Ada® Test and Verification System: (ATVS)

Tom Strelich

General Research Corporation
5383 Hollister Ave.
P.O.Box 6770
Santa Barbara, CA 93111

1 Introduction

The Ada Test and Verification System (ATVS)[1] is an integrated set of software tools for testing, maintaining, and documenting Ada programs. The objectives of the ATVS are to improve the reliability and maintainability of Ada programs. GRC performed the research and analysis leading to the specification of ATVS requirements and its high-level design[2].

1.1 Background and Overview

Software testing, verification, validation, and certification are critical software development problems facing NASA. To overcome these problems, NASA has invested large amounts of time and money to correct and certify systems only to find that, when deployed, they often behave erratically or produce incorrect results. Spending more time and money on exhaustive testing won't solve the problem either since most software programs found in mission critical systems (such as the Space Station) are of such size and complexity that no amount of testing can guarantee completely correct, error-free performance. The objective then is to make the testing process as effective as possible by providing computer-aided assistance to the software engineer to help them discover the greatest number of errors for every hour spent testing.

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

[1] This work was performed under Rome Air Development Center Contract F30602-84-C-0118

[2] Ada Test and Verification System (ATVS): Final Report, General Research Corporation, CR-6-1301, September 1985.

A proven approach to software testing is the use of Automated Verification Systems (AVS). This technology was pioneered both by NASA and Rome Air Development Center, and GRC has participated actively in these efforts. For NASA, GRC developed an AVS for the AED language. For RADC, GRC developed AVS's for FORTRAN, COBOL, and JOVIAL J73 (FAVS, CAVS, and J73AVS). The ATVS represents the logical evolution of AVS technology in support of the Ada programming language.

Ada provides a high-level programming language with advanced capabilities addressing reliability issues (e.g., strong data typing, exception handlers, information hiding, etc.). However, the Ada language alone represents only a partial solution to software development problem confronting NASA: the full benefit of Ada to Space Station Software development will be realized through the synergistic interaction of the Ada language, the Software Development Environment, and supporting software tools (e.g., ATVS).

1.2 Operational Concept

Figure 1.1 illustrates the ATVS high-level operational concept:

1. Ada source code is submitted to the ATVS for Static Analysis (e.g., package dependencies, program call tree, global symbol information, data flow anomalies and errors, unreachable code, potential task deadlocks, etc.). In response to the Static analysis reports and displays, the user makes whatever corrective actions are required and repeats the process until there are no statically detectable errors in the source code.

2. The user's Ada source code is then Instrumented with run-time data collection probes which capture execution information (such as execution coverage, performance timing, and task state activity) for subsequent analysis and reporting.

3. The instrumented Ada source code is then compiled, linked, executed (with user supplied test data) with the ATVS instrumentation probes collecting run-time execution information. Assertion violations are reported to the user who may then make corrective actions and repeat the process.

4. The run-time execution data collected by the ATVS instrumentation probes is analyzed producing execution coverage, timing, and task state reports. Based on these reports the user takes corrective actions such as modifying the test data

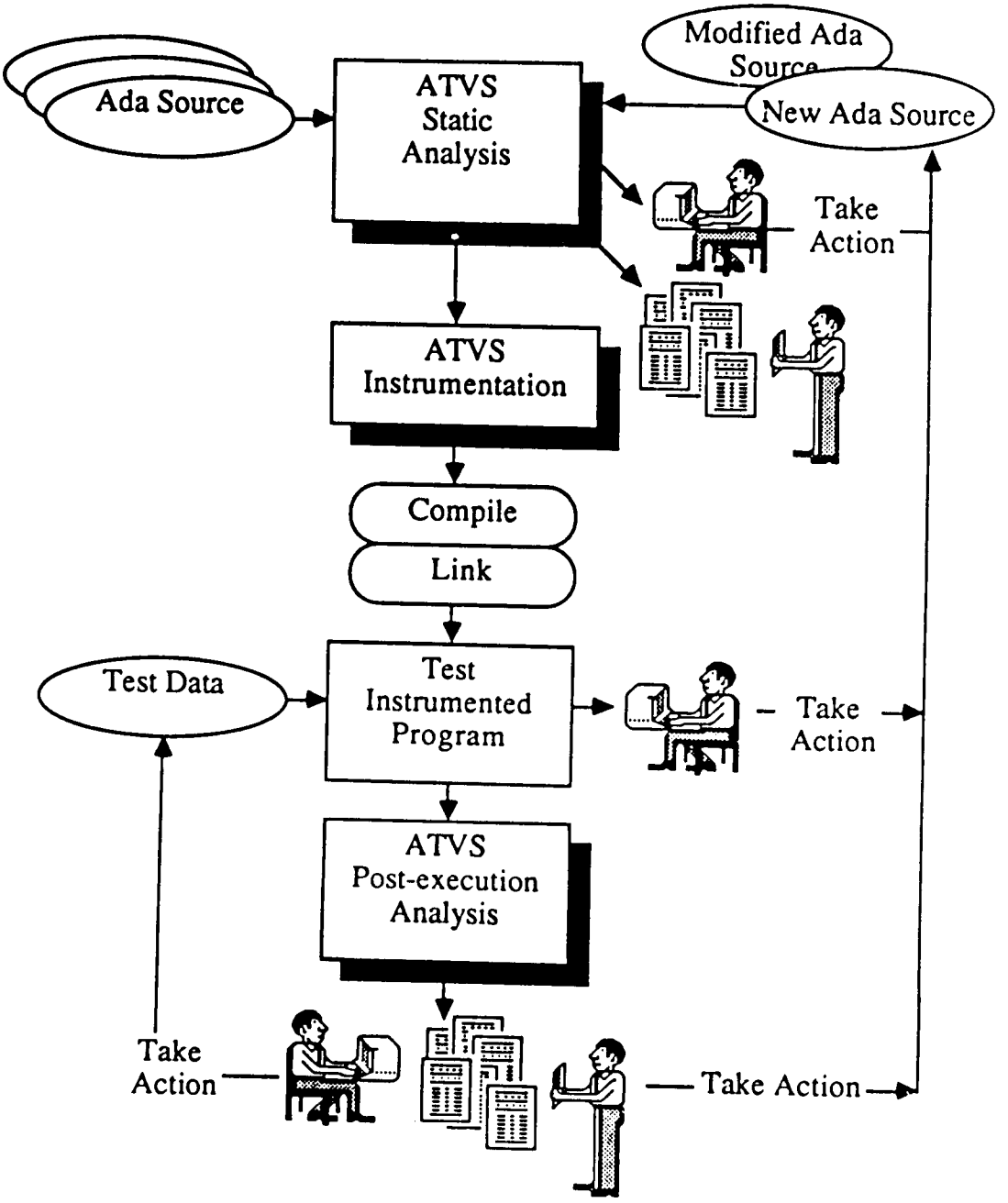

Figure 1.1. ATVS Operational Concept.

to effect execution coverage or modifying the source code to improve performance, eliminate unanticipated task interactions, and correct logic or design errors.

As suggested by the previous scenario, application of the ATVS is focused on the coding and testing phases. Figure 1.2 illustrates the role of the ATVS in the DOD-STD-2167 software development cycle: namely, Coding and Unit Testing, CSC Integration and Testing, CSCI Testing, and Maintenance Phases (while the Maintenance phase is not explicitly described in DOD-STD-2167, we have included it since the ATVS is expected to be used quite heavily for software maintenance).

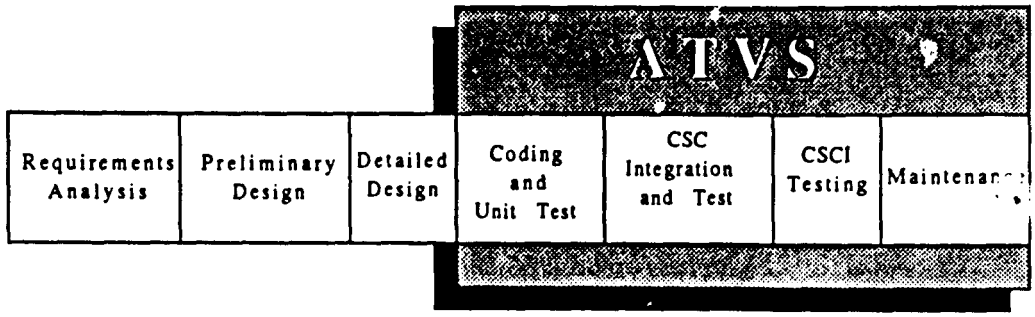

Figure 1.2. Role of the ATVS in the Software Life Cycle.

1.3 Objectives

The objective of the ATVS is to provide a set of computer-based tools which improve the reliability and maintainability of Ada software systems. The specification and design of the ATVS concentrated on the environmental context: that is, its effective integration within an advanced software development environment (such as NASA's SDE) and its contribution to that environment (e.g., support for project management, change and configuration management, test and integration, documentation, requirements traceability, etc.). The ATVS will provide detailed program information for software engineers and programmers and summary information for software project managers. The ATVS can provide management visibility by serving as a window into the software development process.

2 Capabilities

The ATVS will provide both Static and Dynamic Analysis of user programs. The requirements and design of the ATVS concentrated on providing support for the unique features of the Ada language, host-target testing issues, distributed environments, and advanced user interface capabilities.

ATVS capabilities fall into four functional groups: Static Analysis, Dynamic Analysis, Report Generation, and User Interface capabilities. Table 2.1 summarizes ATVS Functional Capabilities by group. Specific capabilities of the ATVS are described in the following paragraphs.

Table 2.1. ATVS Functional Capabilities by Group

Static Analysis	Dynamic Analysis	Report Generation	User Interface
• Source Processing	• Instrumentation -- Coverage -- Timing -- Tasking	• Automated Reports	• Batch and Interactive User Interface
• Static/Structural Analysis	• Executable Assertions	• DOD-STD-2167 Documentation	• Interactive Walkthrough
• Static Task Analysis	• Post-execution Analysis	• Prologue Insertion & Extraction	
• Programming Standards	• Unit Testing	• Software Quality Metric Data	

2.1 Static Analysis Capabilities

Ada Source Processing. The ATVS will process the Ada language and perform lexical, syntax, and semantic analysis necessary for subsequent static and dynamic analysis. It will produce a DIANA intermediate representation of the users program which will be used to build the ATVS database. The ATVS database is the central repository of program information and serves as the primary means of communication between ATVS tool components.

Static and Structural Analysis. The ATVS will provide extensive static and structural analyses concentrating on analyses unique to the Ada language. The analyses include:

- Package Dependencies -- describes "with" and "use" context clause dependencies and is valuable for change impact analysis
- Compilation/recompilation Order Dependencies -- Provided by most compilers, it is useful for maintaining system consistency subsequent to program modification
- Data Flow Errors/Anomalies -- identifies variables declared but not used, uninitialized variables, actual output parameter not set, etc.
- Global Symbol Use -- Identifiers, Types, Overloadings, Generics, Exceptions, Interrupts

Static Task Analysis. This capability identifies the set of all possible sequences of concurrency in a given program. This sequence set is then used to identify features of the program's synchronization structure such as: all possible task rendezvous, all potential areas of concurrent execution, and areas of potential task blockage (i.e., deadlock). This capability will utilize the Temporal Semantic Analysis approach described by Buhr, et al[3] .

Programming Standards Checking. This capability provides for user source code auditing against a set of modifiable programming standards. For example, "the maximum # of statements in a procedure is 25". The ATVS has defined a set of 46 programming standards.

2.2 Dynamic Analysis Capabilities

Instrumentation and Executable Assertions. Instrumentation consists of the insertion of software probes into the user source code. These instrumentation probes collect run-time program information for subsequent analysis and reporting. The types of instrumentation include: program execution coverage, program timing, and tasking activity. An executable Assertion is a statement placed in the source code by the programmer to indicate that the specific condition should exist. For example:

---

[3] Buhr, R., et al. "Experiments with PROLOG Design Descriptions and Tools in CAEDE: An Iconic Design Environment for Multitasking, Embedded Systems," Proceedings of the 7th Int'l Conf. on Software Engineering, IEEE Computer Society, 1985.

--. assert ((velocity - v_naught) > epsilon)

When violated (i.e., the statement evaluates to false) during program execution, the Assert statement can either display an assertion violation message to the user, or take some alternative action defined by the user.

Post-execution Analysis (Coverage, Timing, and Tasking). This capability processes the program execution data collected at run-time by the instrumentation probes embedded in the user's source code. Analyses include: (1) execution coverage for programs at the subprogram, branch, and statement level; (2) execution timing at the subprogram, named block, or statement level; (3) task state transitions, basically a trace of the program's tasking activity. The tasking analysis information can be used in cooperation with the static task analysis information to determine the extent of task sequence set coverage (task synchronization set coverage represents the functional analog of execution coverage in sequential programs).

The ATVS will provide data collection for both single and multiple program executions. This capability allows post-execution analyses to reflect incremental and cumulative execution coverage, timing, and tasking information. This type of historical information is an essential part of software documentation.

Unit Testing. This capability provides for automatic (with user direction) construction of Ada drivers and stubs. It will identify the undeveloped portions of a program and will construct Ada driver and stub "skeletons" which can be customized to a user's particular testing requirements. This capability supports both top-down and bottom-up development methods.

ATVS Dynamic Analyses will be supported for both host-resident and target-resident Ada programs (assuming an upload/download capability between the host and target).

2.3 Report Generation Capabilities

It is important to note that the ATVS design has separated the process of static and dynamic analysis from the process of report generation. The effect of decoupling these two activities (which communicate through the common database) is that it allows definition and incorporation of new analyses and reports to proceed independently of one another. This approach provides the flexibility necessary for the incorporation of new capabilities into the ATVS allowing it to evolve over time in response the the

environment, the user community, and advances in software engineering. Table 2.2 summarizes ATVS automated reports.

Table 2.2. Summary of ATVS Automated Reports.

Static Analysis Reports	Dynamic Analysis Reports
Summary Information Report Compilation Unit Overview Report Compilation Order Report Subprogram/Task Dependency Report Subprogram Cross Reference Task Cross Reference Package With/Use Dependency Report Package Element Set/Use Cross Reference Data Dictionary Report Global Entities Cross Reference I/O Statements Report Type Information Report Type Cross Refrence Report Object Cross Reference Report Type Derivation Report Generic Instantiation Report Exception Handling Report Interrupt Handling Report Overloading Information Report Statement Profile Report Software Metrics Report Target Code Cross Reference Data Flow Anomaly Report Programming Standards Report Source Re-analysis Report	Testcase Report Execution Coverage Summary Report Branch Coverage Summary Report Detailed Coverage Report Branch Report Reaching Set Report Execution Timing Report I Execution Timing Report II Task State Report
	DOD-STD-2167 Reports
	Calling Tree Report Functional Allocation Report Global Data Definition Report Input Data Report Local Data Definition Report Output Data Report Element Utilization Report File Description Report Record Description Report

Automated Static and Dynamic Analysis Reports. All static and dynamic analyses performed by the ATVS will be available to the user in both interactive display and hardcopy forms. The ATVS will provide 25 Static Analysis Reports and 9 Dynamic Analysis Reports.

DOD-STD-2167 Compatible Reports. The ATVS will provide nine automated reports consistent with DOD-STD-2167. These reports are variants of the ATVS automated reports and are generated from database-resident information provided by ATVS static and dynamic analyses. The separation of analysis and report generation described above allows for the definition of several reports based on the same analysis. This will allow

definition of new reports (both informal and DOD-STD) without requiring development of new analyses.

Prologue Insertion/Extraction. The ATVS supports the insertion of selected automated report information (e.g., package, subprogram, and task dependencies, global symbol use, etc.) into a prologue (i.e., a descriptive preface to a program unit). Prologues are embedded in the user's source code as Ada comments and can be augmented with user provided information. Automatic insertion of prologue information ensures current and consistent program documentation. Prologues can be automatically extracted from the source code to generate formal documentation.

Raw Software Metric Data. The ATVS will provide raw software quality metrics for analysis by other environment tools. These metrics (37 individual metrics supporting 18 software quality criteria) are consistent with the STARS Data Collection Forms, Software Evaluation Report and Software Characteristics Report[4].

2.4 User Interface Capabilities

Batch and Interactive User Interface. The ATVS will provide both a batch and interactive user interface. The batch interface will utilize a batch command language to direct ATVS processing. The full complement of ATVS capabilities (except for exclusively interactive activities such as Interactive Walkthrough) will be accessible through the batch command language.

The Interactive User Interface will be based on a hierarchical menu structure providing users controlled access to ATVS functions. There will be an extensive on-line help facility providing both reference and tutorial information. The Interactive User Interface will take advantage of advanced terminal/workstation bit-mapped graphics capabilities such as multiple windows, pull-down menus or palettes, and alternate input devices such as mice.

Interactive Walkthrough. Interactive Walkthrough replaces the manual process of "digging" through large source listings, cross reference reports, and other forms of documentation. It provides users with controlled, interactive access to the source code comprising a large software system. The user can browse the source code based on the program's call

---

[4] Interim Software Data Collection Forms Development -- Software Evaluation Report, Software Technology for Adaptable, Reliable Systems (STARS), RADC/COEE Griffiss AFB, NY, June 1985.

tree or as directed by the user, and the multiple window capabilities of the interactive user interface allow simultaneous access to various ATVS static and dynamic reports.

3 Database and Workstation Issues

3.1 ATVS Database

The ATVS database was designed as an "Entity-Relation-Attribute" (ERA) Database composed of 13 database entities and 17 associated relationships. The ERA model was selected for its expressiveness and flexibilty: The ATVS database contains a great deal of semantic program information that is best represented in the ER model.

3.2 ATVS Functional Distribution to Workstations

The ATVS was designed to operate in whole or in part on either a host machine (such as a VAX) or a microcomputer workstation (such as a SUN or VAXStation II). This flexibility allows program managers to relegate certain ATVS functions (e.g., source processing, instrumentation, etc.) to the host machine, and other functions (e.g., static analysis, post-execution analysis, interactive walkthrough, etc.) to the workstation. Microcomputer workstations often provide advanced capabilities (such as multitasking, bit-mapped graphics, multiple windows, etc.) that the host cannot easily (if at all) provide without serious degradation in system response. An additional benefit target system testing since microcomputer workstations are often used as embedded system development environments.

4 Current Status and Conclusion

The ATVS functional description and high-level design[5] are complete and are summarized in this paper. The ATVS will provide a comprehensive set of test and verification capabilities specifically addressing the unique features of the Ada language, support for embedded system development, distributed environments, and advanced user interface capabilities. Its design emphasis was on effective software development environment integration and flexibility to ensure its long-term use in the Ada software development community.

---

5 Ada Test and Verification System (ATVS): Functional Description, General Research Corporation CR-2-1301, September 1985.