Hybrid Automatic Test Technology for Spacecraft

Wei Lv*, Hongjiang Song, Xiaoyu He
Beijing Institute of Spacecraft System Engineering, Beijing 100094, China
*Corresponding author’s e-mail: lw_bit@163.com, lvwei@cast.casc

Abstract. A new hybrid test case design and execution technology based on component and script is proposed, which combines test component with test script to form test case and realize automatic test of spacecraft. The common test logic is described by dragging components and editing component parameters; for the special and complex logic test requirements, the method of writing script is used to describe. Components and scripts can be mixed in the same test case to describe complex and changeable test cases, and can be executed automatically.

1. Introduction

A lot of test work is needed in the process of spacecraft development, and automatic test technology has been widely used[1]. The existing automated testing technology can be divided into script based[2] and visual component-based[3]. Script syntax is flexible, description function is strong, but easy to use and intuitive, and inconvenient to maintain; visual components are simple and intuitive, and can realize WYSIWYG test case editing, but the test process contained in the components is fixed, and the system function is restricted by the size of the component library, so it is not flexible and easy to expand. Spacecraft test projects are numerous, logic is complex and changeable, it is difficult to edit and maintain a large number of test scripts, and the solidified visual components are not easy to meet the new test requirements.

Aiming at the shortcomings of the existing technology, a spacecraft automatic test technology based on the combination of component and script is proposed. The test script and test component are combined to form test cases to realize the automatic test of spacecraft.

This technology supports adding visual test components and test scripts to a test case at the same time. The common test logic is described by dragging components and editing component parameters; for the special and complex logic test requirements, the method of writing script is used to describe. Components and scripts can be mixed in the same test case to describe complex and changeable test cases, and execute test cases automatically through execution environment.

2. Principle of test component and test script technology

By summarizing, analyzing and refining the spacecraft test items, some common test operations can be summarized, such as sending commands, judging whether the telemetry parameters are in the required range, etc. Through the combination of these operations to complete the test of a project. Because of the reusability of these operation methods, the implementation of these test operations can well meet the needs of spacecraft test, and it is also of great significance to ensure the reliability of spacecraft test.

For some special functions, the complexity of software implementation is high, and it cannot bring significant efficiency improvement. Therefore, the editing method of test script is provided to describe the test logic, which not only avoids the work of software development and modification, but also...
ensures better coverage of automatic test.

2.1. Test components
Test components represent some basic test methods in spacecraft test. The control structure defines the execution mode of test components in spacecraft test. The control structure includes serial, parallel, branch, and loop. When editing a spacecraft test case, testers use the defined test components and control structure information to complete the specific editing of a test case. Then the test execution software parses the user-defined test component combination to complete the corresponding project test.

A complete test component definition includes test component description, test component attributes, input mode, execution mode, storage mode, return result information and display attributes. The test component description defines the functions that the test component should implement; the attribute of test component defines the information of all components of the test component; input method defines the input contents and methods of all elements of the test component; execution mode defines the implementation process of the test component and its display content; the storage method defines the running result information of the test component; the return result information defines the information that needs to be returned according to different execution results after the execution of the test component; the display property defines the content and related styles that the test component can display in the report generation and query side. When analyzing the requirements of each test component, the following describes each test component according to the above elements.

The parallel execution of test components is realized by setting the synchronous and asynchronous execution mode of test components. In the properties of each test component, an attribute "whether the test component is executed asynchronously" is defined. After defining the synchronous and asynchronous mode of the test component, the parallel execution of the synchronous and asynchronous test components can be realized. The specific execution process is that when the execution engine detects that a test component is asynchronous, it will continue to search the execution mode of the subsequent test component in order until it finds the nearest synchronous test component. At this time, the execution system will start the corresponding number of threads according to the number of test components retrieved, and execute these test components in parallel. Only when all of these test components are executed, the synchronous test components and subsequent test components will continue to be executed. If any test component fails to finish, the execution system will wait for it to finish.

2.2. Test script
Scripting language can be surrounded by component technology such as Java or COM [4], so it can effectively use the existing external environment, code and other resources. The spacecraft automatic test framework proposed in this paper already has the function call support to provide various functions for the test script language. Therefore, some basic interfaces of spacecraft test can be called directly by using the script language without re implementing these interfaces. In this paper, BeanShell script language [5] is used as the auxiliary editing language of automatic test cases. BeanShell is a Java scripting language that fully conforms to Java syntax, and has its own syntax and built-in commands. The automatic test platform of spacecraft is developed with Java language. BeanShell is a small, embedded Java source code interpreter that executes standard Java statements and expressions, as well as some script commands and syntax. It can use java reflection API [6] to provide real-time interpretation and execution of Java statements and expressions, and can transparently access any Java object and API. Therefore, it can be said that BeanShell can be used as a script description language tailored for spacecraft automated test cases, and can be seamlessly embedded in the automated test software platform for execution.

In the spacecraft automatic test platform, based on the principle of simple and practical, the command set of BeanShell is expanded, and the special command for spacecraft test is added. The execution logic of these commands is implemented in the spacecraft automatic test platform and can
be called when editing the bean shell test script.

The design of spacecraft test script command library is as follows.

| Function | Purpose | Parameter description | Return value |
|----------|---------|-----------------------|--------------|
| String send(String instNo,String parameter) | send one instruction | instNo: instruction code parameter; can be empty | Success: response information Failure: error message |
| HashMap get(String[] parameterNo) | Get Telemetry parameter values | parameterNo: parameter code array | Success: parameter value hash table Failure: null (null) |
| byte[] get() | Get the whole frame source code | No parameter | Success: whole frame source array Failure: null (null) |
| void setResult(boolean success, String value) | Set the execution result of the current script. Success: is it executed correctly | success: is it executed correctly Candidate value: true/false value: measured value | No return value |

### 3. Design and implementation of component and script hybrid test case

#### 3.1. Automated testing technology framework based on component and script

The framework of automated testing technology based on component and script is shown in Figure 1.
The system is connected to the spacecraft test network. When the test case is finally executed, it controls the front-end hardware test equipment \cite{9} in the form of instructions and receives the test data returned by the equipment. The system is divided into test case editing environment and execution environment.

The use case editing environment includes five modules: test component library, script function library, test case graphical interface, component parameter editing interface and script editing interface. The test component library provides a set of graphical and draggable visual test components, which are divided into business components, process components and self-encapsulated components.

Among them, business component implements basic operation encapsulation for spacecraft test business; the process component realizes the branch and circulation process control; self-encapsulating components are defined by users to further encapsulate a group of business components and / or process components.

Script function library provides blank script, self-encapsulated script and script function. The blank script does not contain any script code, and the self-encapsulated script is a reusable mature script program edited and saved by the user. Script function library provides abundant and common functions for editing test scripts. The graphical interface of test cases is displayed graphically. It supports direct drag and drop from test component library and script function library, and arranges test components and script placeholders. Component parameter editing interface and script editing interface are used to edit test component parameters and write script code respectively. Test cases are stored in the test case library.

The test case execution environment can read the component sequence from the test case library, translate it into script sequence, and send it to the execution engine to drive the execution. According to the use case flow, the execution logic of each component or script is executed separately. Components and scripts share the test data from the device under test. The test case execution environment module is responsible for managing the thread life cycle of test components and scripts, and providing common test process information display interface and test result storage interface for test components and scripts. When the test case is executed, it sends instructions to spacecraft and ground equipment, receives and processes the test data, and stores the test results into the test results database.

3.2. Automatic test workflow based on component and script
The workflow of automated testing technology based on component and script is shown in Figure 2.
Figure 2. Flow chart of automatic test technology based on component and script

1) Start;
2) In the test case editing environment, drag and drop test components and script placeholders to the test case graphical interface;
3) Adjust the position of components and script placeholders in the test case (unnecessary operation) to form the test case framework;
4) Edit test component parameters, write test case scripts, complete the test case parameters customization and logic implementation;
5) Generate test cases and save them to test case library;
6) In the test execution environment, test cases are read from the test case library and parsed into test script sequences. Script sequence is the process description script of the whole test sequence, which does not involve the specific test components or the internal execution logic of the test script;
7) The execution of test case sequence is driven by the test case execution environment module. Each time the test component or script function is executed, a new thread is created for the test component or script function of the current step to execute the internal logic of the test component or script function;
8) According to the internal execution logic of the test component or test script in each step of the
execution process, the test data is read from the test data shared memory, and the global variables shared by the test component and test script are accessed;

9) Test components and scripts call the common interface provided by the test case execution environment module, and publish their own test process information in the monitoring panel of the test case execution environment module.

10) Test components and scripts call the common interface provided by the test case execution environment module, and submit their execution results to the test case execution environment module for display.

11) The test case execution environment module saves the test results to the test results library;

12) end.

4. Test component and script execution

4.1. Test component execution

The key technology principle of test component execution is shown in Figure 3.

1) Test cases are composed by several graphical test components. Different test components represent different test purposes and methods. Each test component consists of three parts: the module name (usually used to describe the purpose of the module), the test business subroutine name corresponding to the module, and some parameters that can be specified by the user. Through the graphical control of programming language, the module is realized.

2) As shown in Figure 3, according to the test requirements of each test component, the corresponding Java class is implemented. This class consists of three parts: several properties, a construction method and an execution method. The attributes correspond to the parameters specified in the test component one by one to customize the business parameters. The construction method is used to create Java class instances and assign values to Java class properties according to the parameters specified in the module.

3) Read the test cases from the database to obtain the test module information of each step.

4) At runtime\[10\], according to the Java class name string of the test business subroutine in the test module, the test case reflection execution engine uses java reflection technology to dynamically load the class object of the Java class of the test business subroutine.

5) Through this operation, the executable subroutine class definition of use case execution level is obtained from the test business module of use case design level.

6) According to the parameters passed in, the Java class instance of the test business subroutine is constructed by reflection. Firstly, the constructor construction method of the class and the parameter array required by the construction method are obtained by reflection. Then, according to the parameter
information in the test module, the parameter data is initialized. Finally, calling the new\(\text{Instance()}\) method and passing the parameter array to create the test business subroutine Java class instance.

7) Through this operation, the test business subroutine object is created, and the business-related parameters in the test business module are passed to the object.

8) Execute the test business method of the instance. In the process of execution, we will use the parameters related to the test business passed in the previous step to make logical judgments related to the test business.

4.2. Execution of test script

The execution mechanism of the test script is shown in Figure 4.

- **Test case script editing and management environment**
  1) BeanShell code editor is the editor of BeanShell script code, and supports automatic prompt keyword, public subroutine name and other auxiliary functions.
  2) Common subroutines are a series of common and reusable methods for spacecraft test requirements development, which are developed by BeanShell. The common subroutine can be called by the test case script. The common subroutine can be maintained and updated at any time. In addition, BeanShell itself has many powerful general functions, such as data calculation function, text processing function, etc.
  3) Based on the built-in syntax verification function of BeanShell, BeanShell syntax verifier can verify the static syntax of spacecraft test case script in real time, and give the error type and location.
  4) The test case tree management module organizes and manages the test cases in a tree structure. The top-down hierarchical structure of the tree structure is: satellite, development stage, electrical test stage, subsystem, test project and test case.
  5) The test plan management sub module manages the test plan tree, including the addition, deletion and modification of plan nodes, and the establishment and deletion of the association between test cases and test plans.

- **Test case script execution environment**
  1) The execution engine automatically loads the associated test case script according to the open test plan task, and uses the BeanShell interpreter to drive the script execution. When the common subroutine is executed, the implementation of the corresponding subroutine is called; when executing the interoperation part with high-level language, call the high-level language interoperation module; during the execution, the execution information is released to the test information release module in real time.
  2) The process management module provides an external intervention interface for users to control the pause, continue and termination of the script execution process.
3) Common subassemblies share the same common subassemblies as the test case script editing and management environment.

4) The running host of the BeanShell script is the test case script execution environment, which is written in Java high-level language. This module is used when the BeanShell script needs to interact with the host.

5) The test information publishing module publishes and displays the real-time information of the script execution process.

6) The test result management module automatically stores the test process information and supports the generation of test reports.

- Data access middleware, test resource management middleware
  Middleware technology [11] is used to shield the diversity of the underlying test database and the variability of the test resource protocol format, and provide a unified and transparent access interface for the upper application.

5. Conclusion
This paper implements the editing of test cases by dragging and dropping test cases and script placeholders. By dragging test components and scripts and arranging them, we can quickly build a test case framework. On the basis of the test case framework, the test components and script functions are further edited to realize the accurate description of the test cases, so that the test cases needed by the spacecraft can be described by the mixed way of test components and script functions. The implementation method is simple, fast and flexible, and can adapt to the complex and changeable test case design of the spacecraft. At the same time, when the test case is executed, the test case composed of test components and script functions is serialized through the test case execution environment, and the serialized test case sequence is executed automatically. In the execution process, the test case and script functions are managed uniformly, and the test case and script functions share global variables.

The automatic test platform described in this paper supports general automatic test for complex spacecraft test requirements. The coverage rate of automatic test is 96.7%, and the test efficiency is improved by more than 50%.

References
[1] Wang Qingcheng, Chen Fengtian, Zhou Geng. (1997) A general test software for spacecraft. Computer measurement and control, 000 (001): 37.
[2] Zhang Jin Xing. (2016) Design and implementation of automatic test software for ground test system of spacecraft electronic equipment. National Space Science Center, Chinese Academy of Sciences.
[3] Sun Bo, Ma Shilong, Yu Dan. (2009) Spacecraft automatic test and spacecraft test language. Journal of Beijing University of Aeronautics and Astronautics, 035 (011): 1375-13781407.
[4] Wen Ben Ying, Tan Shun Tao, Yuan Rong Xiang, et al. (2004) Database design and implementation of SCADA system based on COM technology. Power grid technology, 28 (14): 19-22.
[5] Niemeyer P. (2009) BeanShell: lightweight scripting for Java. http://www.beanshell.org.
[6] Fei Tingwei, Liu Shufen, Qu Zhiyong, et al. (2010) Research on java reflection driven rule engine technology. Computer applications, 30 (005): 1324-1326.
[7] Liu Yuncai. (2000) Telemetry and remote control system (Volume 2). National Defense Industry Press.
[8] Chen Yi en. (2002) Telemetry data processing. National Defense Industry Press.
[9] Li Hongliang, Tan Zheng. (2020) Design of automatic management system for spacecraft front end test equipment. Spacecraft engineering, (5).
[10] Gu Tianxiao. (2017) Research on Java program dynamic update technology. Nanjing University.
[11] Chu Fuying, Wei Wenhua. (2015) Database middleware based on Web services and Hibernate technology. Computer and modernization.