CHANGES OF SOFTWARE UNIT TESTING TOOL - ATTOL TO TESTRT

Su-Hyun PARK, Soo-Yeon KANG, Koon-Ho YANG, Seong-Bong CHOI

Korea Aerospace Research Institute, psh@kari.kr.kr

ABSTRACT ATTOL is a software unit testing tool produced by the ATTOL Testware SA in France. It automates the entire software unit testing process: test plan template and test program generation, test program execution, test result analysis and test report generation. ATTOL is suited for the development of embedded software as it allows programmers to operate in native and cross development environments. Particularly, it is used for the development of the flight software which is embedded in the Communication Ocean Meteorological Satellite (COMS). As the flight software is mission-critical, it requires the strict software quality and high testing constraints. The flight software of COMS is verified by ATTOL in native and cross platforms. In 2002, ATTOL was taken over by the IBM Rational Software and has been supplied with the name of Test RealTime (TestRT). The test process of TestRT becomes different from that of ATTOL as TestRT provides the new functionalities that were absent from ATTOL. TestRT provides the new features in the test script language, as well. In this paper, we compare the test process of ATTOL to TestRT with an example of COMS and explain what has been changed in the test script language.

KEY WORDS: Software Unit Testing, ATTOL, Test RealTime

1. INTRODUCTION

1.1 Software Unit Testing

A software unit is the minimal software component (module) consisting of an application. The software unit testing is to execute an individual unit with a test plan for the verification purpose. It is typically performed by a software developer to verify that a software unit is correctly implemented.

A test plan identifies the test procedure with a set of test data. There are a lot of software unit testing tools to generate a test plan template by analyzing the source code under test. Testers manually modify the test plan generated by the automated tool. As well as the test plan generation, the unit testing tools automate the entire software unit testing process: test program generation, test program execution, test result analysis and test report generation.

Particularly, the unit testing is essential for the mission-critical software such as avionics, satellite and weapon systems. Most of the mission-critical software is embedded in the special purpose hardware. To verify the correct execution on the target hardware, the unit testing is performed on the cross development environment where the target hardware is simulated on the host computer. On the other hand, the native development environment is the general test platform to execute the software on the host computer. For the verification of the embedded software, the unit testing tools is required to provide the testers both native and cross development environment.

1.2 ATTOL

ATTOL is an automated software unit testing tool produced by the ATTOL Testware SA in France. It is suited for the development of the embedded software as it allows programmers to operate in native and cross development environment. For example, ATTOL is used for the development of the flight software of Communication Ocean Meteorological Satellite (COMS). Korean geostationary satellite, COMS has been jointly developed by Korea Aerospace Research Institute and EADS-Astrium since 2005. The flight software is embedded in COMS to control the COMS operation. As one of the mission-critical software, the flight software requires the strict software quality and high testing constraints. In case of COMS, the flight software is verified by ATTOL in both native and cross platforms.

In 2002, ATTOL was taken over by the IBM Rational Software and has been supplied with the name of Test RealTime (TestRT). While ATTOL provides the command-line interface, TestRT supports the graphical user interface as well as command-line interface for user-friendliness.

The test process of TestRT becomes different from that of ATTOL, as TestRT provides the new functionalities that were absent from ATTOL. TestRT provides the new features in the test script language, as well. In this paper, we compare the test process of ATTOL 3.3a to TestRT 7.0 and explain what has been changed in the test script language.

In this paper, we take an example of COMS. The COMS flight software is written in ADA language and verified by ATTOL. Note that both ATTOL and TestRT support C, JAVA and ADA language. In this paper, we focus on the ADA programs to illustrate the test process and to explain the test script language.

2. TEST PROCESS

Figure 1 shows the test process of ATTOL. Source code under test and the test script are input to ATTOL. ATTOL provides its own test script language to write a test plan. Like any language, the ATTOL test script is translated by the test script compiler and converted into a test program in ADA. The test program is to be compiled and then linked to all or part of the application being tested. ATTOL runtime must also be linked to the test program to take into account the nature of the execution target. A trace file is created when the test program is executed. This file will be analyzed by the test report generator.

The minor difference between ATTOL and TestRT is the format of the test report. As a test report, both ATTOL and TestRT generates an intermediate low-footprint file (.rod files), which is not human-readable. ATTOL generates the test report in text format (.ro files) in addition to rod files. On the other hand, TestRT provides a tool (rod2xrd.exe) to convert the rod file to xml format (.xrd file). The fancier test report in xml format is open with the TestRT graphical user interface.

The major differences between ATTOL and TestRT comes from the new functionalities such as Target Deployment Port (TDP) and the code coverage analysis.

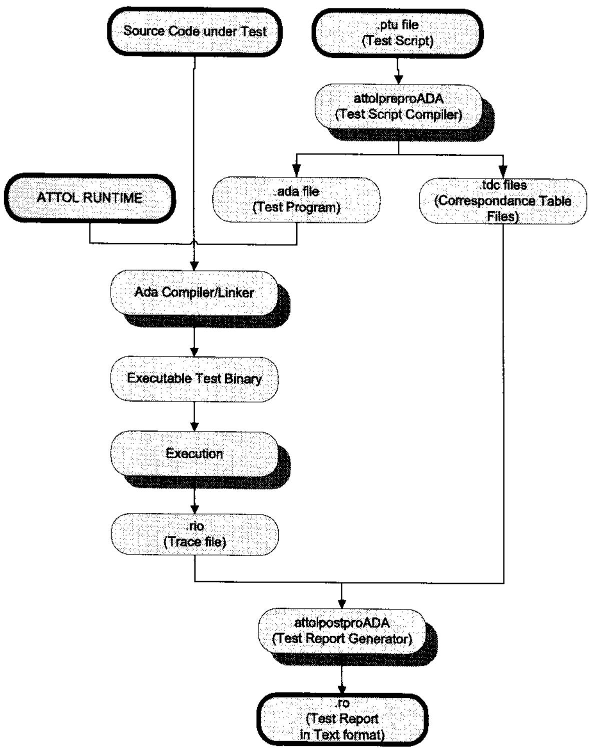


Figure 1 Test Process of ATTOL

2.1 Target Deployment Port (TDP)

ATTOL runtime is replaced by TestRT TDP. ATTOL runtime is specific to the compilation sequence and the target system. The runtime takes care of:

- the characteristics of the compiler used,
- the characteristics of the test execution target.

The test program generated by ATTOL is independent of these two elements. It is therefore possible to execute the same tests in different environments.

Likewise ATTOL runtime, TestRT TDP enables the portable tests for use in multiple environments. Testers shall customize the TDP to execute the test on a different environment. Basically, IBM provides a set of TDPs for the widely-used platforms and compiler systems.

TestRT TDP is more powerful than ATTOL runtime in that it enables the batch processing for the entire software unit testing. ATTOL automates each step of the software unit testing process: test program generation, test program execution, test result analysis and test report generation. However, testers shall process the unit testing step by step in command-line interface. On the other hand, TestRT TDP supports the batch processing for the entire software unit testing in graphical user interface. In the TDP editor, testers can specify what actions shall be done each step by the perl script. The software unit testing process of TestRT is fully automated thanks to the TDP.

2.2 Code Coverage Analysis

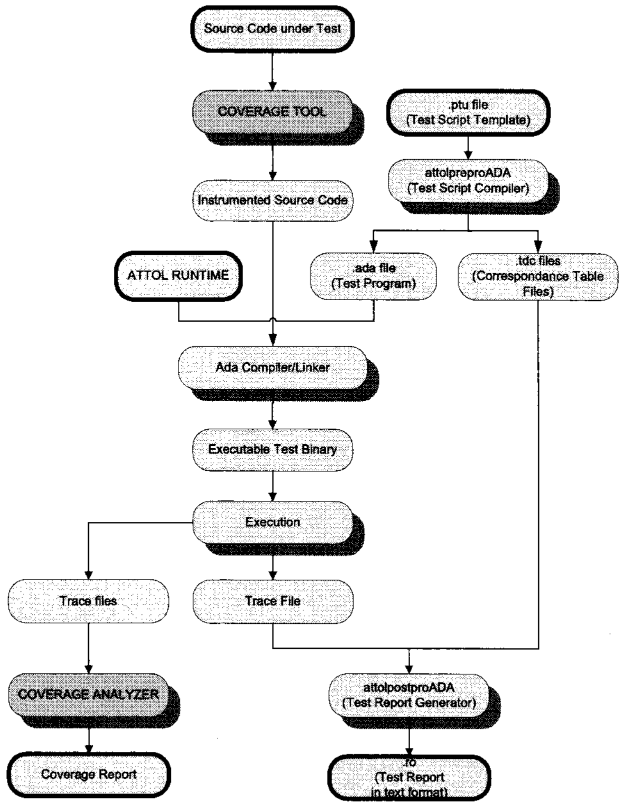


Figure 2 Coverage Analysis Process of ATTOL

TestRT provides a new functionality to analyze the code coverage. Code coverage is a measure used in software testing. It describes the degree to which the source code of an application has been tested. There are a number of coverage criteria. For example, condition coverage is used for the verification of the COMS flight software. Condition coverage criteria requires every evaluation point (such as a true/false decision) to be executed.

Most coverage tools works by adding controls to the source code under test. In case of the COMS project, ATTOL works with a coverage tool, Telelogic LOGISCOPE to check if the unit test meets the condition coverage criteria. Figure 2 illustrates how ATTOL works with a coverage tool. The coverage report is produced by the coverage analyzer as the result of the test.

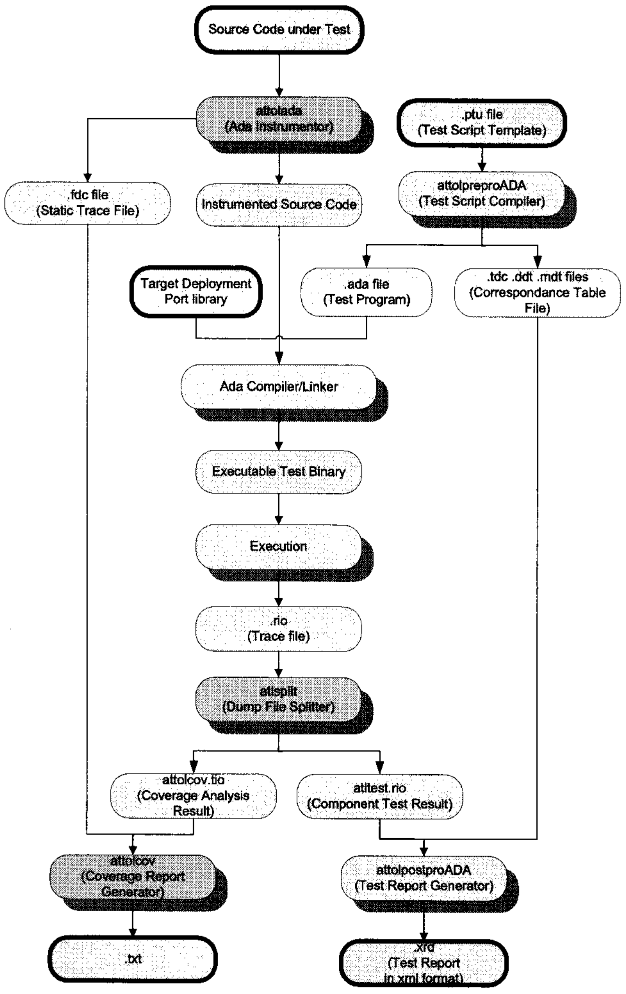

Source Code under Test
attolada (Ada Instrumentor)
.ptu file (Test Script Template)
.fdc file (Static Trace File)
Instrumented Source Code
attolpreproADA (Test Script Compiler)
Target Deployment Port library
.ada file (Test Program)
.tdc .ddt .mdt files (Correspondance Table File)
Ada Compiler/Linker
Executable Test Binary
Execution
.rio (Trace file)
atlsplit (Dump File Splitter)
attolcov.tio (Coverage Analysis Result)
atltest.rio (Component Test Result)
attolcov (Coverage Report Generator)
attolpostproADA (Test Report Generator)
.txt
.xrd (Test Report in xml format)

Figure 3 Coverage Analysis Process of TestRT

TestRT provides the coverage analysis function without any coverage tool. TestRT includes the source code instrumentor (attolada.exe) and the coverage report generator (attolcov.exe) in itself. The dump file splitter (atlsplit) splits the trace file into the component test result and the coverage analysis result. Figure 3 shows how TestRT performs the coverage analysis.

3. TEST SCRIPT LANGUAGE

The test script language of ATTOL is almost same as that of TestRT. Generally, the ATTOL test script can be run with TestRT. The biggest difference is the grammar for the complex stub definition.

A stub is a dummy software component designed to replace a component that the code under test relies on, but cannot use for practicality or availability reasons. A stub can simulate the response of the stubbed component. Figure 4 illustrates the concept of the ATTOL stubs.

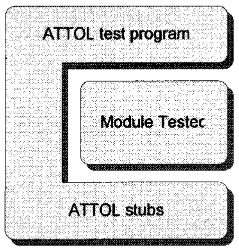
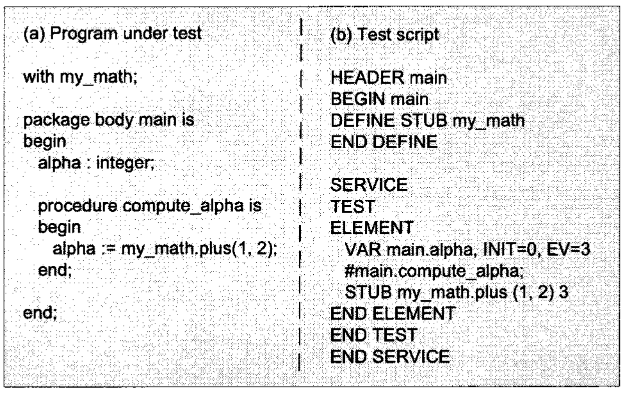


Figure 4 Concept of ATTOL stubs

Figure 5 shows the example of the ATTOL stubs. The tested unit, main calls the functions from the external unit, my_math. In order to isolate main unit from the possible anomalies or regression from my_math, the test script (b) defines the my_math as a stub. When the stubbed function is called, ATTOL checks if it is called with the correct input parameters (1, 2) and simulates the expected return value (3).



```
(a) Program under test               | (b) Test script
                                     |
with my_math;                        | HEADER main
                                     | BEGIN main
package body main is                 | DEFINE STUB my_math
begin                                | END DEFINE
  alpha : integer;                   |
                                     | SERVICE
  procedure compute_alpha is         | TEST
  begin                              | ELEMENT
    alpha := my_math.plus(1, 2);     |   VAR main.alpha, INIT=0, EV=3
  end;                               |   #main.compute_alpha;
                                     |   STUB my_math.plus (1, 2) 3
end;                                 | END ELEMENT
                                     | END TEST
                                     | END SERVICE
```

Figure 5 Example of ATTOL stubs

If necessary, testers can make the stub operation more complex by inserting native code into the body of the simulated function. The biggest difference in the test script language is the grammar for the complex stub definition. The test script language of TestRT becomes more rational than that of ATTOL by adding new features and by removing old features.

3.1 Global Simulated Variables

In the ATTOL test script, the stub definition can access the variables from the package, which will be generated by the test script compiler. TestRT, however, does not support this feature any more. Instead, TestRT provides the new feature in the test script language, i.e. the global simulated variable.

In Figure 6, the ATTOL test script (a) defines the stub with the variable of the package TTest_var_simule_S_1, which will be generated by the test script compiler. The variable Idx stores the number of the calls of the stub KERNEL.SEND_NF_RT_CMD. Logically, it doesn't make sense that the ATTOL test script accesses the variable which does not even exist before the compilation of the test script.

TestRT does not allow the stub definition to access the variables, which does not exist. Instead, testers can simulate the global variables by declaring them in the DEFINE STUB block. The global simulated variables are used by the functions under test. For example, the TestRT test script (b) declares a global simulated variable, nb_call_SEND_NF_RT_CMD to count the number of the stub calls. Whenever the stub is called, the variable nb_call_SEND_NF_RT_CMD is incremented by one. Eventually, the TestRT test script (b) provides the exactly same function as the ATTOL test script (a) by the global simulated variable.

```
(a) ATTOL test script

DEFINE STUB KERNEL
#procedure SEND_NF_RT_CMD (...) is
#begin
#  if (TTest_var_simule_S_1.TTest_var_simule_S_1_SEND_NF_RT_CMD Idx <= 4)
#  then
..
END DEFINE
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
(b) TestRT test script

DEFINE STUB KERNEL
#nb_call_SEND_NF_RT_CMD  integer := 0
#procedure SEND_NF_RT_CMD (...) is
#begin
# nb_call_SEND_NF_RT_CMD := nb_call_SEND_NF_RT_CMD + 1
# if (nb_call_SEND_NF_RT_CMD <= 4)
# then
..
END DEFINE
```

Figure 6 Stub definition of ATTOL vs. TestRT

Note that the global simulated variables are not accessible by the test procedure. That is because the test script compiler does not add the declaration of the global simulated variables to the stub specification, but to the stub body. The global simulated variable can be regarded as the stub-scope variables in that they are visible by the stub functions and the stub procedures only.

3.2 Communication between stub and test procedure

ATTOL runtime provides an integer array, attol.user_int for communication between the stub and the test procedure. TestRT does not support this feature any longer. In this paper, we propose an alternative plan to substitute this feature.

In Figure 7, the ADA specification attol.ads (a) is a part of ATTOL runtime. It declares an integer array, user_int for communication between the stub and the test procedure. The ATTOL test script (b) defines the stub with the reference to attol.user_int(1). The variable attol.user_int(1) can be read and written by both the stub definition and the test procedure. Note that the ATTOL test procedure can access the variable of the attol package without the reference phrase '#with attol'. Logically, it doesn't make sense in that the test procedure accesses the variable of the ATTOL runtime, which will be linked to the test program later.

The test script language of TestRT is more complete than ATTOL because it does not allow to access the variable of the TestRT TDP (ATTOL runtime). Ideally, the test procedure shall be independent from the TestRT TDP.

For the communication between the stub and the test procedure, testers can define the extra ADA specification, global.ads with a global variable user_int and then access the global.user_int instead of attol.user_int. Note that both the test procedure and the stub definition shall include '#with global' phrase.

```
(a) attol.ads

package attol is
 user_int : array(1..10) of integer;
 ...
end attol;


(b) ATTOL test script

DEFINE STUB KERNEL
#procedure READ_CURRENT_DATA(...) is
#begin
...
#  VALUE_TO_READ := attol.user_int(1);
...
END DEFINE

SERVICE
TEST
ELEMENT
 ...
 attol.user_int(1) := 1;
 STUB KERNEL.READ_CURRENT_DATA(···)
 ...
END ELEMENT
END TEST
END SERVICE
```

```
(c) global.ads

package global is
 user_int : integer :=0;
end global;


(d) TestRT test script

#with global;

DEFINE STUB KERNEL
#with global;
#procedure READ_CURRENT_DATA(...) is
#begin
...
#  VALUE_TO_READ := global.user_int;
...
END DEFINE

SERVICE
TEST
ELEMENT
 ...
 global.user_int := 1;
 STUB KERNEL.READ_CURRENT_DATA(···)
  ...
END ELEMENT
END TEST
END SERVICE
```

Figure 7 attol.user_int and global.user_int

4. CONCLUSION

A software unit testing tool, ATTOL evolved into TestRT in the point of the test process and the test script language. The test process of TestRT becomes more powerful than ATTOL in that it provides the new functionalities such as the target deployment port and code coverage analysis. The test script language of TestRT is more complete than that of ATTOL with respect to the complex stub definition.

References

2000. ATTOL UniTest Programmer's Guide v3.4, *ATTOL Testware*

2006. IBM Test RealTime User Guide v 7.0.0, *IBM Corporation*