Study on real-time test script in Automated Test Equipment

Conference Paper · August 2009

DOI: 10.1109/ICRMS.2009.5270090 · Source: IEEE Xplore

4 authors, including:

Bin Liu
Beihang University (BUAA)
71 PUBLICATIONS **356** CITATIONS
SEE PROFILE

Yongfeng Yin
Beihang University (BUAA)
18 PUBLICATIONS **87** CITATIONS
SEE PROFILE

Study on Real-Time Test Script in Automated Test Equipment

Chongwu Jiang, Bin Liu, Yongfeng Yin, Chang Liu
Department of System Engineering of Engineering Technology
Beihang University
Beijing, China
jcw@dse.buaa.edu.cn

***Abstract*—In this article we propose a generic test script for real-time embedded software system testing, which has been applied to ATE (Automated Test Equipment). After a summary of the theory about embedded software automated test based on test script, the design philosophy and implementation details are described. We have chosen an ATE and integrated python interpreter into it. Modules for test control are developed on base of python's expandability. To ensure real-time, we have trimmed python interpreter's source code. This test script has advantages of simple, flexible, controllable, reusable and portable. The use of third party python tools results in decreased test script development time. A number of real-time embedded systems are tested by this script technology. Both the correctness and the real-time performance are validated.**

Keywords-automatic software testing; test script; real-time embedded software; simulation testing environment; python

I. INTRODUCTION

Most of embedded systems have features of hard real-time and resource-constrained, and usually require high reliability. System test is an important means to improve the quality of embedded software, and aims at finding the inconsistencies or contradictions between software specification and application[1]. Testers put SUT (System under Test) which has passed integration testing in a real or simulated runtime environment, and then send input to SUT, collect its datas and compare them with test oracle. ATE (Automated Test Equipment) is widely utilized in embedded software system testing, and one critical component of ATE is test script interpreter. Test script is the scenario directing test process running normally, which includes datas and instructions with regular grammar, and plays a key role in automated software testing. It is an effective method using scripts to describe test cases and executing them by interpreter in ATE. Test script usually describes the operations of one or more test cases in text form, to drive SUT to run according to tester's intents, and then the consistency of test result and test oracle can be verified.

Reference [2] and [3] discussed the requirement for test script in ATE, that is, ATE needs a real-time, flexible script language for embedded software system testing. Also, since engineers using the script language have varying levels of experience and work on multiple projects, it is very necessary that test scripts be validated early in the test development cycle. ATE also wants to increase the possibility of easily reusing test scripts previously developed. A solution to these needs is a test script language implementation based on Python.

Python is an object-oriented, dynamic semantics, cross-platform, open-source script language with beautiful syntax. Theoretically, python program can be compiled and run in any platform (including a wide variety of embedded operating systems, such as the Palm OS, VxWorks, etc)[4]. With the API provided by platform, python can be expanded functionally in C or C++ language, which not only preserves features of convenient and flexible, but greatly improves the operating efficiency.

In this paper, a new universal test script is proposed for real-time embedded software testing, it is based on python and taking into account the characteristics of embedded system. Then we describe the design and implementation of the technology in detail, at last the correctness and effectiveness of this test script are validated through a test project of embedded software.

II. THEORY OF EMBEDDED SOFTWARE AUTOMATED TEST BASED ON TEST SCRIPT

Reference [5] and [6] provided the definition of ATE: ATE is a kind of automated device that is used to quickly test a wide range of electronic devices and systems. More broadly conceived, ATE can be regarded as a virtual instrument with more complex functions and customized by the user. ATE has functions including stimulus, measurement, switch, power, interface, etc, most of which can be implemented by means of programming with support of minimum number, inexpensive, commercial hardware. The hardware resources of ATE are utilized to the full. Consequently, the overall cost of test is significantly reduced.

As a typical ATE, ESSTE (Embedded software simulation testing environment) is often used to take a real-time, non-invasive system testing for embedded software. ESSTE can simulate the running environment of SUT, and drive SUT to run in accordance with the description of the test cases by means of providing test inputs to SUT through interfaces and collecting SUT's output[7].

978-1-4244-4905-7/09/$25.00©2009 IEEE

As we know, system testing is concerned with testing an entire system based on its specifications. When embedded software is tested through ESSTE, testers first analyze the relevant documents such as requirement specification and ICD (Interface Control Document), then construct the models of SUT's runtime environment on the basis of ICD, which simulate the behaviors of equipments communicating with SUT, therefore, SUT can run in a virtual environment constructed by ESSTE. So, according to the requirement specification, testers can compose various types of system test cases.

As an indispensable component of ESSTE, test script interpreter makes it possible that test case can be executed automatically. Fig 1 shows the process of automated system testing based on test script.

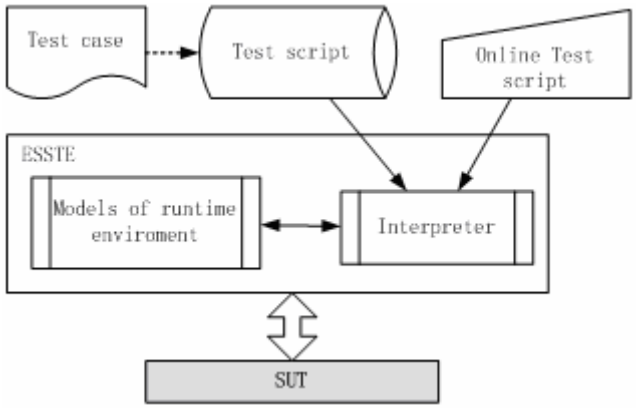

Fig 1: Automated system testing based on ESSTE

III. DESIGN AND IMPLEMENTATION OF TEST SCRIPT FOR REAL-TIME EMBEDDED SOFTWARE

A. Extending python for software system testing

In general, it is necessary for test script to contain datas and instructions such as control information (stimulus to SUT), time (when to input and the laws of the data inputting), data collection (get output of SUT), and information about arbitrating (how to determine whether a test is passed), etc. Embedding Python in ESSTE and expanding it for more functions can make python as a fully functional test scrip language.

In order to make python meet the requirement of real-time embedded software system testing with support of ESSTE, we have accomplished the following specific work:

1) Embed python interpreter into ESSTE program, so that test scripts may be executed by means of invoking the API functions provided by interpreter.

In order to ensure real-time performance of interpreter, we have python source code trimmed. As result, only core interpreter and a few necessary modules are kept.

2) Developing extension modules through C/C++ to add testing functions (In windows, extension modules usually exist in form of .PYD files). Testers may use "import" statement to add testing modules to python interpreter, and then a variety of testing functions provided by modules are available in test scripts.

3) Giving ESSTE ability to call procedures defined in python scripts. Testing tasks can be described by way of python functions, thus ESSTE can schedule these tasks through the callback mechanism.

B. Details of extending for real-time embedded software testing

In this section, some python extension has been made in detail. Concretely, we have added some modules on the basis of python's morphology and functions to control testing process, including the implementation for inputting test data, accessing SUT's output and precise time controlling.

(1) Input test data

In ESSTE, only equipment models can communicate with SUT, through equipment models test scripts transfer kinds of stimulation to SUT. By way of extension, we encapsulate the interfaces of model controller in ESSTE. So test scripts can change equipment models' variables on interface through method "set_variable" provided by extension "ts", the simulation model will transfer data automatically to SUT in accordance with the communication protocol. For instance:

```
import ts  # Import test script module
ts.set_variable('DCMP', 'ins_main_mode', 1) # send data to
   "INS" via "DCMP", the main mode of "INS" will be set to 1
```

(2) Access SUT's output

Similarly, it is through equipment models that test scripts can access SUT's output data. Simulation models receive and store data from SUT in accordance with the communication protocol, test scripts can get models' variables through method "get_variable". For example:

```
import ts  # Import test script module
H = ts.get_variable('FCS', 'ins_H') # get height of "INS" via
   "FCS", and save it to variable "H"
```

(3) Time control

Generally, there are two kind of time control for test tasks: cycle-type and timing-type. Cycle-time means that test task will be scheduled periodically between its start time and end time in accordance with the frequency predefined, while timing-type means that test task will run only once when it's time is reached.

In this paper, test tasks of cycle-type and timing-type are implemented on the basis of delegation pattern, as shown in Fig 2:

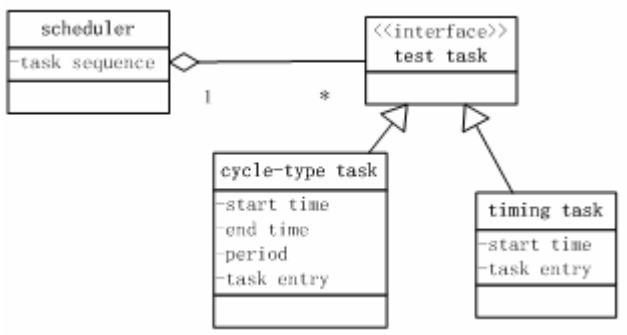

Fig 2: Test tasks' implementation

Testers can describe test task by means of python function, and then register task's attributes including type, time, and period, etc to ESSTE, through method "task_register" provided by module "ts". When testing is started, ESSTE will automatically release signals to kinds of threads which imply test tasks and consist of python functions according to their time characteristic. This is an example:

```
import ts  # Import test script module
def task_perd():
      # The content of test task is ignored here

# The following code example registers the test task
   "taskFly" to ESSTE. The task will be scheduled per 50
   millisecond between 60 seconds and 180 seconds.
ts.task_register('taskFly',
       50,     # cycle-type task, period: 50ms
       60000,  #start time, 60s
       180000  #end time, 180s
       )
```

For simple time delay operation, it is tedious to use the mechanism of delegation and schedule provided by ESSTE. Extension "ts" provides a method named "delay" for time delaying in millisecond level. The interpreter thread will be suspended when delaying, CPU resource won't be wasted.

4) Test assertion

It is necessary to analyze the test result to judge whether the test case being executed should be passed. We can use the method "assert" provided by extension "ts" to affirm expected test results. Whichever assert statement fails, the test case will be deemed to failed, and the conclusion will be wrote to the database of test result, also, the test case being executed will be terminated. This is an example:

```
import ts  # Import test script module
ts.set_variable('RC', 'pitch', 5) # Set the pitching angle of
   aircraft to 5 degree by sending command to 'FCMS'
   through model 'RC'(Remote Control)
ts.delay(5000) # Delay 5 seconds

pitch = ts.get_variable('AIRCRAFT', 'pitch') # Get the
   pitching angle of aircraft
ts.assert(4.9<pitch<5.1) # 5 seconds later, if the pitching
   angle is not in the correct range, judge this test case failed
```

(5) Test scripts' reusing

We add method "run_script" to module "ts" in order to implement test scripts' reusing.

```
import ts  # Import test script module
ts.run_script('script1')  # Test script "script1" will run here
```

C. *Interface for test scripts in ESSTE*

In ESSTE, test scripts are executed by interpreter. First, interpreter comprehends the meaning of the test script, and then directs equipment models by model controller according to test inputs. In this way, the goal of communicating with SUT is achieved.

The interpreter is a crucial component of ESSTE. Fig 3 shows the relationship of interpreter and other components:

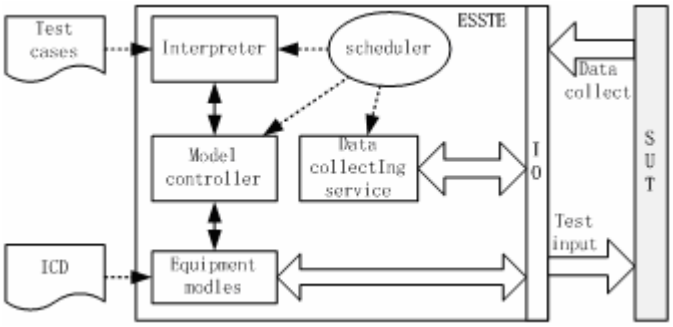

Fig 3: Structure of ESSTE

Model controller is designed to control and manage all of the simulation models, and plays a role of communication bridge between test scripts and equipment models. Through the interfaces provided by model controller, test scripts can set models' status and access models' information.

In order to make test script interpreter has ability to control equipment models, model controller is required to provide the interfaces as shown in Table 1.

Table 1: Interfaces provided by model controller

Interface	parameter	return value	function
get_time	none	Time since testing started (ms)	Get current time since testing started in ESSTE.
get_model	Name of model	Model's handle	Get the handle of a model according to its name.
get_variable	Name of Model Name of Variable	Variable's handle	Get the value of a variable according to its name and model's name.
set_model	Model's handle New status	none	Set the equipment model to a new status.
set_variables	Variable's handle New value	none	Set the variable to a new value.

D. *Test script interpreter*

Test script interpreter's function is to invoke various kinds of functions, analyze and dispatch real-time test data in

accordance with the test statements, so that the testing actions are carried out truly. Generally in ESSTE, the interpreter runs on a specific RTOS (real-time operating system).

We have expanded the core python interpreter to add operation primitives to make simulation models controllable. And also, some APIs (Application Programming Interface) of RTOS are encapsulated to make test scripts have ability to control the tasks (threads) of RTOS, including operations for semaphore, message queue and task priority. Thus the controllability and flexibility of test script are improved.

IV. EXAMPLE FOR EVALUATING

In this section, we present an example to illustrate the validation of python test script technology described above within the GESTE toolset (A kind of ESSTE developed by Reliability Engineering Institute of BeiHang University).

The following are the main content of our work:

1) Integrating python core interpreter to the simulation test platform. GESTE's RTOS is VxWorks. Reference[8] discussed how to integrate python interpreter into VxWorks.

2) Developing extension module of python for testing. Extension mainly encapsulates the interfaces of model controller and APIs for real-time task, semaphore and message queue of VxWorks.

We have selected an FCMS (Flight Control and Management System) of UAV (Unmanned Aerial Vehicle) as SUT to validate the python test script technology.

FCMS is a typical real-time embedded system. It plays a role of control center in UAV with many important functions such as stability augmentation, automatic pilot, and intelligent flight management, etc. As a subsystem of avionics, FCMS communicates with other subsystems including INS(Inertial Navigation System), GPS, ADC(Air Data Computer), RA(Radio Altimeter), RC(Remote Control), Telemetry, Rudder, Engine, etc. The information interchange between FCMS and other subsystems is shown in Fig 4. FCMS gathers data from all kinds of sensors and receives instructions from remote control, then send the corresponding commands to rudders and engine, and reports current status of UAV through telemetry. There is a higher real-time request for FCMS whose operation cycle is usually less than 25ms.

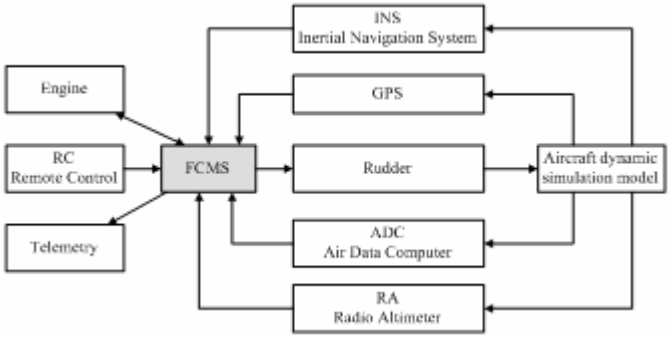

Fig 4: Communication between FCMS and other subsystems

Here is an example for system test case of FCMS:

1) Execute a test case named "takeoff_airline", to make UAV fly along the airline preinstalled;
2) Confirm the aircraft's status, if it's not flying airline, record this test case failed;
3) From the testing began between 120s to 720s, make communication failure(Set the fault flag of remote control to 1);
4) During communication failure, check the status of the aircraft per second, UAV should make a left circled flight(The roll angle should be held at -5 degree);
5) 10 minutes later from communication failure, confirm whether the plane returns to base (Flag "return_to_base_flag" should be set to 1, which can be watched from telemetry);
6) At 730s, recover the communication (Set the fault flag of remote control to 0);
7) Send a command of "hold height at 2000 meters" to FCMS through remote control;
8) At 800s, confirm if the height of UAV is held at 2000m (The error is less than 20m). If the height is not in the correct range, record this test case failed.

We can describe this test case in python script language, as follows:

```
import ts  # Import test script module
ts.run_script('takeoff_airline')  # Test script "takeoff_airline"
   will be invoked
mode = ts.get_variable('TELEMETRY', 'mode') # Get the
   working mode of FCMS from 'TELEMETRY'
ts.assert(mode == 0) # Confirm the aircraft is flying along an
   airline

def task_rc_fail():
  ts.set_variable('RC', 'fault_flag', 1) #Make communication
    failure(Set the fault flag of remote control to 1)
  if ts.get_time()>125000
    roll = ts.get_variable('INS', 'roll') #Get the roll angle of the
    aircraft from model "INS"
    ts.assert(-5.2 < roll < -4.8) # Affirm UAV is left circling
    (The roll angle should be held at -5 degree, and error is less
    than 0.2 degree)
  if ts.get_time() >= 720000
    ts.delay(1000)
    flag = ts.get_variable('TELEMETRY', 'return_to_base_flag')
    # Get the flag "return_to_base_flag" from model
    "TELEMETRY"
    ts.assert(flag == 1)

# The following statements register the task named
   "task_rc_fail"
ts.task_register('task_rc_fail',
       1000,    #period, 1s
       120000,  # Start time, 120s
       720000   # End time, 720s
       )

def task_rc_recovery():
```

```
  ts.set_variable('RC', 'fault_flag', 0) # recover the
    communication
  ts.delay('1000')
  ts.begin_block_assign('RC', 'command')
  ts.set_current_block('instruction', 0x6B) # Instruction of
    holding height
  ts.set_current_block('h', 2000) #hold the height at 2000m
  ts.end_block_assign()
  ts.delay(70000)
  h = ts.get_variable('ADC', 'h') #Get the height from model
    "ADC"
  ts.assert(1980 < h < 2020)

# The following statements register the task named
    "task_rc_recovery"
ts.task_register('task_rc_recovery',
       0,      # "0" means that it's a timing task
       730000,  # Start time, 730s
       -1      # end time, "-1" means never end
       )
```

In this test script, 'RC', 'ADC', 'fault_flag', etc. are symbols of models and variables created in the process of constructing test environment through GESTE.

In order to make testing more efficiently, we can get the handle of the model or variable by methods "get_model" and "get_variable" provided by module "ts", in this way, models and variables can be accessed through their handles instead of names in further statements.

V. CONCLUSION

There are many benefits obtained by using python as a test script language. Scripts are simple, flexible and real-time. We have tested the time performance of test script interpreter in GESTE by using CodeTest (A performance testing tool for embedded software). Execution time of medium-scale test script (Example as above) does not exceed 10 ms, and timing error is less than 1ms. More detailed test data are shown in Table 2 (Computer configuration: CPU P4 2.8G, RAM 1G, OS VxWorks).

Table 2: Time performance of python

Test item	execution time
100 sequential order statements (No conditional statement, no loop statement, no function call)	3.368 ms
1000 sequential order statements (No conditional statement, no loop statement, no function call)	19.841 ms
2000 loops (Each loop implement a simple statement)	2.342 ms
150 conditional statements	7.341 ms
100 function calls (The function is simple with one formal parameter and one return value)	2.637 ms

From Table 2 we conclude that python as a test script can meet the test requirement of those embedded softwares whose real-time request are of millisecond level. Furthermore, there are many methods and tools such as Psyco, Pyrex, etc. can accelerate the speed of python interpreter. We are going to test and utilize these tools in our future work to further improve the efficiency of the test script.

And also, use of third party python development tools helps speed the development and validation of test scripts. These tools can be found easily in open source software community. Since Python is standardized and cross platform, the potential for test script portability and reusability is better than using a non-standard scripting language. In short, the test script described in this paper can meet the requirements of real-time embedded software system testing.

REFERENCES

[1] J. Hartmann, M. Vieira, H. Foster, and A. Ruder, "A UML-based approach to system testing," Innovations in Systems and Software Engineering, vol. 1, p. 12-24, 2005.

[2] D. J. Johnson and P. Roselli, "Using XML as a flexible, portable test script language," in AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference. Proceedings, California, 2003, p. 187- 192.

[3] H. J. Zainzinger and S. A. Austria, "Testing Embedded Systems by Using a C++ Script Interpreter," in Proceedings of the 11 th Asian Test Symposium(ATS'02), 2002, p. 380 - 385.

[4] G. Lindstrom, "Programming with Python," IT Professional, vol. 7, p. 10-16, 2005.

[5] E. H. Dare, "Automated test equipment synthetic instrumentation," in Autotestcon, 2005. IEEE, Orlando, 2005, p. 175- 179.

[6] D. B. Droste and B. Allman, "Anatomy of the next generation of ATE," in Autotestcon, 2005. IEEE, Orlando, USA, 2005, p. 560- 569.

[7] D. Zhong, B. Liu, and L. Ruan, "Research on software architecture of embedded software simulation testing environment," Journal of Beijing University of Aeronautics and Astronautics, vol. 31, p. 1130-1134, 2005.

[8] S. Raskin (1999). "How to port Python to VxWorks," Available from: http://mail.python.org/pipermail/python-list/1999-May/003929.html. [Accessed on 7th May, 2008].