# USE OF THE ADA LANGUAGE SYSTEM IN CONFIGURATION CONTROL OF FORTRAN BASED SOFTWARE

by

Dan Dyke
Naval Training System Center
University of Central Florida
Orlando, FL

## Abstract

The Configuration Control and Management of Trainer Systems Software has been recognized as a significant problem for many years. Many software tools are being developed for use as aids in this task. The efforts of the Department of Defense in the development of Ada® and Ada Programming Support Environment (APSE) have strongly addressed the problems of Configuration Control and Management. However, these efforts have been targeted for Ada based software as it is anticipated that this will be the primary language of the DOD. The author feels that FORTRAN based software will require maintenance for many years even after the acceptance of Ada, and that the features of the APSE could be applicable to this task.

The Ada Language System (ALS) which will soon be available, is the Army's implementation of the Ada Programming Support Environment. ALS has also been accepted by the Navy. This paper discusses the applicability of the ALS to the problem of FORTRAN based software Configuration Control and Management. A basic demonstration tool will be developed and analyzed.

## Introduction

Computer software represents the majority of the cost of training equipment. In the Naval Training System Center's COG '20' inventory alone, there are hundreds of computer programs and many millions lines of code representing close to a billion dollars of investment. While the need for effective configuration management of this investment has been recognized for many years, only recently have automated software tools been applied to the task. Software tools such as NTSC's Training Equipment Software Configuration Control System (TESCCS), the UNIX Source Code Control System (SCCS), Softool's Change and Configuration Control (CCC), and Digital Equipment Corporation's Code Management System (CMS) have been utilized on a single device basis; but a standard automated tool applied to a large number of projects is far from a reality. Other tools are or will shortly be available as well.

These numerous tools represent a wide variety of implementations on a variety of computer models and operating systems. Portability of Trainer System Software from a contractor utilized configuration control tool system to a different tool system in use by the post-deployment software support activity is a lengthy manual process requiring a "cold start" verification of the baseline. It is unlikely that any of the potentially useful information contained in one tool's project data base could be utilized by another tool at all.

While the proliferation of programming languages was one of the key catalysts for the DOD initiative to adapt Ada as a standard language, the proliferation of operating systems and tools as mentioned above prompted STONEMAN [1] and a DOD initiative for an integrated Ada Program Support Environment (APSE). Configuration control and management was a key concern in STONEMAN and in the development of APSE's which are now being made available. While a single standard is not yet available, much effort is being expended to ensure portability of tools, files and data bases between different APSEs. The work of the Kernel Ada Programming Support Environment (KAPSE) Interface Team [2] is an example of this effort. The Ada Language System (ALS), developed by Softech for the Army (and will be used by the Navy), has many features designed for configuration control and management.

Why an APSE for Configuration Control and Management of Trainer System Software written primarily FORTRAN and Assembler languages?

- These Ada environments were designed with configuration control and management as a prime concern. The ALS is one of the first programming environments to include native rather than "tacked on" support for configuration management of program families [3].

- The ALS is likely to be widely available. It is owned by the DOD and will probably be Government Furnished Equipment to contractors.

- The ALS is open-ended and expandable. Many new tools such as the Army's Software Management and Control System (SMCS) can be expected to be

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

available in the near future.

-FORTRAN-based Trainer System Software is still being specified on new trainer systems. Thus life-cycle management of FORTRAN programs will be required for at least 15 years.

This paper describes an attempt to apply the ALS to the task of providing an environment for post-deployment software support of FORTRAN-based trainer system software. Some obstacles which could hinder such an attempt were obvious in the early stages of this project and are noted here:

-The ALS is in its infant stage (released February 1985) and thus the realization of its full potential will require years of debug, optimization, enhancement and experimentation.

-The ALS was designed for Ada and assembler programming language requirements as the use of other languages was not encouraged by the DOD initiative.

## ALS Features for Configuration Control and Management

The ALS file system is sophisticated enough that it can be viewed as a data base management system. It is a collection of objects called nodes. The three varieties of nodes are files, directories and variation headers. The file and directory nodes are quite similar to those found in hierarchical operating systems such as UNIX and VMS. As each file in the ALS is actually a member of a revision set (similar to VMS), it is possible to track changes made to a file over time. The variation header object is used to represent families of programs.

Members of variation sets are functionally similar software components that differ in their implementation details. Members of variation sets are different from revision sets in that they do not supersede each other. Thus they are named, not numbered. A variation set header can occur anywhere a directory node can occur except at the root of the database. The members can be revision sets (files), directories, other variation sets, or any combination of these. There is also a provision for node sharing between revision sets, allowing for reduced storage where different variation sets contain many identical components. This concept of variation sets could prove very useful for cases where multiple configurations of a trainer program are required by staggered modifications, or if a unique requirement exists at a single site.

Each node in the ALS data base is automatically assigned a unique identifier consisting of the following three fields:

-Object serial number (10 bytes).
-ALS data base identifier (7 bytes).
-Organization identifier (10 bytes).

This identifier provides absolute identification of all objects, a capability basic to all configuration management requirements.

All nodes in the data base can possess descriptors called attributes and associations. An attribute is a named character string that describes the node which possesses it. Associations establish relationships between nodes in addition to the relationships established by virtue of the groupings under directories and variation headers. Associations are represented by a list of valid ALS path names.

A derivation is a special combination of attributes and associations used in the generation of detailed histories of objects required by STONEMAN. The derivation is used to document the circumstances under which a file was created or modified. A derivation consists of the attribute derivation_text and the associations logged_inputs, derived_from, and other_inputs. Derivations are controlled by the Kernel Ada Programming Support Environment (KAPSE) and cannot be modified except by the creating tool. Files which are named in the logged_inputs association will possess a cited_by association and a derivation_count attribute which are used to prevent deletion of any file which was used to derive another. This feature prevents deletion of a source file while any executable derived from it is still in use.

The ALS also contains a sophisticated access control system based on a lock and key mechanism. Users and programs have keys and data base objects have locks. The mechanism allows for project teams as well as individual users. A more detailed explanation of ALS capabilities for configuration management can be found in Thall [3].

## Implementation Approach

While the ALS is designed with Ada and Assembler programs in mind, most configuration control mechanisms within the ALS were designed to be used for specifications, documentation, test data, etc.; in addition to the Ada source and the associated Ada compiler/linker outputs. Thus, these mechanisms are versatile enough to serve as a source code control environment for FORTRAN programs with little effort. However, an obvious requirement for FORTRAN code maintenance is access to the FORTRAN compiler and this is not provided for in the ALS. In fact the STONEMAN requirements for tool portability discourages the use of the host system (in this case VMS) which would be needed

```
   -- PACKAGE SPEC FOR HOST_ESCAPE

   -- This package is the non-shareable KAPSE implementation for the
   -- ALS host escape facility

procedure issue_host_command
              (command_line : in out string_util.var_string_rec;
              log_name      :  in kapse_defs.short_id_str;
              command_status    :  in out prog_defs.call_status_rec;
              completion_proc   : in kapse_defs.address_int;
              completion_arg    : in kapse_defs.address_int
              );

-- A VAX/VMS sub-process is created with the VAX/VMS DCL command
-- interpreter and the contents of command-line are passed to DCL to
-- execute.  The size of command_line must be less than 133 characters
-- long.  No initialization of the sub-process is performed and, in
-- particular, LOGIN.COM is not executed.  If the operation is
-- initiated successfully, request_status is set to PS_OK,
-- result_status contains a VAX/VMS 32 bit status value, and
-- result_string contains the message test that represents this
-- status.  If the operation does not initiate successfully,
-- request_status indicates the nature of the error condition, and
-- result_status and result string contain further diagnostic status
-- information.  The status returned from the sub-process is that of
-- the last operation which can be the "EXIT value" DCL command.
-- Command_line can contain the DCL command "@" (for command
-- procedures).
```

Figure 1. Issue_Host Command Specification

to access VMS FORTRAN. (The methods to be discussed could easily be applied to accessing compilers on other distributed processors, as proposed in Hargrove [4], provided that required communication links existed.) Similar problems exist in accessing the appropriate linker which resides outside the ALS environment. A mechanism also should be provided for controlled execution of test programs written in FORTRAN and the passing of data to and from the executing program.

The implementation described in this paper concentrates on accessing the FORTRAN compiler of the host and bringing the appropriate object and listing files into the ALS environment for configuration control and management. Existing ALS mechanisms and tools (as well as future tools to be developed) can then be used for this task. The same principles will apply for accessing a linker outside the ALS and executing exported FORTRAN programs.

An early version of ALS system specification [5] indicated that an ALS service called Host_Escape would be available to tool writers by calling the KAPSE from programs written in Ada. This service would provide for the execution of host operating system (VMS) commands. When the ALS arrived, an Ada subprogram specification called Issue_Host_Command was found in the KAPSE BODY of the program library. This specification is shown in Figure 1.

After many experimental attempts to use this service, it was found to be "hidden" from the general tool developer and available only within the KAPSE itself. The final ALS system specification [6] does not mention the Host_Escape capability, which could be quite useful in this application.

Another method of issuing a VMS command from within the ALS was found which is somewhat cumbersome but effective. This method involves creation of a VMS process which continually checks for the existence of a particular file. If that file exists, then it is executed as a DCL command procedure. Submission of the DCL Command Procedure shown in Figure 2 on the VMS Batch queue creates such a process.

```
$!  DCL procedure to create process which:
$!     (1) checks for existence of a file
$!     (2) executes file as a DCL procedure
$!     (3) deletes the file after execution
$!
$set default dr2:[dan.ada.als]
$!
$start:
$exist  alscom.com;1
$if $status.eq.1 then goto invoke
$wait 0:00:03
$goto start
$!
$invoke:
$@alscom.com;1
$delete alscom.com;1
$goto start
$exit
```

Figure 2. DCL Procedure to Execute File

```
--
--  command procedure to access VMS FORTRAN compiler from
--                       the ALS - p1 is first parameter of procedure call,
--                       the file to be compiled ,in this case,
--                       object file will be copied to   p1_obj
--                       listing file will be copied to   p1_lis
--
--copying FORTRAN source VMS file
              cpydata p1 ((vms))dr2:[dan.ada.als]p1
--
-- create VMS command procedure file
              edt alscom.com in=)inline
delete whole
insert 1 ; $FORTRAN/LIST  P1.
exit
              end_inline
--
              cpydata alscom.com ((vms))dr2:[dan.ada.als]alscom.com
--
--VMS process should now execute alscom.com
--
              waiting_lis : while  RSTATUS /= 1 loop
                 cpydata ((vms))dr2:[dan.ada.als]p1.lis  p1_lis
              end loop waiting_lis
--
              waiting_obj : while  RSTATUS /= 1 loop
                 cpydata ((vms))dr2:[dan.ada.als]p1.obj  p1_obj
              end loop waiting_obj
--
--            write applicable derivations and associations
--
--            Ada program tool to write derivations to new object
--              and listing files and associations to FORTRAN source
              write_fortran_deriv
--
```

Figure 3. ALS Command Procedure to Access VMS FORTRAN Compiler

Thus, when the ALS copies an ALS data base file containing one or more DCL commands to this particular VMS file, it is executed. The ALS command procedure which copied the DCL command file to VMS then checks for existence of output files from the compilation (object and listing files) and copies these back to the ALS data base. Appropriate derivation information is then added to the attributes and associations of the FORTRAN source, object and listing files. The ALS Command Procedure used for this purpose is shown in Figure 3.

## Conclusions

The implementation described above is a very basic approach to prove the feasibility of FORTRAN-based software maintenance in the ALS. Execution of the command procedures described above has proved that a FORTRAN compiler outside of the ALS environment can be accessed, and that resultant files can be brought into the ALS for configuration control and management using the wealth of tools presently available in the ALS, as well as those expected in the future. Extension of these procedures could provide for linking via an outside linker and would involve copying the appropriate object files to VMS, changing the DCL command to be executed, and returning the executable image and map to ALS. Similar techniques would be applied to copying the executable image and input test files to VMS, executing the image, and returning output test data.

Although this concept has proved feasible, it is noted that a working system as described is far from a reality. Some of the problems which may hinder the development of this working system are listed below:

-The ALS is large, therefore it is quite slow and resource intensive. Optimization will most likely occur, but it will come slowly and user acceptance will suffer as long as delays exist in system throughout and equipment costs are high.

-There are not yet enough tools (for configuration control or otherwise) available to make the effort described here cost effective. Required portability constraints will prevent immediate availability of new tools, especially those for configuration control in distributed environments.

In summary, the ALS could provide a "standard" environment for software

maintenance of all Trainer System Software in the future. Research and experimentation with the ALS in this area should continue as the Department of Defense Ada initiatives will soon force the use of APSEs for software systems (assuming acceptance of the Ada language). However, the full implementation described in this paper should be considered a long term goal.

## List of References


1. Requirements for Ada Programming Support Environments "STONEMEN", US Dept. of Defense, Feb. 1980.

2. KAPSE Interface Team: Public Report, Volumes I-3, NOSC Technical Document 509.

3. Thall, R.M. 1984. Configuration Management with the Ada Language System, Proc. of the 2nd Annual Conference on Ada Technology.

4. Hargrove, M. 1983. Baselined Software Configuration Management: An Automated Approach, Proc. of the 5th Interservice/Industry Training Equipment Conference.

5. Ada Language System Specification Prelimanary, 1983, CECOM Contract No. DAAK80-80-c-0507, CDRL Item B010.

6. Ada Language System Specification Final, Jan. 1985, CECOM Contract No. DAAK80-80-c-0507, CDRL Item B006.



Dan Dyke is an Electronics Engineer in the Software Engineering Division of the Naval Training Systems Center. Mr. Dyke holds a Bachelor of Science in Electrical Engineering degree from the University of Florida, is pursuing post graduate studies in Computer Engineering at the University of Central Florida, and is a registered Professional Engineer in the State of Florida. He joined the Naval Training Equipment Center in 1979, and has been involved primarily with Trainer System Software support and modification.