Common Ada Bindings To Compartmented Mode Workstations

Cedric Armstrong, Ph.D.
armstrong@mls.saic.com

Ramesh Venkatraman
ramesh@mls.saic.com

Science Applications International Corporation
10770 Wateridge Circle
San Diego, CA 92121

Abstract

This paper describes a project to build a common set of Ada bindings to the Compartmented Mode Workstation (CMW) operating system. The paper discusses various design alternatives for creating these bindings. An object-oriented CMW model is described together with an object-oriented common Ada-CMW interface specification. A two-layered design solution is presented and its merits and demerits are discussed. Issues relating to the development of these bindings including testing, Ada 83 vs. Ada 9X issues, and portability are described and some of the ways to handle these issues are discussed .

1.0 Introduction

The emergence of Unix as the de facto operating system of the future raises new concerns about the security of the operating environment and data. In order to address this specific need, the Defense Intelligence Agency (DIA) has recently defined a secure operating system standard called the Compartmented Mode Workstation (CMW) [1]. CMW provides an open system, Multi-Level Secure (MLS) computing solution for customers with secure computing requirements. A CMW system is assumed to be operating at a security level of B1 enforcing both Mandatory Access Control (MAC) and Discretionary Access Control (DAC). CMW also provides secure networking support.

In order to embrace this newly emerging CMW technology, the Ada Joint Program Office (AJPO) recognized the need to provide Ada bindings to CMW Application Programmer's Interface (API). Using the Ada Technology Insertion Program (ATIP) guidelines, AJPO funded this project to create Common Ada Bindings to CMW APIs.

2.0 Defining a Common CMW API

Like any new technology, the CMW operating system evolved out of very high-level requirements provided under the CMW Evaluation Criteria [2]. Each CMW vendor either took an existing CMW operating system and ported it to their platform with very little or no modifications to the original APIs or created a set of unique API for their platform.

After a preliminary assessment of CMW APIs from various CMW vendors, it was found that three major vendors (viz. Digital, HP, and IBM) basically use SecureWare's CMW technology as their operating system base and therefore share a more-or-less common set of CMW API. Sun's CMW implementation, however, while it incorporated some SecureWare technology, differs significantly at the API level.

This presents problems for a common set of CMW APIs. Since there are no standard "abstract" CMW specifications, our task of defining a common set of CMW APIs involved selecting the best among the two design alternatives discussed below.

2.1 Design Approaches

<u>Design Alternative A - The Bottom-up Strategy:</u>

This approach involves studying the existing set of CMW APIs as implemented by the various CMW vendors, grouping them into objects and their operations, and developing a common set of CMW APIs.

The main advantage of this approach is that it lends itself to object-oriented design. Another important advantage is that the resulting common CMW API set will be fully supported by the existing vendor-specific CMW APIs.

One of the major disadvantages of this approach is the incompatibility between the SecureWare-based CMW APIs and the Sun CMW API.

<u>Design Alternative B - The Top-down Strategy:</u>

This approach involves creating a common set of "abstract" CMW specifications using the original CMW Evaluation Criteria document and implementing a set of Ada CMW APIs to this.

The main advantage of this approach is that it relates more closely to the initial project objectives. Also, by creating a generic set of Ada-to-CMW specifications, we can defer platform-specific implementations; thereby enhancing portability across the various CMW platforms.

One of the major disadvantages of this approach is that the task of creating a common set of "abstract" specifications from the CMW Evaluation Criteria document. Since this is not a requirements document, the task of extracting a common set of CMW requirements will be more involved than planned to perform under the project's scope. Moreover, there is no guarantee that the resulting CMW API will be fully supported by the existing CMW implementations.

Finally, the actual vendors' CMW system architecture seem to have (to some degree) driven the design of the APIs. Hence it might not be possible to change the API and retro-fit an actual CMW system architecture.

2.2 The Chosen Design Alternative

Based on the discussions above, it was decided to follow Alternative A - The Bottom-up Strategy as our basis for defining a common set of Ada bindings to CMW. However, while mapping SecureWare- specific CMW APIs over to Sun CMW APIs, it was found that either we have to ignore APIs that are unique to each domain or make those calls as a no-op in an unsupported CMW system.

If we ignore unique APIs, the resulting API set is not only inadequate in its functionality but also necessitates the use of platform-specific APIs. Any use of platform-specific APIs would defeat the objective of common bindings. If we make unique APIs as no-ops, it adds more complexity and possible confusion.

Thus it was decided to develop the common CMW API set using the SecureWare-based CMW APIs as the basis.

3.0 An Object-Oriented CMW Model

Having identified a common set of CMW APIs for the SecureWare-based CMW implementations, the next step was to develop an object-oriented model of CMW. For this analysis, we used the Coad/Yordon object-oriented analysis methodology [5].

Using the inherent groupings of operations on CMW objects as a guide, we came up with three major abstract CMW Classes and several other groupings of operations on CMW system data:

- Secured Data Class
- Security Attribute Class
- Security Database Class

3.1 Secured Data Class

This abstract class is further decomposed into two abstract classes:

- User Data Class
- Kernel Data Class

<u>User Data Class</u>
The following objects are grouped under the User Data Class:

- File
- Directory
- Multi-level Directory
- Device

<u>Kernel Data Class</u>
Under the Kernel Data Class, the following objects are grouped:

- Semaphore
- Message
- Shared Memory

An object-oriented model of the Secured Data class is given in Figure 3.1.

3.2 Security Attribute Class

This abstract class is further decomposed into two abstract classes:

- Mandatory Access Control (MAC) Class
- Discretionary Access Control (DAC) Class

<u>MAC Class</u>
The Following objects are grouped under the MAC Class

- Privilege
- Label (Abstract Class)
    - Clearance
    - Information Label (ILABEL)
    - Security Label (SLABEL)

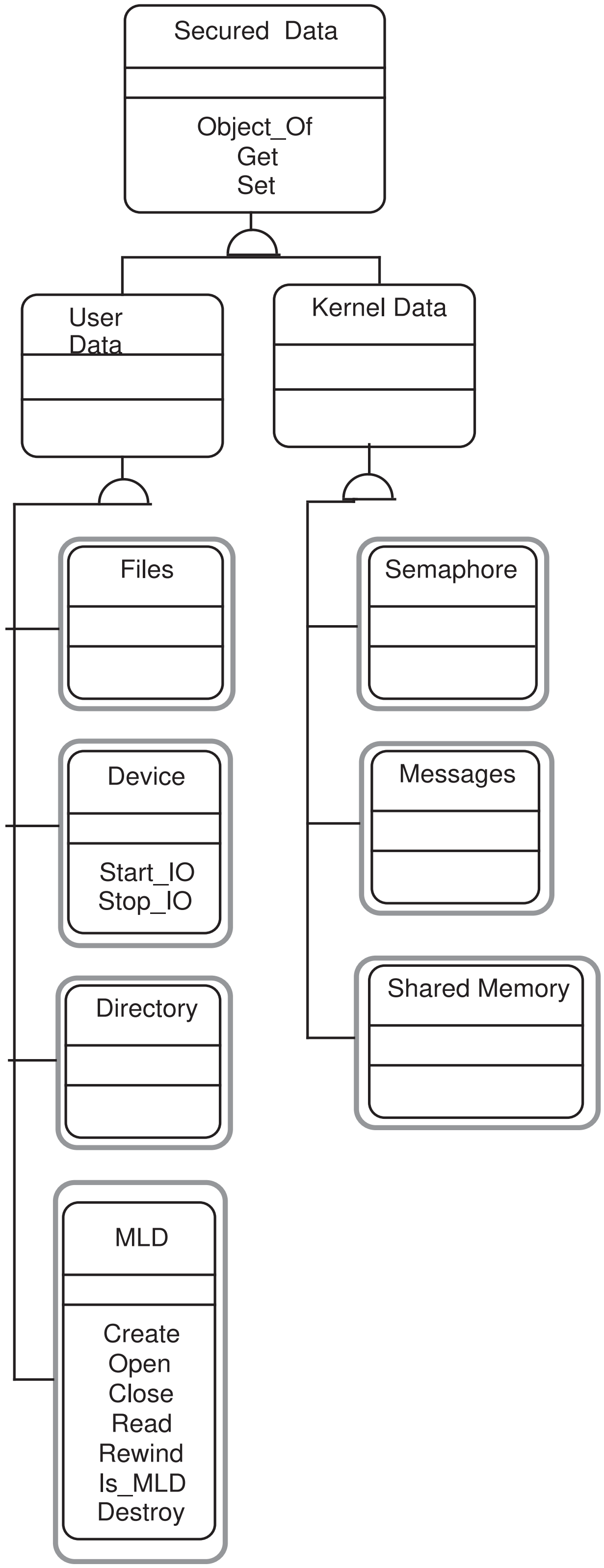

Figure 3.1 - Secured Data Class

DAC Class

The DAC Class contains only one object:

- o Access Control List (ACL)

An object-oriented model of the Security Attribute class is given in Figure 3.2.

3.3 Security Database Class

This abstract database class contains the following objects:

- o Defaults DB
- o Passwords DB
- o Devices DB
- o File Control DB
- o Privilege Control DB
- o Terminal Control DB
- o Command Authorization DB
- o TNET Interface DB
- o TNET Remote Hosts DB
- o TNET Config DB

An object-oriented model of the Security Databases class is given in Figure 3.3.

3.4 Other CMW System Objects

The Following is a set of objects for which there are operations provided by the CMW operating system:

- o Current Process
- o User
- o Audit
- o Trusted Control Block (TCB)
- o Disk

It should be noted here that the common CMW API set does not provide any support for operations on objects that are not CMW specific (e.g. chmod, rmdir, etc.,).

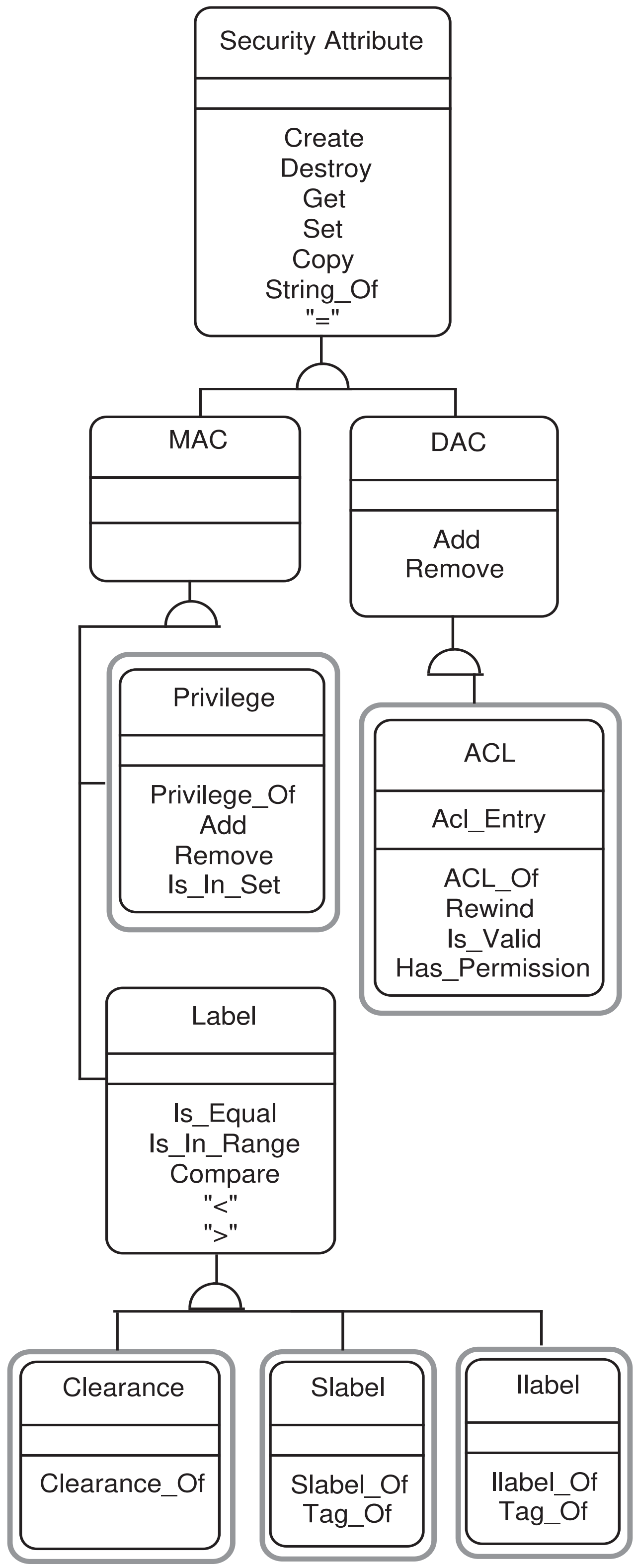

Figure 3.2 - Security Attribute Class

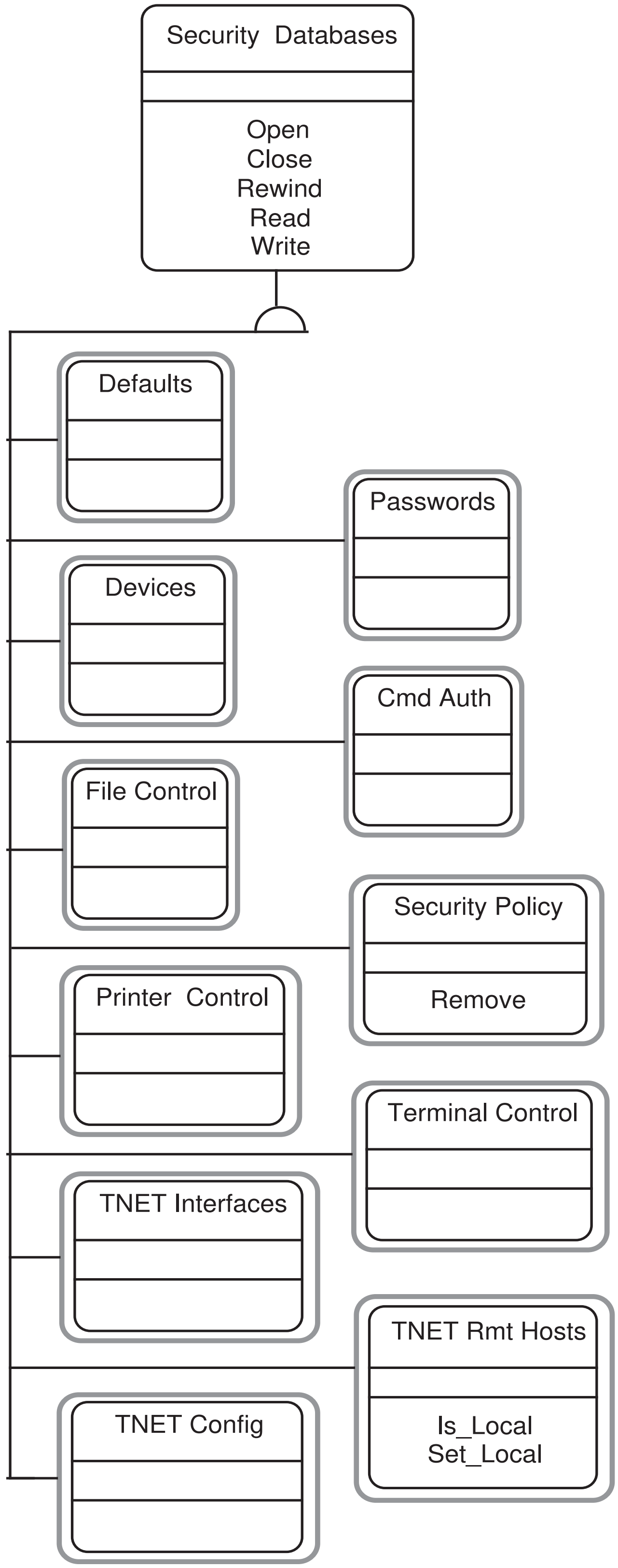

Figure 3.3 - Security Database Class

4.0 The Two-Layered Design Approach

Implementation of the identified set of common CMW APIs in Ada involves encapsulating the CMW objects and its operations as Abstract Data Objects (ADOs) and Abstract Data Types (ADTs). However, translating data types and functions from C to Ada is a non-trivial task. Especially one that involves over 200 functions and several hundred structures.

Since the objective of a common Ada binding to CMW is to provide a portable interface that promotes the use of good Ada constructs and naming conventions, two design alternatives were examined.

<u>Design Alternative A - The Single-Layer Solution:</u>

According to this design approach, we will create a CMW Types package containing data structures and other constants that reflect our object-oriented refinement. We will then create a CMW package and sub-packages containing operations for the identified CMW objects. The body of these packages will perform all the necessary data conversions and "pragma" calls to the corresponding platform-specific CMW APIs.

The main advantage of this approach is that the users of the CMW package need not have to worry about platform-specific implementations and the C naming conventions. Any code "withing" the CMW package is portable across SecureWare-based CMW platforms at the specification level.

One major disadvantage we could see is the need to develop a complex set of CMW vendor-specific bodies for each common CMW API. When a higher-level API needs to call more than one lower-level vendor-specific C API or more than one higher-level API call the same lower-level C API , we need to recode the entire interface or create many local functions within the body for its internal use. Also, this approach will not provide bindings to all the vendor-specific CMW APIs. Since there is no mapping between the existing CMW vendor-specific APIs and our common CMW API, porting an existing C based CMW application to Ada would be a non-trivial task.

<u>Design Alternative B - The Two-Layer Solution:</u>

In this design approach, we will create a layer very much like the one mentioned in Design Alternative A and call this an "Ada-CMW Interface Layer". Underneath this layer, we will create a platform-specific "Concrete Layer". This concrete layer will map one-to-one with the respective C functions including the C naming convention for variables and functions.

This approach not only has almost all the advantages of the "Single-Layer" approach but also solves several of its deficiencies. Ada users can now have access to all the CMW APIs as provided by the particular vendor. This is particularly important to security engineers who are used to vendor-specific CMW APIs. Since there is a one-to-one mapping between each vendor-specific API, users can easily port existing CMW applications written in C to Ada. However, for new CMW applications written entirely in Ada, it is recommended that the Ada programmers use the Ada-

CMW Interface layer.

This approach will also reduce the complexity of the "Ada-CMW Interface Layer" body and makes the task of creating the bindings a little easier.

A major disadvantage of this approach is the development of an extra layer. Ada users who "with" this concrete layer directly will be making their application code non-portable.

However, merits of the "Two-Layer" approach more than outweigh the "Single-Layer" approach.

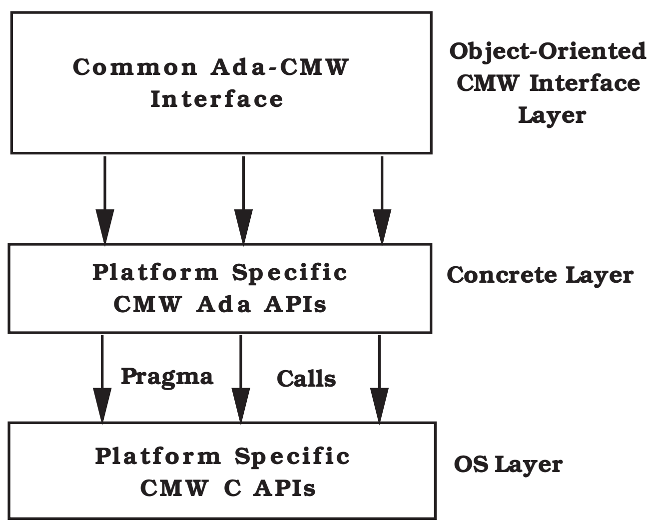

Figure 4.1 - The Two-Layered Ada-CMW Interface

4.1 Ada-CMW Interface Layer

The Ada-CMW Interface layer hides all C language-specific CMW API conventions and provides an Ada application programmer with an object-oriented CMW package. Since one of the objectives of this project is to provide these CMW bindings for Ada 9X, this interface layer is implemented in both Ada 83 and in Ada 9X.

In both implementations, the CMW objects are encapsulated into subpackages within the main CMW package. Operations on the CMW objects are implemented as over-loaded procedures and functions. Since these higher-level Ada-CMW APIs are to be compatible between Ada 83 and Ada 9X and due to the availability of an Ada 9X compiler on the selected CMW platform, this interface layer was first designed around Ada 83 and later ported to Ada 9X. This approach not only solves the portability problems but also makes the bindings look similar between the two versions of Ada.

4.1.1 Ada-CMW Interface Layer Specification

The Ada-CMW interface layer specification is broken into 29 subpackages. Most of these packages contain an "Object_of" function to return a handle for the specific CMW object and use "Set" and "Get" procedures to set and get attributes of that object. Due to the nature of the CMW operating system, some of the objects have no "object_of" functions. However, the "Set"/"Get" operators operate upon a pre-defined system object (e.g. Security Database objects, Audit, Current_Process, etc.,).

Due to language differences between Ada 83 and Ada 9X, there will be some differences at the specification level between the two implementations.

4.1.2 Ada-CMW Interface Layer Body

The body of the Ada-CMW Interface layer is broken into separate compilation units. All platform-specific interface coding is hidden in the body, and it uses the platform-specific concrete module to perform its task. Hence, for each CMW implementation, there is a specific interface body.

In order to maximize ease of portability, almost all data conversions are performed at the platform-specific concrete layer.

4.2 Platform-Specific Concrete Layer

The Platform-Specific Concrete Layer is responsible for interfacing with the C language-based CMW APIs. Due to reasons identified earlier, this layer is implemented as a "thin" binding which preserves the naming conventions of the corresponding C APIs. This layer primarily consists of data conversions and "pragma INTERFACE" calls to the C functions. The Ada 83 implementation of this layer differs slightly with the Ada 9X implementation due to C language calling conventions.

It should be noted here that string conversion between C based character strings and Ada String type in Ada 83 follows the model of Ada 9X's predefined package "Interfaces" [3]. This "Interfaces" package and its subpackages "C" and "C.Strings" were implemented in Ada 83 for the use of the concrete layer. This method helped to port this layer over to Ada 9X with very little modification.

In the current implementation of Ada-CMW bindings, there are two versions of the concrete module; one for Digital CMW and one for HP CMW.

4.2.1 Digital CMW

The Digital CMW concrete bindings are based on the "External Field Test" version of the Digital OSF/1 MLS+ operating system running on their Alpha 3000 model 500 AXP computer. The Ada 83 version of the code is compiled using the Digital Ada compiler. The Ada 9X version of the code is compiled using the GNAT Ada 9X compiler.

4.2.2 HP CMW

The HP CMW concrete bindings are based on the HP-UX B-Level Security (BLS) operating system running on their HP 9000 series 755 computer. The Ada 83 version of the code is compiled using the Alsys Ada compiler. The Ada 9X version of the code will be compiled using the GNAT Ada 9X compiler.

5.0 Ada 83 vs. Ada 9X Issues

One of the requirements of this project was to address Ada 83 vs. Ada 9X issues relating to the Ada bindings to CMW. After our initial review of the CMW APIs and the Ada 9X language features, it was found that bindings implemented using Ada 9X will lend themselves more nicely to our object-oriented refinement of CMW APIs. However, several new issues arose relating to the implementation of the bindings in Ada 9X. They were as follows:

- o Compiler Availability
- o Compiler Dependencies
- o Posix Support

5.1 Compiler Availability

Since a validated 9X compiler will not be available before the end of this project, we tried to use the GNU-NYU Ada 9X Translator (GNAT) Ada 9X compiler available from the Free Software Foundation. Unfortunately, as of this time, there are no GNAT Ada 9X compiler binaries for neither the Digital Alpha MLS+ nor the HP-UX BLS.

The problem of porting the Ada compiler is twofold. First there is the issue of creating the binaries compatible for the new architecture. Second, due to the changes made to the underlying Unix operating system and its libraries for CMW, there are some incompatibilities between the vanilla Unix system libraries and the CMW version. The respective CMW vendors are currently in the process of solving this issue.

5.2 Compiler Dependencies

One of the goals of this project was to provide Ada-CMW bindings for Ada 9X; hence, care was taken to make the interface look as similar as possible at the specification level. However, some of the Ada 9X-specific constructs, such as those for interfacing to the C language APIs had made the bodies of the two versions incompatible.

<u>Use of Class Wide Programming Support</u>

Since Ada 9X supports "class wide" programming, abstract classes were created to exploit the "dynamic dispatching" capabilities of Ada 9X [3]. Under the Ada 83 implementation of the bindings, these operations are overloaded to act upon a specific object type. In Table 5.1, you can find several overloaded Ada 83 functions and their equivalent Ada 9X functions.

Ada 83	Ada 9X
Slabel_Of (Directory) Slabel_Of (File) Slabel_Of (Device) Slabel_Of (MLD) Slabel_Of (Semaphore) Slabel_Of (Shared_Mem) Slabel_Of (Messages)	Slabel_Of (Secure_Data'Class)
Ilabel_Of (Directory) Ilabel_Of (File) Ilabel_Of (Device) Ilabel_Of (MLD) Ilabel_Of (Semaphore) Ilabel_Of (Shared_Mem) Ilabel_Of (Messages)	Ilabel_Of (Secure_Data'Class)
ACL_Of (Directory) ACL_Of (File) ACL_Of (Device) ACL_Of (MLD) ACL_Of (Semaphore) ACL_Of (Shared_Mem) ACL_Of (Messages)	ACL_Of (Secure_Data'Class)

Table 5.1 - List of Over-loaded Security Attribute Functions

Use of Argv/Argc

Some of the CMW APIs require the use of argv/argc command-line variables. On most Ada 83 compilers for Unix platforms, these can be found under one of the vendor supplied utility packages. Unfortunately, we were not able to use the "argv" support provided by Posix because it is incompatible with the type of variable expected by the CMW API [4]. Also, any use of Posix makes the Ada 83 to Ada 9X port very difficult (see discussions under the section on Posix Support).

Therefore, due to a lack of standardization in this issue, we have provided "argv/argc" support using compiler-supplied packages. However, use of this is very much restricted.

5.3 Posix Support

As mentioned earlier in this paper, this common set of Ada bindings do not provide any support for non-CMW-specific system APIs. Users of these bindings may have to use the Posix bindings to gain access to such calls. Since there is no Posix support in Ada 9X at this time, we have minimized our dependency on Posix in our Ada 83 version of the Ada-CMW bindings in order to make porting to Ada 9X easier.

6.0 Other Issues

While creating the bindings, we encountered a few other minor issues. Though several of these issues have been solved and many of them have became non-issues (e.g. incompatible SecureWare and Sun APIs), we would like to mention the three major ones. They are as follows:

- Graphic User Interface (GUI)
- Testing
- Portability

6.1 Graphic User Interface (GUI)

CMW systems are, by nature, heavily window-based. Any Ada application written for a CMW operating system (Digital, HP, or IBM) must use Ada-Motif bindings. However, all the security window decorations are provided by the Secure Motif Window Manager of the CMW system.

As such, there are no common CMW-GUI APIs in the SecureWare CMW domain. Moreover, data types for these Motif- specific APIs are based on the type of Ada-Motif bindings being used. Hence, it was decided not to create any specific dependencies to GUI at the Ada-CMW Interface layer.

6.2 Testing

The most difficult part of the development of these bindings was testing. This difficulty arises due to the following:

- Large Number of APIs
- Difficulty Creating an Appropriate Test Environment

6.2.1 Large Number of APIs

At the platform-specific concrete level there are over 210 CMW APIs. Many of these perform more than one simple function. At the Ada-CMW Interface level, there are over 270 common CMW APIs. Since many these APIs are interdependent, it was not possible to develop a unit-level rigorous test routines for each one of them. Moreover, development of such testing strategy will make the testing task bigger than that of the bindings themselves.

Thus, the decision was made to concentrate our testing efforts at the common Ada-CMW interface level. The testing strategy consists of creating a test suite for each CMW object. These tests will provide test coverage not only for the respective CMW object but also across several other objects as well. At the concrete API level, tests programs were written to test as many individual APIs as possible. Since our objective is to test the validity of the bindings to the CMW API rather than testing the actual APIs, test suites were written to exercise paths within our bindings code and the compatibility of data structures.

6.2.2 Difficulty Creating an Appropriate Test Environment

Because the CMW environment is highly compartmentalized, testing is inherently difficult. Also, due to CMW's security mechanism, creating a specific test case may require extensive preparation. This involves creating an appropriate privilege set for the test process and to the CMW objects being tested. To complicate this task even further, certain test cases have to be performed by specific user roles (ISSO, System Administrator, Operator, and other users) and within a specific clearance level (System High, System Low, etc.,).

6.3 Portability

The portability issue is twofold. One part involves porting the code developed on one CMW platform to another. The other involves porting an Ada 83-based CMW application over to Ada 9X.

<u>Porting Across CMW Platforms</u>

An Ada application programmer developing an application on the CMW platform will be able to use the common Ada-CMW APIs as provided for the Digital Alpha MLS+ and port the code over to an HP-UX BLS system with no changes to their CMW code. However, they may need to compile with a new set of bodies for the CMW package and the platform-specific concrete layer.

<u>Porting From Ada 83 to Ada 9X</u>

Since at the specification level there are few minor differences, Ada application code using the Ada 83 version will need to be modified slightly. However, these differences are solely due to functions exploiting the "dynamic dispatching" of Ada 9X. Apart from these minor changes, an Ada 83-based CMW application program can be compiled along with the Ada 9X version of the CMW bindings.

It should be noted here that any dependency to Posix or GUI support packages may make the port from Ada 83 to Ada 9X very difficult.

7.0 Conclusions

Development of common Ada bindings to CMW raised several new issues:

- First, the non-availability of an abstract set of CMW APIs.

- Secondly, the incompatibilities between the SecureWare-based CMWs and the Sun CMW.

- Thirdly, the availability of an Ada 9X compiler on the CMW platforms and Posix bindings for Ada 9X.

The solution for the non-availability of an abstract set of CMW APIs involved taking the existing set of CMW APIs and working our way bottom-up to arrive at a set of object-oriented CMW interface APIs. In order to solve the incompatibilities between the SecureWare-based CMWs and the Sun CMW, a decision was made to use the SecureWare technology as the basis for the development of common bindings.

However, the availability of an Ada 9X compiler on the CMW platforms is being addressed at the present time. Non-availability of Posix bindings for Ada 9X was addressed by minimizing and isolating the use of Posix calls in our CMW bindings.

Portability of our common Ada bindings will be demonstrated by using Digital MLS+ and HP BLS as the two CMW platforms. Ada 83 code written using the common set of Ada-CMW bindings for the Digital MLS+ system will be compiled and tested under the HP BLS system without any modification.

Portability across Ada 83 and Ada 9X will be demonstrated by compiling the concrete layer code written in Ada 83 on the GNAT Ada 9X compiler after modifying the "pragma " interface calls.

Because the objective of this project is to create common Ada bindings to CMW APIs, careful consideration was given to minimize the influence of non-CMW Unix APIs on our bindings.

Since this project was sponsored by AJPO, these bindings will be available from their FTP site. Users of these bindings will be able to compile with a subset of packages given below based upon their platform need and the choice of Ada 83 vs. Ada 9X compiler.

Common Ada-CMW Interface Packages

- o package CMW
- o package CMW_Types

Digital MLS+ CMW Concrete Layer

- o package DEC_CMW
- o package DEC_CMW_Types

HP BLS CMW Concrete Layer

- o package HP_CMW
- o package HP_CMW_Types

It should be noted here that the naming convention between the Ada 83 version of the bindings and Ada 9X version are the same. Also, the Ada 83 version of the concrete modules use the services of "package Interfaces" provided along with the bindings.

8.0 Acknowledgment

The authors would like to acknowledge the support received from Digital Equipment Corporation who lent their equipment, software, and services to this project.

References

[1] Security Requirements for System High and Compartmented Mode Workstations,
DIA Document DDS-2600-5502-87
November 1987.

[2] Compartmented Mode Workstation Evaluation Criteria Version 1 (Final).
DIA Document DDS-2600-6243-9
November 1991.

[3] Ada 9X Reference Manual
Draft Version 4.0
Proposed ANSI/ISO Standard
September 1993, pp. M-3

[4] Posix Ada Language Interfaces
Part 1: Bindings for System API
IEEE Std. 1003.5-1992
June 1992, pp. 88-90

[5] Peter Coad, Ed Yourdon,
Object-Oriented Analysis
Yourdon Press,
Englewood Cliffs, NJ, 1991