker.url=tcp://localhost:61616
jsa.activemq.topic.answer=C2.REQUESTS_DATA
jsa.activemq.broker.username=admin
jsa.activemq.broker.password=admin
spring.jms.pub-sub-domain=true
Logging.level=DEBUG
```

If one wishes to use different values than the ones provided above, it is possible to override these when starting the jar application, by inserting them as environment variables, as so:

```
java -jar DRDC-Coat-1.0-SNAPSHOT.jar --jsa.activemq.broker.url=tcp://192.168.99.99:61616
--jsa.activemq.topic.answer=C2.OTHER_OUTPUT_TOPIC
```

Annex C: Data example for BanditRequest inputs

Example of an input BanditRequest message, serialized

properties = {_type=ca.drdc.coat.model.BanditRequest, requestID=8d8d2980-723b-4263-8d37-bcd5edc7a64d}, readOnlyProperties = true, readOnlyBody = true, droppable = false, jmsXGroupFirstForConsumer = false, text = {"startUncertainty":72,"missionLength":48,"missionLength" : ...T15:27:04Z"}}

As one can see, the actual message is contained in the "text" portion of the ActiveMQ message. The _type property specifies to the consumer that the object contained here must be deserialized by this particular class. This structure is due to the nature of JMS producers and consumers.

One could recreate these messages by building *ActiveMQTextMessages* and setting the text to the desired inputs and setting the _type and requestID properties before sending the message.

Annex D: Original NRLScenario example


```
{
  "name": "Case3",
  "classification": "Official Use Only (Cat. 3, Export Control/ITAR)",
  "id": "4a120a68-17d3-441b-8d4f-0018039449e4",
  "missingData": -999,
  "targetType": "GF",
  "targetValue": {
    "value": 1000,
    "uncertainty": -999
  },
  "confFactor": 4,
  "targetVelocityKnots": {
    "value": 30,
    "uncertainty": -999
  },
  "targetSubmergedHoursPer24": {
    "value": 0,
    "uncertainty": -999
  },
  "startDate": {
    "value": "2015-04-01T16:27:04Z",
    "uncertainty": 72
  },
  "waypoints": [
    {
      "lat": 9.032112309823082,
      "lon": -76.79953497719063,
      "uncertainty": 20
    },
    {
      "lat": 10.014153613930346,
      "lon": -78.18922587992964,
      "uncertainty": 20
    },
    {
      "lat": 10.154453461144609,
      "lon": -79.6807851101996,
      "uncertainty": 20
    },
    {
      "lat": 9.528798850049062,
      "lon": -81.06918142623961,
      "uncertainty": 20
    },
    {
      "lat": 11.284793517763562,
      "lon": -82.98166891504986,
      "uncertainty": 20
    },
    {
      "lat": 13.722593710261787,
```

```
      "lon": -82.99234709725732,
      "uncertainty": 20
    }
  ],
  "blueTarpCONOP": 0,
  "iwedge": 1
}
```

DOCUMENT CONTROL DATA

*Security markings for the title, authors, abstract and keywords must be entered when the document is sensitive

1. ORIGINATOR (Name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or tasking agency, is entered in Section 8.)

OODA Technologies Inc.
4710 rue St-Ambroise, suite 226
Montreal, QC H4C 2C7
Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

CAN UNCLASSIFIED

2b. CONTROLLED GOODS

NON-CONTROLLED GOODS
DMC A

3. TITLE (The document title and sub-title as indicated on the title page.)

Integration of Adversarial Behaviour Prediction in COA-T

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

Marion-Ouellet, L.O.; Mayrand, M.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.)	6b. NO. OF REFS (Total references cited.)
March 2018	24	4

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter.)

Contract Report

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

DRDC – Atlantic Research Centre
Defence Research and Development Canada
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7
Canada

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.)
01db	

10a. DRDC PUBLICATION 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.)
DRDC-RDDC-2018-C058	

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

Behaviour Prediction; Course of Action; BANDIT; COA-T

13. ABSTRACT/RESUME (When available in the document, the French version of the abstract must be included here.)