Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	517737

TITLE

Integration of Advanced Models in the CF-18 Air Combat Evaluator - Example of a Missile Representation

System Number:

Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to: CL

THIS PAGE IS LEFT BLANK

THIS PAGE IS LEFT BLANK

DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH
(MARITIME, LAND & AIR)

DOR(MLA) RESEARCH NOTE RN 2002/07

INTEGRATION OF ADVANCED MODELS IN THE CF-18 AIR COMBAT EVALUATOR – EXAMPLE OF A MISSILE REPRESENTATION

by

Y. GAUTHIER

JULY 2002

OTTAWA, CANADA

National Defence Défense nationale

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

REPRODUCTION QUALITY NOTICE

This document is the best quality available. The copy furnished to DRDCIM contained pages that may have the following quality problems:

: Pages smaller or Larger than normal

: Pages with background colour or light coloured printing

: Pages with small type or poor printing; and or

: Pages with continuous tone material or colour photographs

Due to various output media available these conditions may or may not cause poor legibility in the hardcopy output you receive.

☒ If this block is checked, the copy furnished to DRDCIM contained pages with colour printing, that when reproduced in Black and White, may change detail of the original copy.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF AIR STAFF OPERATIONAL RESEARCH

DIRECTORATE RESEARCH NOTE RN 2002/07

INTEGRATION OF ADVANCED MODELS IN THE CF-18 AIR COMBAT EVALUATOR – EXAMPLE OF A MISSILE REPRESENTATION

by

Y. GAUTHIER

Recommended by _____ DASOR

Approved by _____ DOR (MLA)

Directorate Research Notes are issued by directorates They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication They may record development work done in support of sponsored projects which could be applied elsewhere in the future As such they help serve as the corporate scientific memory of the directorates

OTTAWA, ONTARIO JULY 2002

ABSTRACT

The CF-18 Air Combat Evaluator (CF-18 ACE) is a computer-based simulation developed in Java that enables investigation of few-on-few fighter engagements. It includes representations of the aircraft platform, aircrew, data link, sensors, and weapons. This research note explains how new models should be integrated in the CF-18 ACE. It also explains how this integration can be achieved when the models are coded in a language other than Java, such as C or C++. The example of a high-fidelity missile model recently incorporated in the CF-18 ACE is used to illustrate how the integration can be done in practice.

RÉSUMÉ

L'Évaluateur de Combats Aériens (ECA) de CF-18s est un logiciel de simulation développé en Java qui permet l'étude de combats entre petits groupes d'avions de chasse. Le logiciel comprend des modèles représentant les platte-formes, les équipages, la transmission de données, les senseurs, et l'armement. Cette note de recherche explique comment de nouveaux modèles devraient être intégrés à l'ECA. Elle explique aussi comment l'intégration de modèles écrits dans un langage autre que Java, tels que le C ou le C++, peut être accomplie. L'exemple d'un modèle de missile de haute-fidélité ayant été récemment ajouté à l'ECA est utilisé pour illustrer comment ce genre d'intégration peut être réalisée en pratique.

TABLE OF CONTENTS

PAGE

ACKNOWLEDGEMENTS

The author would like to acknowledge the collaboration of the Precision Weapons Section of Defence R&D Canada in Valcartier, especially Mr. Alfred Jeffrey and Mr. Marc Lauzon who provided DASOR with their missile model. The author is also grateful to Mr. Stan Isbrandt for fruitful discussions about this work.

INTEGRATION OF ADVANCED MODELS IN THE CF-18 AIR COMBAT EVALUATOR (ACE) – EXAMPLE OF A MISSILE REPRESENTATION

I. INTRODUCTION

Background

1. The CF-18 Air Combat Evaluator (CF-18 ACE) is a computer-based simulation developed by the Directorate of Air Staff Operational Research (DASOR) that enables investigation of few-on-few fighter engagements [1,2]. It was developed to provide support to the Director of Aerospace Requirement (DAR) on evaluating various weapon and avionics upgrade options for the CF-18 fighter aircraft through a wide range of combat studies.

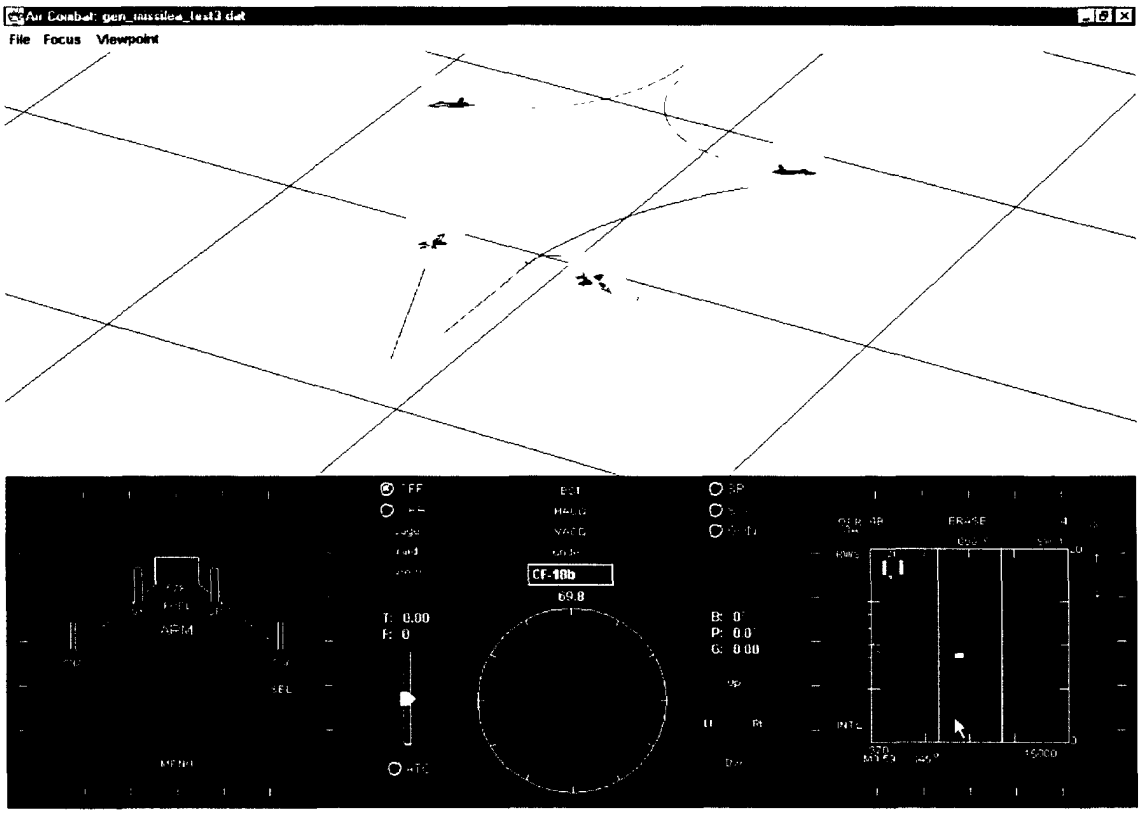

Figure 1: Graphical user interface of the CF-18 ACE.

2. The graphical user interface of the CF-18 ACE is presented in Figure 1. The software can be run as a stand-alone simulation or as a man-in-the-loop simulation with humans controlling red and blue forces. It includes representations of the aircraft platform, aircrew, data link, and weapon systems. It also has different sensor models for airborne detection, such as radars (APG-65 and APG-73), radar warning receiver, and airborne warning and control systems (AWACS).

3. Originally written in Smalltalk, the CF-18 ACE was ported to Java™ in its second version in order to access richer developmental environments, ensure its long-term viability, and facilitate external support during its development. Java is an object-oriented language that permits a high level of abstraction and modular designs through the utilization of objects. One advantage of Java is its Java Native Interface (JNI) that enables the integration of code written in other languages such as C or C++ with the code written in Java [3]. This feature is particularly interesting for the CF-18 ACE, because it makes possible the incorporation of native models developed by DND research labs, contractors, or allied countries.

4. The JNI was recently used to integrate a high-fidelity representation of a short-range air-to-air missile (SRAAM) into the CF-18 ACE. This missile model was developed by the Precision Weapons Section of Defence R&D Canada in Valcartier (DRDC-V). The main features of the model include [4]:

- Six degrees-of-freedom (6DOF) aerodynamics;
- Representations of the guidance, propulsion, and control systems;
- Mass properties changing as a function of time;
- Kinematic relationships between fighter, target and missile;
- Self-tuning autopilot for performance and stability.

5. The integration in the CF-18 ACE of a new missile model with such a level of complexity was made necessary because some combat simulations require a higher degree of realism. For instance, a forthcoming DASOR study will be the investigation of new rules of engagements for the new SRAAM that will be acquired by the air force. In this type of analysis, the validity of the missile model is critical to obtain trustworthy results. The fact of using a SRAAM model that was developed, verified, and validated by domain experts will substantiate considerably simulation results produced with the CF-18 ACE.

Overview

6. The aim of this research note is to demonstrate how high-fidelity models can be integrated into CF-18 ACE. Its content is rather technical and is mainly directed towards those who will maintain and further develop CF-18 ACE, and need a better understanding of the JNI. A part of the content is also directed towards the requirements of operational research analysts who will use the CF-18 ACE and need to understand how it was developed, and how it can be improved.

7. The following section describes the software structure of the CF-18 ACE and explains, in general terms, how to proceed to integrate a new model inside it. The example of DRDC-V's missile model is used to illustrate how the JNI can be used in certain situations to achieve this task. Annexes

A, B, and C give additional details on the utilization of the JNI, with samples of the code used to link the missile model with the CF-18 ACE. The example of a CF-18 ACE scenario data file using the missile model is given in Annex D.

II. INTEGRATION OF MODELS IN THE CF-18 ACE

Structure of the CF-18 ACE

8. The CF-18 ACE has an internal object-oriented structure. This means that the software was written in a modular fashion through the use of objects. Each object represents either a concrete entity (e.g., an aircraft) or an abstract entity (e.g., a manoeuvre) and has associated with it properties and behaviour.

9. In an object-oriented program, the object properties and behaviour are defined within a *class*. The *class variables* define the properties of the object, while the *class methods* are associated with the object's behaviour. For example, CF-18 ACE's class called *Platform*, which is related to the aircraft model, contains a variable called *inclination* representing the inclination of the aircraft, and methods called *pitchUp* and *pitchDown* that change the aircraft inclination. The advantage of object-oriented programming is that the objects allow data to be encapsulated with the methods manipulating that data.

10. Another important feature of object-oriented programming is *class inheritance*. Inheritance consists of the derivation of a parent class in one or more subclasses. A subclass is used to further refine the definition of the parent class. It contains all the members (i.e., the variables and methods) of its parent class, in addition to its own members. A subclass, however, may redefine methods of its parent class thereby overriding the inherited behaviour.

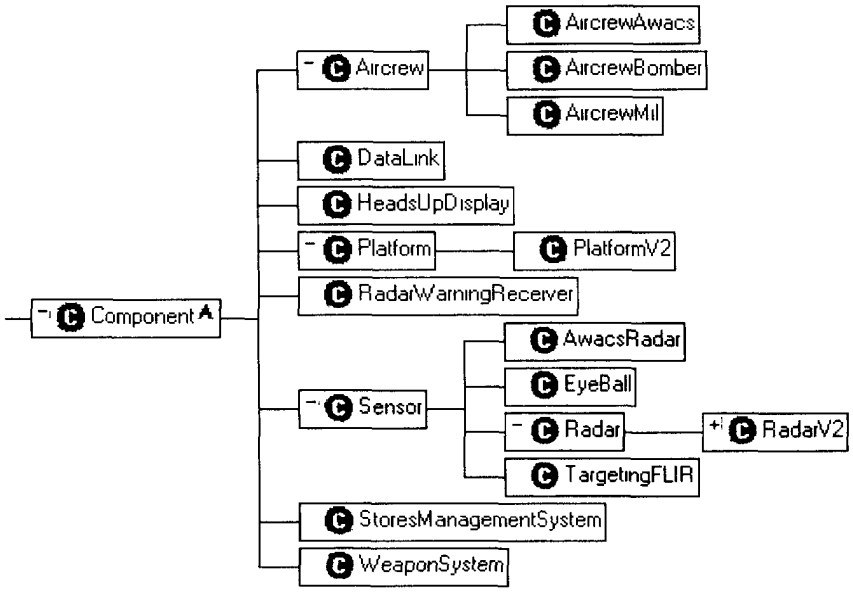

Figure 2. Class inheritance for the aircraft component models.

11. The concept of inheritance is illustrated in Figure 2, which presents the part[1] of CF-18 ACE's class hierarchy that defines the aircraft components. For instance, all sensor models (highlighted in yellow on the figure) inherit the properties of the basic *Sensor* object. This object contains the essential variables of any sensor representation (range, field of view, scanning rate, etc). The other sensor models inherit these variables, and bring additional variables that are required only by them. They also bring new methods, or override methods of the *Sensor* class. A good example of overriding is the initialisation method *initForFile,* which exists in the *Sensor* class and all its subclasses, but differs from one class to another.

Integrating a Java model

12. The CF-18 ACE already includes representations of the most critical components of a fighter aircraft, such as the platform, sensors, aircrew, and weapon systems. It also includes representations for different types of units, such as aircraft, missiles, and semi-automated missiles. In general, a better representation of a component or unit should be introduced as an extra level in the hierarchy.

13. Accordingly, if the model to be incorporated is written in Java, its integration can be easily achieved by creating a new class at the appropriate place in the software structure and inserting the model's Java code within this class. The new model itself can be a combination of several Java classes, but only one of them has to be directly linked with the rest of CF-18 ACE's code.

Integrating a C/C++ model

14. If the model to be integrated is not written in Java, but in C or C++, then the JNI must be used. Essentially, a shared library has to be built from the C/C++ code, and a new Java class must be created to interface the shared library with the CF-18 ACE. This is what has been done for the missile model developed by DRDC-V; the model was first developed using the Matlab/Simulink® simulation environment, and then converted to a standard C++ shared library that can be used by the CF-18 ACE, as shown in Figure 3. A new Java class was inserted in the CF-18 ACE, but the code it contains is there only to link the CF-18 ACE with the C++ library.

15. This approach is more complex, but has some advantages. One of them is that the model's code is hidden in the library. Some code must be written to interact with the library, but the essence of the model is not modified during the integration. When the developer of the model issues a new version, it can be simply integrated in the CF-18 ACE by replacing the library. On the other hand, because the library acts as a black box, the validity of the model must be ensured.

---

[1] The whole class hierarchy of the CF-18 ACE currently contains 80 classes

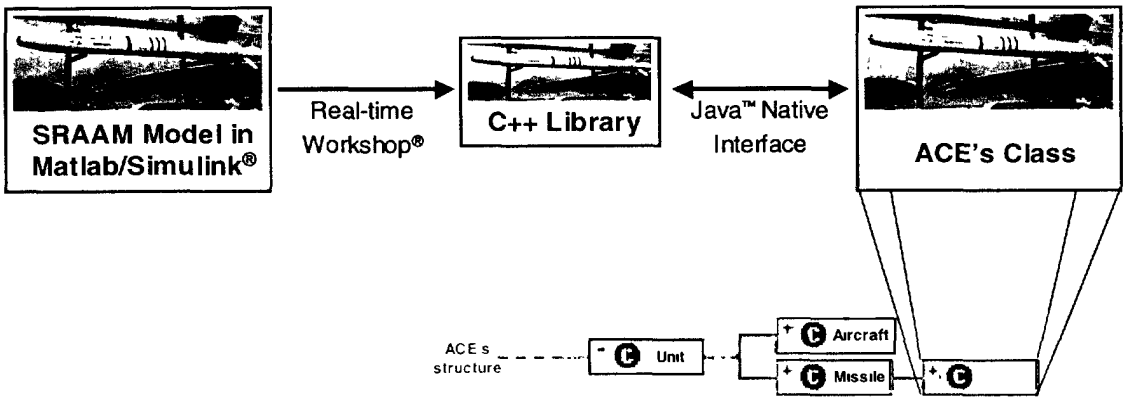

Figure 3: Integration of DRDC-V's SRAAM model in the CF-18 ACE

16. Annex A provides a step-by-step demonstration of how a native model can be linked this way by using the JNI. The instruction lines used to integrate the missile model in the CF-18 ACE are used as examples, and a part of the interface's code is provided in Annexes B and C.

17. Note that because the CF-18 ACE is written in Java, it can be run on almost any computer platform. However, the utilization of the JNI may restrict this capability, depending for which platforms the libraries associated with the external models have been compiled. If a particular external model is available only as a Microsoft Windows® library (a *.dll* file), CF-18 ACE's scenarios using this model will only work in Windows. In the case of DRDC-V's SRAAM model, the libraries were built under Windows, but could eventually be built under Unix or Linux if need be.

Integrating a classified model

18. The source code of the CF-18 ACE is completely unclassified, but the software can treat and produce classified data. DASOR wants to keep the source code unclassified to facilitate its development, but this does not prevent the integration of classified representations of state-of-the-art aircraft components. In the case of the SRAAM model for example, two C++ libraries were actually provided: one classified and one unclassified. When the CF-18 ACE is run on a classified computer, it connects to the classified library. Otherwise, the unclassified library is used.

19. The integration of a classified model written in Java can be achieved in a similar way. A "dummy" unclassified class having the same method declarations as the classified class can be inserted in CF-18 ACE's structure for developmental purposes. When the classified model has to be used, the compiled dummy class (the *.class* file) can be simply replaced by the real class on a classified computer.

IV. CONCLUDING REMARKS

20. The quality of the design of an external model is the first prerequisite to its integration in the CF-18 ACE. If the input/output structure of the model is adequate and well documented, then an interface can be easily built between the CF-18 ACE and any Java or C/C++ model.

21. Using the JNI, a representation of a state-of-the-art SRAAM developed outside DASOR was successfully incorporated in the CF-18 ACE The Precision Weapons Section of DRDC-V currently maintains the missile model, and as new versions of the model will be released to DASOR, they will be included in the CF-18 ACE.

22. The same collaborative approach could eventually be used to integrate other high-fidelity models produced by DND research labs or allied countries. Representations of ECM and ECCM systems[2] are good examples of models that are currently missing in the CF-18 ACE, but that require a fair amount of technical knowledge to produce realistic results. A similar example is the introduction of intelligent agents to pilot the fighters, which would dramatically increase the number of scenarios that can be simulated in stand-alone mode.

23. There are other avenues for linking external models with CF-18 ACE. It is feasible to modify CF-18 into a form that is compatible with the High Level Architecture (HLA) protocol or the Distributed Interactive Simulation (DIS) protocol [5]. This would enhance the interoperability of the CF-18 ACE and would reduce the computational resource requirements in a multiple computer simulation.

[2] Electronic countermeasures and electronic countercountermeasure systems.

REFERENCES

1. P.J. Young, Analyst's Manual for the CF-18 Air Combat Evaluator (Version 1.0), DASOR Contractor Report, Department of National Defence, March 1998.

2. S. Isbrandt and Y. Gauthier, CF-18 Air Combat Evaluator Version 2.0 Overview, ORD Project Report, Department of National Defence, April 2002 (to be published).

3. A. Jeffrey, R. Lestage, M. Lauzon, Short-Range Air-to-Air Missile Modeling (Draft), Technical Memorandum, Defence Research Establishment in Valcartier, March 2002 (Unclassified information from a secret report).

4. S. Liang, The Java™ Native Interface, Addison-Wesley, 1999, ISBN 0201325772.

5. EWA Canada, Software Development Plan for the Implementation of Distributed Interactive Protocols into the CF-18 Air Combat Evaluator (Version 1.0), DND Contractor report, March 1997.

ANNEX A
DOR(MLA) RESEARCH NOTE RN2002/07
JULY 2002

STEP-BY-STEP MODEL INTEGRATION USING THE JNI

A.1. The following paragraphs explain step-by-step[3] how the JNI should be employed to integrate a new model in CF-18 ACE. For some of these steps, the specific example of the SRAAM model is used to illustrate how the integration can be done in practice.

Step 1: Write the Java class for the new model

A.2. The first thing to do is to build the Java class for the new model. This requires a good knowledge of how the new model works, what its input variables are, what its outputs variables are, and what the methods it can perform are. The new class must be inserted properly in the existing structure. In the present case, the *GenNativeMissile* class in inheriting from the *Missile* class, as show in Figure A-1.

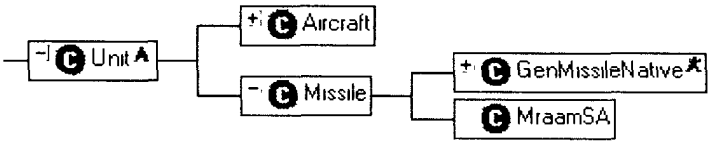

Figure A-1: The *Unit* class and its subclasses

A.3. In the new class, new variables can be created if they are necessary and if they are not already inherited from upper classes. The methods that are required for the functioning of the model must also be declared. The keyword *native* must also be included as part of the methods' definition within the Java class. This keyword signals to the Java compiler that the functions are native language functions.

A.4. The Java class must also load the shared library associated with the native model (see Step 4 for creating this library) and *all the underlying libraries* called by the native model. Here, the shared library associated with the SRAAM model is called *GenMissile*, and this library is based on a library called *Cmissilea_gen*, so the class needs to be initialised by the following lines:

[3] The steps presented here are similar to those given in Sun's tutorial on JNI For a very simple example of native method implementation using JNI, the reader is referred to reference [3] or http //java sun com/docs/books/tutorial/native1 1/index html

```
public class GenMissileNative extends Missile {
      static {
            System.loadLibrary("Cmissilea_gen");
            System.loadLibrary("GenMissile");
      }


}
```

A.5. A part of the *GenNativeMissile* class code is given in Annex B to illustrate how the Java class should be built, including the definition of new variables and new methods.

Step 2: Compile the Java code

A.6. The Java class created in the previous step must be compiled before passing to the next step. This can be done using the *javac* command of Sun's Java Development Kit (JDK), although in most development environment this task is performed automatically at save time.

Step 3: Create the header file

A.7. In this step, the *javah* utility program of JDK must be run to generate a header file (a *.h* file) from the compiled Java class. This header file provides a C function signature for the implementation of the native methods in the external model. When running *javah*, the *–jni* option must be used. The package name must also be included if the Java class is part of a package.

A.8. In the case of the native SRAAM model, this was done with the following instruction line:

```
javah -jni acebase.GenMissileNative
```

which was executed from the root directory of the *acebase* package. The file produced by this instruction line is called *acebase_GenMissileNative.h* and must not be edited.

Step 4: Create a shared library for the new model

A.9. This step is probably the most difficult and may require some C or C++ coding if the source code of the external is not available. This was the case for the SRAAM model. The model was provided as a single C++ library, called *CMissilea_gen.dll*. This library was well documented such that all the public functions within the library were known, as well as the input/output parameters and their respective data types. The header file associated with this library, *CMissilea_gen.h*, was also provided.

A.10. A C++ interface was written to connect the library provided by DRDC-V with the CF-18 ACE. This interface is essentially a "wrapper" for the external model that links the external library with the Java class. Figure 4 shows how it was done for the missile model.

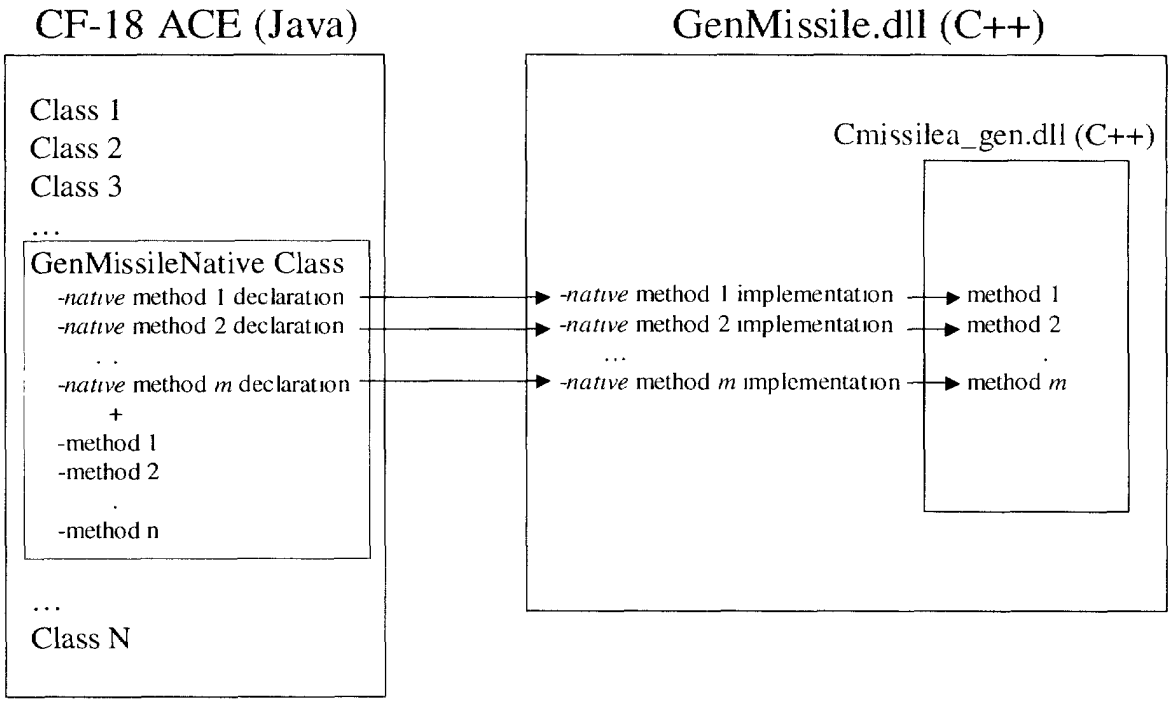

Figure A-2: Linkage of CF-18 ACE with an externally provided library.

A.11. The C++ program called *GenMissile.cpp* (given in Annex B) implements all the native methods declared in the Java class (see Step 1). The declaration of these methods must consistent with the headers of the file produced in produced in Step 3. At the beginning of the C++ program, the *CMissilea_gen.dll* library is loaded such that its internal functions can be called.

A.12. The methods implemented in *GenMissile.cpp* are very simple and are there only to interface with the "real" methods contained in the *CMissilea_gen.dll* library. Why is this interface necessary? Because in some cases, like the one discussed here, the source code of the external model is not available and the only thing that is provided is the library. It is thus impossible to modify the code to make it compliant with the headers produced in Step 3. The only solution is to write a short C++ program that wraps the provided library and build a new library from it.

Step 5: Run the program

A.13. At this point, the external model is integrated into the Java program and should work properly. If a *java.lang.UnsatisfiedLinkError* exception is thrown by the program at runtime, then the shared libraries cannot be accessed, most likely because they were not properly included as resources files for the program.

Interfacing a classified library

A.14. In the case where two C++ libraries are available, one classified and one unclassified, there are two ways to connect both of them to the Java code. The first one is to give the same name to both C++ libraries, and place the one that needs to be use in the appropriate directory of the CF-18 ACE. The interface built by following the steps above will simply connect to the library placed in the directory at runtime.

A.15. However, there are two problems with this approach: the libraries can be easily mixed up and they cannot be used at the same time. A better way to proceed is to build a second interface that inherits the first one, as shown in Figure A-2. The new Java class loads the classified C++ library (see Step 1) instead of the unclassified one. It also re-declares all the native methods of its parent class. These methods are implemented in a C++ wrapper, which is identical to the one use in the unclassified case, but that loads the classified library and thus use the classified version of the model's C++ methods.

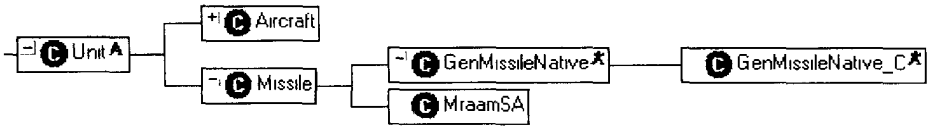

Figure A-2: The *GenMissileNative_C* class inherits *GenMissileNative*, but loads the classified C++ library.

ANNEX B
DOR(MLA) RESEARCH NOTE RN2002/07
JULY 2002

THE C++ WRAPPER

Partial code of the *GenMissile.cpp* code

```
#include "stdafx.h"
#include "acebase_GenMissileNative.h"
#include "CMissilea_gen.h"
#include "jni.h"

#ifdef GENMISSILEPROJECT_EXPORTS
#define GENMISSILEPROJECT_API __declspec(dllexport)
#else
#define GENMISSILEPROJECT_API __declspec(dllimport)
#endif

JNIEXPORT void JNICALL Java_acebase_GenMissileNative_clean
(JNIEnv *env, jobject self, jlong missilePointer){
       CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
       delete missile;
       return;
}

JNIEXPORT jdoubleArray JNICALL Java_acebase_GenMissileNative_getPara
(JNIEnv *env, jobject self, jlong missilePointer){

       CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
       jdouble* parameters =
              missile ->GetPara();
       jdoubleArray paraArray = env -> NewDoubleArray(8);

       env -> SetDoubleArrayRegion(paraArray,0,7,parameters);
       return paraArray;
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getPhi
(JNIEnv *env, jobject self, jlong missilePointer){
       CMissilea_gen *missile = (CMissilea_gen*)missilePointer,
       return missile->GetEuler_M_3();
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getPsi
(JNIEnv *env, jobject self, jlong missilePointer){
       CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
       return missile->GetEuler_M_1();
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getX
(JNIEnv *env, jobject self, jlong missilePointer){
       CMissilea_gen *missile = (CMissilea_gen*)missilePointer,
       return missile->GetXYZ_M_1();
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getY
(JNIEnv *env, jobject self, jlong missilePointer){
       CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
```

```
        return missile->GetXYZ_M_2();
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getZ
(JNIEnv *env, jobject self, jlong missilePointer){
        CMissilea_gen *missile = (CMissilea_gen*)missilePointer,
        return missile->GetXYZ_M_3();
}

JNIEXPORT jdouble JNICALL Java_acebase_GenMissileNative_getTheta
(JNIEnv *env, jobject self, jlong missilePointer){
        CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
        return missile->GetEuler_M_2();
}

JNIEXPORT jboolean JNICALL Java_acebase_GenMissileNative_isStopped
(JNIEnv *env, jobject self, jlong missilePointer){
        CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
        return missile->Stop();
}

JNIEXPORT jlong JNICALL Java_acebase_GenMissileNative_initialize
(JNIEnv *env, jobject self){
    CMissilea_gen *missile = new CMissilea_gen();
    missile->Initialize(),
        return (jlong)missile;
}

JNIEXPORT void JNICALL Java_acebase_GenMissileNative_run
(JNIEnv *env, jobject self, jlong missilePointer){
        CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
        missile->Run();
    return;
}

JNIEXPORT void JNICALL Java_acebase_GenMissileNative_setPara
(JNIEnv *env, jobject self, jlong missilePointer, jdoubleArray paraArray){
        jboolean isCopy1;
        CMissilea_gen *missile = (CMissilea_gen*)missilePointer;
        jdouble* parameters = env -> GetDoubleArrayElements(paraArray,&isCopy1);
        missile->SetPara(parameters);
        if (isCopy1 == JNI_TRUE) {
         env -> ReleaseDoubleArrayElements(paraArray, parameters, JNI_ABORT);
    }

    return;
}
JNIEXPORT void JNICALL Java_acebase_GenMissileNative_setXYZ_1F
(JNIEnv *env, jobject self, jlong missilePointer, jdoubleArray paraArray){
        jboolean isCopy1;

        CMissilea_gen *missile = (CMissilea_gen*)missilePointer;

        jdouble* parameters = env -> GetDoubleArrayElements(paraArray,&isCopy1);

        missile->SetXYZ_F(parameters);

        if (isCopy1 == JNI_TRUE) {
         env -> ReleaseDoubleArrayElements(paraArray, parameters, JNI_ABORT);
    }

    return;
}
```

ANNEX C
DOR(MLA) RESEARCH NOTE RN2002/07
JULY 2002

THE JAVA CLASS OF THE INTERFACE

Partial code of the *GenMissileNative* class

```
package acebase;

import java.text.*;
import java.util.*;

public class GenMissileNative extends Missile {

   static {
       System.loadLibrary("CMissilea_gen");
         System.loadLibrary("genmissile");
   }

   private long missilePointer = -99999;
   private double oldMach;
   private boolean launchFlag;
   private double[] offset;

   public GenMissileNative() {
       super();
   }

   public native void clean(long missilePointer);
   public native double getDeltaPitch(long missilePointer);
   public native double getDeltaYaw(long missilePointer);
   public native double getMach(long missilePointer);
   public native double[] getPara(long missilePointer);
   public native double getPhi(long missilePointer);
   public native double getPsi(long missilePointer);
   public native double getSeekerEl(long missilePointer);
   public native double getStep(long missilePointer);
   public native double getTheta(long missilePointer);
   public native double getTime(long missilePointer);
   public native double getX(long missilePointer);
   public native double getY(long missilePointer);
   public native double getZ(long missilePointer);

   public long getMissilePointer() {
       return missilePointer;
   }

       .
       .
       .

}
```

ANNEX D
DOR(MLA) RESEARCH NOTE RN2002/07
JULY 2002

CF-18 ACE SCENARIO DATA FILE EXAMPLE

D.1. In the following CF-18 ACE scenario file example, ***bolded italicised*** data lines instruct the model to use classes specific for the missile model documented here, and **bolded** data lines contain specific data required by the *GenMissileNative* class. In order to use the classified version of the model, the *GenMissileNative_C* class must be loaded (see the instructions for the second aircraft below).

```
Manager                      Sim manager type
45.0 0.05                    Simulation end time (s), time step (s)
315169                       Random number seed - 6 digit integer
gen_missilea_test1.out       Output file name
both                        Color of units which will be displayed (blue,red,or both)
2 CF-18 Mig-29               Number of selectable units and their names
2 false                      Number of units, remote attached
Aircraft                    Class type for unit 1        ****************************
Blue CF-18                      Unit 1: color (Blue or Red), PIN
CAP                             Initial status
Platform                        Class type for platform
abs -6000 -995.0 4572.0 0.0        info type, initial location (m) and heading (deg)
209.47                             initial velocity (m/s)
ConSpdHgt                       Class type for platform manoeuvre
false                           No RWR
false                           No data link
1                               Number of sensors
Sensor                          Class type for sensor 1
20.0 40000                         fov (degrees), max. range (m)
1                               Number of weapon types
WeaponSystem                    Class type for weapon system 1
2 GenMissileNative              Number of missiles, Class type of missiles
-1.5 -2.5 1.1                     offset X,Y,Z of missile 1 relative to aircraft (m)
 1.5 -2.5 1.1                     offset X,Y,Z of missile 2 relative to aircraft (m)
8.0 1.0                            lethal radius (m), probability of kill
650.0 15000.0                      minimum and maximum target ranges at launch (m)
30.0                               maximum time of flight (s)
Aircrew                         Class type for aircrew
2.0                                time to detect/fire after target enters FOV
2                                  num evasion manoeuvres
2.0                                time delay for evasion manoeuvre
ConSpdRadHgt                    Class type for evasion manoeuvre
2 1 4.799                          num manoeuvre parameters, turn dirn, load
2.0                                time delay for evasion manoeuvre
ConSpdHgt                       Class type for evasion manoeuvre
0                                 num manoeuvre parameters
Aircraft                    Class type for unit 2        ****************************
Red Mig-29                      Unit 2: color (Blue or Red), PIN
CAP                             Initial status
Platform                         Class type for platform
```

```
abs 6000 -1005.0 4572.0 180.0     info type, initial location (m) and heading (deg)
221.39                             initial velocity (m/s)
ConSpdHgt                        Class type for platform manoeuvre
false                             No RWR
false                             No data link
1                                 Number of sensors
Sensor                            Class type for sensor 1
45.0 40000                           fov (degrees), max  range (m)
1                                Number of weapon types
WeaponSystem                     Class type for weapon system 1
2 GenMissileNative_C               Number of missiles, Class type of missiles
-1.5 -2.5 1.1                      offset X,Y,Z of missile 1 relative to aircraft (m)
 1.5 -2.5 1.1                      offset X,Y,Z of missile 2 relative to aircraft (m)
8.0 1.0                            lethal radius (m), probability of kill
650.0 15000.0                      minimum and maximum target ranges at launch (m)
30.0                               maximum time of flight (s)
Aircrew                           Class type for aircrew
1.0                                  time to detect/fire after target enters FOV
2                                    num evasion manoeuvres
1.5                                  time delay for evasion manoeuvre
ConSpdRadHgt                      Class type for evasion manoeuvre
2 -1 1.5                             num manoeuvre parameters, turn dirn, load
8.0                                  time delay for evasion manoeuvre
ConSpdHgt                         Class type for evasion manoeuvre
0                                    num manoeuvre parameters
```

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM
(highest classification of Title Abstract, Keywords)

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 Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8)

Operational Research Division
Department of National Defence
Ottawa, Ontario K1A 0K2

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 or U) in parentheses after the title)

INTEGRATION OF ADVANCED MODELS IN THE CF-18 AIR COMBAT EVALUATOR – EXAMPLE OF A MISSILE REPRESENTATION (U)

4 AUTHORS (last name, first name, middle initial)

GAUTHIER, Y

5 DATE OF PUBLICATION (month Year of Publication of document)	6a NO OF PAGES (total containing information Include Annexes Appendices, etc)	6b NO OF REFS (total cited in document)
JULY 2002	21	5

7 DESCRIPTIVE NOTES (the category of 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)

RESEARCH NOTE

8 SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development Include the address)

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)
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 NOS (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11 DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)
(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors, further distribution only as approved
( ) Distribution limited to government departments and agencies, further distribution only as approved
( ) Distribution limited to 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), or (U) It is not necessary to include here abstracts in both official languages unless the text is bilingual)

The CF-18 Air Combat Evaluator (CF-18 ACE) is a computer-based simulation developed in Java that enables investigation of few-on-few fighter engagements. It includes representations of the aircraft platform, aircrew, data link, sensors, and weapons. This research note explains how new models should be integrated in the CF-18 ACE It also explains how this integration can be achieved when the models are coded in a language other than Java, such as C or C++. The example of a high-fidelity missile model recently incorporated in the CF-18 ACE is used to illustrate how the integration can be done in practice.

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)

Air Combat Evaluator
CF-18 ACE
Missile
Java
JNI
SRAAM
Engagement simulation

Canada

#517737

CA021000