VISTA: VItro Software Test Application Program

Michael B. Paulkovich
Bill R. Brykczynski

Vitro Corporation
Silver Spring, MD

INTRODUCTION

The ultimate goal of any software element is its effective operation in a complete system. Integrating a collection of software elements into an embedded real-time system is usually a complex and time-consuming task. Therefore, thorough and comprehensive testing and evaluation of individual software elements at the procedure- or module- level prior to system integration is an effective way to minimize the number of program errors still existent at final integration. This preliminary testing normally involves tedious mathematical conversions and data analysis, and program execution and variation of inputs by test personnel on a dedicated (target) computer system.

The VISTA project involves the design of a high-order "command language" for use in controlling the execution of preliminary software testing; the software under test is thus exercised with a variety of user-specified stimuli (inputs), and a computer-analysis of the software execution and output parameters is performed and reported. This software testing is performed via simulation on a host mainframe computer.

The VISTA project was started in February 1984 as a 1-year, 2-man effort for the preliminary operational program, with continued improvement and support expected thereafter. Ada [1] was used as a PDL (Program Design Language) tool, using a classical top-down structured design approach with stepwise refinement of the design. Final implementation was also done using Ada, by refining the final (detailed) design PDL program into functional Ada code.

Additionally, the Ada PDL design program was given preliminary test/debug workouts at several intermediate levels of design, actually executing the "prototype" design program, and debugging until execution was satisfactory.

The initial VISTA program was developed on the DEC VAX 11/780 [2] using the VMS Operating System and Telesoft-Ada [3], and has been designed to be readily transportable to other host computers and systems.

The final VISTA program, to date, consists of 6 separate Ada subprograms (invoked separately in batch mode), amounting to 8500 lines of Ada code.

This paper describes our experience in utilizing Ada in such a compiler-like and analyzer/report generator applications program, and our approach, procedures, and insight gained in the areas of:

- Using Ada as a PDL
- Using Ada Design as a proto-typing (executable) program

- Using Ada as the target implementation programming language
- Design and construction of such a compiler/simulator/analyzer program.

The material in this paper is presented as follows:

I. VISTA DESCRIPTION
II. DESIGN of VISTA, including use of Ada
III. IMPLEMENTATION of VISTA
IV. CONCLUSIONS

I. VISTA DESCRIPTION

VISTA Functions

The VISTA program was designed to utilize the Navy Standard Machine Transferrable Support Software (MTASS/M) at the "core" of its processing, while VISTA provides powerful and user-friendly interfaces via the VISTA "command language" and output analyzer. The Navy MTASS/M software provides support for the computers AN/UYK-20, AN/UYK-44, and AN/AYK-14 in the languages ULTRA-16 and CMS-2/M. (These computers are widely used for real-time military Command and Control and digital signal processing applications). MTASS/M is used within VISTA to:

1) Linkload the program(s) under test with VISTA support programs (combining previously compiled relocatable object elements using the MTASS SYSGEN/M Loader/System Generator).

2) Execute multiple simulation runs (using the MTASS SIM/M Program Interpreter).

The VISTA program is cued and controlled by the input command file, which consists of source text written in the VISTA Programming Language ("VPL"). This sequence of commands will direct VISTA to load and System Generate (SYSGEN) a (previously compiled) computer program object element, and to invoke the MTASS/M Program Interpreter/Simulator (SIM/M) with a specified sequence of inputs and output/analysis options.

The program under test is then "executed" (in the simulated environment on a mainframe host) under MTASS SIM/M, and the output reports are generated following SIM/M execution.

VPL -- VISTA Programming Language

The purpose of VPL is to provide inputs to the VISTA program to control and monitor test conditions and report generation. VPL contains powerful features, providing conversions for 16-bit, 2's complement arithmetic for 1, 2, and 4 word-lengths using various format options, in any scale, with either octal or decimal notation. These conversions facilitate both the control of software stimuli and analysis of outputs. Sophisticated command constructs are incorporated into VPL which provide a convenient means for specifying complex test conditions.

VISTA Output -- Description of Reports

The primary output from a VISTA execution run is the collection of reports that is generated. These reports are a result of a detailed and elaborate analysis of the output from the MTASS simulator, and are described below.

Execution Report

This report is a matrix of all P-register addresses executed within the test program.

Non-executed Points Report

This report is a list of all points within the test program that were not executed during any portion of the test run.

Comparison/Dump Report

This report provides conversion of octal data into decimal form (if desired), and a comparison analysis of expected versus actual values.

OS Procedure Calls Report

This report provides a chronological listing of all calls made from the test program to the (simulated) real-time Operating System (WDS OS or SDEX-20), and the input parameters passed to those OS procedures.

Jump History (trace) Report

This report is a history of all jumps taken within the test program, thereby providing a condensed "trace" report.

Simulator (MTASS SIM/M) Report

This is simply the "LOG" printout from the MTASS simulator run. It can be used by the test personnel for the purpose of a detailed program trace report, or as a tool by the VISTA support/maintenance personnel when problems occur.

II. DESIGN of VISTA

HIGH-LEVEL DESIGN

The VISTA program was subdivided into 5 subprograms -- 4 of them are on the "front-end" prior to the MTASS/M interface, and the other (the Post-processor) is on the "back-end", analyzing outputs from the MTASS SIM/M run:

Lexical Analyzer	(VPL compiler step 1)
Interpreter	(VPL compiler step 2)
Assembler	(VPL compiler step 3)
Linker/Monitor	(MTASS/OS Job Control)
Post-processor	(SIM/M Analyzer and report generator)

Design of VISTA was accomplished by first creating functional specification documents and data/ interface descriptions, then structure charts, and finally Ada/PDL.

DETAILED DESIGN/CODE

PROGRAM DESIGN LANGUAGE (PDL)

The Tier Concept.

All program design was done using Ada/PDL as a "pseudocode" design language. A design document (Ada program) was constructed and refined incrementally at discreet "Tiers" of detail which correspond to predefined levels of abstraction, as follows:

Tier 0 -- Program Interaction Definition and external interfaces, including subprogram definition.

Tier 1 -- Subprogram Definition including module definition.

Tier 2 -- Module Definition and identification of procedures.

Tier 3 -- Procedure Definition.

The Tier 3 level of design is at such a level of detail that this PDL program becomes the base for the target code -- the next phase of refinement is initiated by the execution/debug of this Tier 3 PDL program, which is used as the initial version of the actual program code.

Each Tier in the PDL development process can be saved, thus providing higher-level Design Documents for life-cycle reference, and ensuring traceability of requirements to the target program code. These PDL documents can be input to automated analysis programs to provide cross-references and other design information at any Tier. For the VISTA project, the Byron [4] method of Ada/PDL design structure was used as a guideline, although no special PDL

processor or report generators were used (other than the Ada compiler for verification of syntax). Informal walk-thrus and reviews were conducted at the completion of each design Tier.

Each Tier of PDL was compiled to verify proper program structure, coupling, and symbol (label) references. Beginning at the Tier 2 level, the compiled PDL programs were actually executed until satisfactory performance was attained, e.g.,

- External (file) interfacing was verified
- No unhandled exceptions occurred
- No endless loops occurred
- Pertinent outputs were verified
- Proper program termination occurred.

Primary design concerns at the PDL and Code levels were:

a) Standard Structure Concepts -- hierarchy of functions, data coupling and isolation, cohesion, factoring, packaging:

- global constants were factored to Ada packages as a way of (1) localizing VPL Keywords and other constants to facilitate maintenance and retargeting, and (2) factoring common (redundant) data.

- data and procedures were declared at a level of scoping visibility only as high as was necessary to enable visibility to "fanned-in" references.

- procedures and functions were coupled explicitly via parameter lists; the only hidden data coupling (other than implicit "USED" package services) was the use of global constants.

- in many cases, "USED" package services were explicitly qualified (e.g., ASCII.ESC and STRING_UTILITIES.BLANKS) in order to aid program clarity, even though visibility rules were able to determine unambiguous resolution of the reference.

b) Low priority "bells and whistles" were not implemented until the initial VISTA program was completed.

c) Fan-out was kept within the "Hrair" limit (maximum of 7-9).

d) Ada procedure parameter lists were kept within the Hrair limit.

e) Design was structured such that any change proposals to VISTA performance, or changes in MTASS, would cause minimal impact. Design was kept structured and modular; foresight was applied to any areas where change would likely occur.

f) Consideration was given to portability to other host computers and operating systems.

g) Real code (vice remarks) was used in PDL as much as possible, where appropriate; also, the PDL was geared toward the goal of executing the design program as a high-level prototype program starting at the Tier-2 (module) level.

III. IMPLEMENTATION of VISTA

INTERNAL DATA STRUCTURE

In order to isolate data coupling between VISTA subprograms and to provide a re-targetable internal data structure, an intermediate "language" was developed, and dubbed "V-Code" (VISTA internal Code). This group of data files is output from the VPL Interpreter subprogram as an input to the Assembler subprogram, and contains an interpretation of all declaratives and runtime instructions for a given VPL input file.

V-Code also represents an intermediate breakdown of VPL; workload is partitioned so that the next step (VISTA Assembler) performs further processing, interfacing with the MTASS/M system. Thus, the task of the VISTA Assembler is to process the V-Code files to assemble command-streams for input to the MTASS SYSGEN/M and SIM/M support programs.

Since the output from the MTASS

simulator is ultimately destined for the VISTA Postprocessor for analysis, we also embedded cues and flags (which are implemented as comments in the simulator command-stream) to aid the Postprocessor in its analysis of the output simulator Log file.

The entire internal data structure within VISTA and between separately invoked subprograms was configured using TEXT_IO string-format files, because: (1) human-readable intermediate files are easier to "dump" or examine offline for debug and test purposes, and (2) interfaces to and from MTASS are string-format files. Thus, the VISTA project required much string-handling functions, for external inputs (VPL) and throughout internal processing and external outputs (reports); such processing is not inherently supported by built-in Ada features.

PROGRAM UTILITY PACKAGES

Program utilities were developed simultaneously beginning at the Tier 2 level of VISTA design. These utilities include:

1. VISTA-particular functions:

a) Packages DECIMAL_TO_OCTAL and OCTAL_TO_DECIMAL -- for the MTASS interface, we needed conversions for multiple 16-bit words in 2's complement format, with MULTIPLIER and SCALE arguments.

b) These functions required extensive mathematical and numeric/string conversion procedures. Note that the built-in GET and PUT procedures, which do provide numeric/string and Base conversions, were not adequate, since 2's complement, 16-bit arithmetic is not supported.

2. Functions necessary due to limitations in the Telesoft compiler:

a) VALUE/IMAGE and WIDTH attributes were not implemented (see 1b above).

b) Some other attributes were not implemented: FLOAT'SAFE SMALL and FLOAT'LARGE.

Additionally, we had to circumvent other limitations in the compiler:

c) Certain types of overloading were not implemented.

d) Generics were not implemented.

e) Data Representation Specification was not supported.

3. Limitations/deficiencies in Ada as a language.

In general, any non-trivial computer program which is to be implemented in Ada can not be efficiently produced until a library of string and math utilities has been established, in order to fill a "hole" which is inherent in the Ada language. The following packages were created for the VISTA project to meet that requirement: STRING_UTILITIES, MATH UTILITIES, VALUES, and IMAGES.

STRING UTILITIES Examples:
- UNPAD
- LEFT- and RIGHT-JUSTIFY
- MATCH
- UPPER and LOWERCASE
- INSERT STRING
- SUBSTRING POSITION
- FORCE STRING LENGTH

MATH UTILITIES Examples:
- SIGN(INTEGER)
- SIGN(FLOAT)
- ROUND(FLOAT)
- TRUNCATE(FLOAT)
- CHARACTER_VALUE(CHARACTER)
- CHARACTER_IMAGE(INTEGER)
- SQUARE_ROOT
- LOGICAL_AND(OCTAL_DIGIT, OCTAL_DIGIT)

VALUES/IMAGES:

a) Predefined Ada attributes do not allow FLOAT'VALUE and FLOAT'IMAGE; and, the TEXT_IO GET and PUT procedures for STRINGS are very unforgiving -- essentially, "Runtime Typing" has been applied, causing exceptions (DATA_ERROR). Effective

exception handling for these is more inconvenient than writing your own user-friendly conversion procedures.

b) Our IMAGE function provides alternate options in the format of the output string. Also, our IMAGE function operates in 2's complement representation for octal/binary; and VALUE operates in either sign-magnitude or 2's complement. These features were mandatory for the VISTA-SIM/M interface.

IV. CONCLUSIONS

We are able to draw the following conclusions from our experience using Ada for the VISTA project:

1. Using Ada as a PDL.

Utilizing Ada as a Program Design Language facilitated (and necessitated) structured program design, and allowed easy analysis of high-level program design aspects, such as procedure coupling and functional cohesion. Our method of step-wise refinement (the "Tier" concept) could, however, benefit by incorporating a software tool which would generate new higher-level PDL documents based upon lower-level PDL documents as an input. For example, if the Tier 3 PDL (or code-level) program is modified, we would like to be able to create the updated Tier 2 and Tier 1 documents automatically, using the Tier 3 code as the input. The feasibility of this has been studied at Vitro to a small extent, and we believe it to be quite achievable.

The use of the Ada PDL as a prototyping executable design program proved to be beneficial; in retrospect, perhaps we did not place as much emphasis on this as might be desirable. In future projects, we advise that a structured design plan be created and followed regarding the use of PDL Ada and PDL prototype testing. This plan should indicate the exact amount of detail (both code and remarks) that is required at each Tier of design, and the prototype execution testing that is required at each phase.

Obviously, the use of Ada as the target implementation language aided greatly in efficient conversion to (structured) final code. However, even if the target language was assembly (for instance), the structured form of the PDL program could be retained by the use of certain techniques such as pseudocode comments, data localization, partitioning, packaging, et cetera.

2. Using Ada as a target implementation language.

Basically, the use of Ada as the target implementation language for VISTA resulted in the same structural advantages that arise from its use as a PDL. The detailed nature of the program at code-level, however, brings about some new problems. Perhaps one of the more common complaints about Ada as a language would be the lack of string-manipulation functions; the processing of string data types is frequently required in many software applications -- especially one such as VISTA.

Additionally, regarding the lack of mathematical functions, many functions (such as trigonometric) which are normally provided in High-order languages are project-dependent, and it is debatable whether these should be included in the Ada specification. However, there are some elementary math functions which are very useful and yet require low overhead. These were listed in the discussion above.

We feel that a minimal set of these String and Math functions should have been specified in MIL-STD-1815A in packages (in the same way that packages TEXT IO, ASCII, STANDARD, et cetera were specified). Such functions are an integral part of most other quality High Order Languages. This would eliminate much of the unnecessary and redundant coding, commonly referred to as "Re-inventing The Wheel".

Notes:

[1] Ada is a registered trademark of the U.S. Government (AJPO).

[2] VAX and VMS are trademarks of Digital Equipment Corporation.

[3] Telesoft-Ada is a trademark of Telesoft Corporation.

[4] Byron is a trademark of Intermetrics, Inc., Cambridge, MA.