Design and Implementation of Spacecraft Automatic Test Language

YU Dan*, MA Shilong
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China

Received 26 May 2010; revised 17 September 2010; accepted 1 November 2010

Abstract

Spacecraft automatic test system, a comprehensive spacecraft test information system based on the various spacecraft test specifications formalized as spacecraft test language, is an important means to improve test efficiency. With the new requirements of the multi-spacecraft test in China, the study of the spacecraft test language becomes a new challenge for spacecraft test field. In this article, a high-order spacecraft test language, China aerospace test and operation language (CATOL), is given as associated with the current test requirements; meanwhile, the structure of the language is presented. Then, for characterizing and formalizing the spacecraft processes, the syntax and operational semantics of one of the sub-languages, CATOL-PR, are defined. Finally, the prototype system of this proposed language is presented. This language will improve the specification of spacecraft test work in China and the efficiency of spacecraft testers, and promote the development in spacecraft automatic test.

Keywords: spacecraft test language; high-order language; spacecraft; automatic test; operational semantics

1. Introduction

Spacecraft test processes involve the test data in different formats, test equipments in different specifications and tested spacecraft in different types. The implementation of the test is to use the given basic operations to complete the test flows according to a certain method combination in the test environments[1-2]. The ultimate goal of spacecraft automatic test is to transfer the attention of testers from focusing on the process implementation to the test planning, flow monitoring and result analyzing.

Spacecraft automatic test needs a set of standards, which can uniformly describe the test architecture, test data, test equipment, test environment, basic test operation, logic operation and method combination of test flows[3]. With the standards, accessing the test data and test equipment, interacting with the test environment to perform basic test operation, and completing the spacecraft test automatically can be achieved.

Spacecraft test language is the formalization on various test standards. The programs which specify the test requirements, describe the test flows and define test information can be written by the testers with the test language. Running the program on the testing system is to realize test automation which saves the testing time and resources. Therefore, it is essential to carry out the research and development on the definition and implementation of spacecraft test language.

Currently, the test requirement for test mode arrangement is transformed from the single spacecraft test to a multi-spacecraft test in China[4]. It becomes a new challenge of spacecraft test language and needs to address the following issues: how to organize the test task to reduce the complexity of parallel testing, how to manage test data to ensure data integrity and security, how to integrate test equipments to achieve equipment unified access and how to manage the basic test processes to facilitate the unified process calling.

In this article, we study and design a new spacecraft test language which takes problems occurring in the...
multi-spacecraft testing circumstances as the background, investigates the typical spacecraft test languages, and applies the advanced network and computing technology.

The rest of the article is organized as follows. Section 2 introduces the related work. In Section 3, the architecture of the spacecraft system is presented. Section 4 and Section 5 give the structure of a new spacecraft test language—China aerospace test and operation language (CATOL), as well as the syntax and operational semantics of one of its sub-languages. In Section 6, a comparison is given between CATOL and some typical spacecraft test languages. Section 7 introduces the prototype system of CATOL. Section 8 gives the conclusions.

2. Related Work

There is an early start at aerospace engineering in countries with advanced spacecraft technologies, so the spacecraft test language has gone through a long course of development. National Aeronautics and Space Administration (NASA) used ground operations aerospace languages (GOAL)\(^5\). ESA applied the ETOL\(^6\) and TCL/TK\(^7\). In the field of commercial language, the famous Integral Systems (ISYS) company in the United States released satellite test and operations language (STOL)\(^8\) in 1992 to configure and control the components of ground system. In addition, in the standards of test language, IEEE released ATLAS2000 standard\(^9\), which is an upgrade of ATLAS\(^10\). These languages are representative and play an important role in the development history of spacecraft testing language.

Although these representative test languages have their own characteristics, they are not suitable to be introduced to the spacecraft test for China due to the high costs, the local requirements and the new background of current test. Ref.\(^{[11]}\) surveyed and compared these representative languages, summarized their basic characteristics, and proposed the objectives and directions of the development for a test language for China.

Moreover the survey also explains that the design goal of spacecraft test language is to develop specification standard for spacecraft automatic test to meet the domestic demands. On the one hand, such a standard makes the testers describe the test resources and processes with a uniform format which can promote the information sharing, improve the efficiency and ensure safety. On the other hand, testers can focus their attention on analyzing problems and evaluating results appearing in the test processes.

3. Architecture of Spacecraft Automatic Test System

Based on the requirements of spacecraft automatic test\(^{[12-13]}\), the structure of the spacecraft automatic test system is divided into three layers. With the structural and functional optimization of each layer, the collaborative work environment of spacecraft test can be created. The spacecraft test system architecture is illustrated in Fig.1.

The first layer is the test resource layer, which deploys the equipment servers related to the sub-systems. These servers connect the general or special test equipment, receive and execute the commands from

![Fig.1 Architecture of spacecraft automatic test system.](image-url)
the above layer, and manage and handle the basic equipment. Meanwhile these servers receive the test data collected from the special test equipment and command execution information and return these data and information to the above layer. The basic operation for the test equipment is completed in this layer.

The second layer is the test support layer, which deploys the overall control server. The server is responsible for transferring the data and command to test resource layer, storing the test data automatically and monitoring the test processes. All the test application software can send the telecontrol commands, get the telemetry data and manage the test equipment when connecting the overall control server.

The third layer is the test application layer. As Fig.1 shows, the application process is composed of three stages, including the test preparation, test execution and test evaluation. The spacecraft automatic test system takes the test data resources as the core, organizes the test processes for the planning, and completes the test work with the test application software.

4. Overall Structure and Characteristics of CATOL

According to the current requirements of spacecraft test and the architecture of spacecraft test system, we design CATOL, which combines the current advanced network and computing technology, especially service computing technology and resource abstraction technique\(^{[14-15]}\). CATOL is a service-oriented high-level spacecraft test language and its structure is shown in Fig.2.

From Fig.2, we can see that CATOL is a high level language which includes three layers. The center layer, which consists of the spacecraft test equipment unified description language, the spacecraft test data unified description language and the spacecraft test atom unified description language, is responsible for abstracting spacecraft test resources to gain seamless access to resources and improve the reliability and security of the testing process. The middle layer contains the spacecraft test process unified definition language and the spacecraft test strategy modeling language, which makes the test programs in accordance with the standard definition and can reuse existing strategies. The top layer contains the spacecraft test architecture description language, which is responsible for building components of spacecraft test system. Each layer packages its resources as services and provides the services for its upper layer. The following presents the functions of each language.

(1) CATOL-EQ (spacecraft test equipment unified description language). CATOL-EQ shields the heterogeneity of operations to attain transparent access of test equipments. The language allows users to handle and monitor the test equipment remotely without knowing the drive even the type of equipment.

(2) CATOL-DA (spacecraft test data unified description language). The new arrangement mode of multi-spacecraft batch testing applies the distributed network system to the support of remote test. CATOL-DA can give a unified description of distributed mass and heterogeneous data for the spacecraft automatic test to use these heterogeneous data transparently.
(3) CATOL-AT (spacecraft test atom unified description language). There are some common operations in spacecraft test, such as sending commands to determine whether the telemetry parameters are at the required interval, etc. A process test can be completed through a combination of these operations. As these operations are reusable, they can be extracted to form a special kind of resource, testing atoms, which represent the indivisible basic test method in the testing flows. By defining the test atoms, CATOL-AT can guarantee the security and accuracy of the test system effectively and provide a better maintainability and scalability.

(4) CATOL-PR (spacecraft test process unified description language). CATOL-PR can describe the test processes in a unified way, making different testers a consistent understanding for the testing program and maintain the program easier.

(5) CATOL-ST (spacecraft test strategy modeling language). There are many repetitive tasks during the spacecraft testing. CATOL-ST can model the testing strategy so that the common methods for spacecraft testing can be shared and reused. That is seen as an effective way to improve test efficiency.

(6) CATOL-AR (spacecraft test architecture description language). CATOL-AR can adapt to multi-spacecraft batch testing mode and achieve effective management of multi-spacecraft test tasks, resources and environments.

5. Definition of Spacecraft Test Process Language CATOL-PR

In CATOL, the test process unified description language CATOL-PR is used to describe the test flows which face spacecraft testers directly. Testers use this language to write test programs which can be executed automatically by running the test application software.

**Definition 1** BNF definition of CATOL-PR.

```
<multi spacecraft test>::= <multi spacecraft description><multi spacecraft test procedure>
<spacecraft test>::= <spacecraft test description> <spacecraft test procedure body>
<spacecraft test procedure>::= <spacecraft test procedure description><statement>
| <<spacecraft test description> <spacecraft test procedure body>
<spacecraft test procedure body>::= <spacecraft test resource declaration><spacecraft test procedure>
<statement>::= <assignment statement>
| <<spacecraft test procedure description><statement>
| <<spacecraft test procedure> ; <spacecraft test procedure>
| <<spacecraft test procedure> || <spacecraft test procedure>
| if <logic expression> then <spacecraft test procedure> else <spacecraft test procedure>
| while <logic expression> do <spacecraft test procedure> fi
```

By analyzing requirements of spacecraft automatic test, this article summarizes the characteristics which spacecraft automatic test should have and defines the elements of CATOL-PR language based on the researches of a spacecraft automatic test system.

Similar to traditional test language, testers can use the language to compile the test program. But besides the general command and execution control command, spacecraft test process language should provide the functional statement for the spacecraft test, e.g. telemetry statement, and so on.

Here, we define the main components of the language and give the syntax of CATOL-PR. In addition, in order to ensure the accuracy of test procedures, we use the semantic model[16-17] to abstract the practical implementation of CATOL-PR, so designers and users of the language have the same understanding for the language.

5.1. Syntax of CATOL-PR

The components of CATOL-PR can be divided into two categories. The first is the procedural language elements used for the algorithm and behavioral descriptions, which mainly consist of variable assignment statement, sequence statement, condition statement and loop statement, etc. The second is the language elements used for spacecraft test and reflects real-time characteristics, which mainly consist of operational statement, data statement, time statement and control statement.

The following is a subset of CATOL-PR, which embodies the core components of the spacecraft test process and contains the two major language elements described above.

To denote the CATOL-PR, we use BNF[18] which is a formal mathematical way to describe the language.
is a spacecraft test program defined by CATOL-PR and its semantic model firstly.

**Definition 2** Storage status of CATOL-PR. Assume that \( P \) is a spacecraft test program defined by CATOL-PR and \( V = (i_1, i_2, \ldots, i_k; o_1, o_2, \ldots, o_m) \) is the set of variables of \( P \). \((i_1, i_2, \ldots, i_k)\) represents the input parameters and local variables of \( P \). \((o_1, o_2, \ldots, o_m)\) denotes the output parameters of \( P \). So \( \sigma \), the storage state, is a mapping from \( V \) to value set \( M \), expressed as \( \sigma : V \rightarrow M \). \( \sigma (x_i) \rightarrow m_i \) represents that the value of variable \( x_i \) is \( m_i \) under the state of \( \sigma \), where \( x_i \in V, m_i \in M \).

**Definition 3** Tested object status of CATOL-PR. Assume that \( P \) is a spacecraft test program defined by CATOL-PR, \( D \) is the tested object by \( P \), \( A = (situ, r_1, r_2, \ldots, r_j; s_1, s_2, \ldots, s_k) \) is the set of operations of \( D \), where situ is the status of \( D \), \((r_1, r_2, \ldots, r_j)\) represents the input parameters, and \((s_1, s_2, \ldots, s_k)\) denotes the output parameters of each statement.
the output parameters of $D$. $\omega$ is a mapping from $A$ to the value set $\mathcal{W}$, expressed as $\omega: A \rightarrow \mathcal{W}$. $\omega(c_i \rightarrow u_i)$ represents that the value of variable $c_i$ is $u_i$ in the state of $\omega$, where $c_i \in A$, $u_i \in \mathcal{W}$.

**Definition 4** Process status of CATOL-PR procedure. Assume that $S$ is the statement executing in the spacecraft test program which is defined by CA-TOL-PR, $\sigma$ the current storage state of the test program and $\omega$ the status of the testing object. The triple $<S, \sigma, \omega>$ is the process status of CATOL-PR procedure.

**Definition 5** Process status conversion of CATOL-PR procedure. Assume that $<S, \sigma, \omega>$ is the status of CATOL-PR procedure.

1. When $S$ finishes, the new storage status $\sigma'$ and the tested object status $\omega'$ are created. If there is still any statement $S'$ executed under the status of $\sigma'$ and $\omega'$, the new process status is $<S', \sigma', \omega'>$, represented by $<S, \sigma, \omega> \rightarrow <S', \sigma', \omega'>$.

2. When $S$ finishes, the new storage status $\sigma'$ and the tested object status $\omega'$ are created. If there is no any statement executed under the status of $\sigma'$ and $\omega'$, the new process status is $<\Phi, \sigma', \omega'>$, represented by $<S, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>$.

**Definition 6** Process status conversion rule of CATOL-PR procedure. Assume that $C$ and $R$ are process status conversion of CATOL-PR procedure. If $C$ is set up, then $R$ is set up, which is expressed by $C \rightarrow R$.

With the above definitions, semantic model CATOL-PR is built by characterizing the status transition rule for each statement. The following is the operational semantic of CATOL-PR for core statements.

**A** Multi-spacecraft test

\[
<\text{multi spacecraft test}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{multi spacecraft test}_{\text{spacecraft test}}, \sigma, \omega> \rightarrow <\text{spacecraft test}, \sigma, \omega>
\]

\[
<\text{multi spacecraft test}_{\text{test}}, \sigma, \omega> \rightarrow <s, \sigma', \omega'>
\]

\[
<\text{multi spacecraft test}_{\text{testtest}}, \sigma, \omega> \rightarrow <s', \text{multi spacecraft test}, \sigma', \omega'>
\]

\[
<\text{multi spacecraft test}_{\text{testtest}}, \sigma, \omega> \rightarrow <s', \text{multi spacecraft test}, \sigma', \omega'>
\]

\[
<\text{multi spacecraft test}_{\text{testtest}}, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>
\]

\[
<\text{multi spacecraft test}_{\text{testtest}}, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>
\]

\[
<\text{multi spacecraft test}_{\text{testtest}}, \sigma, \omega> \rightarrow <s, \sigma', \omega'>
\]

**B** Spacecraft test procedure

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\text{spacecraft test procedure}, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <s, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <s, \text{spacecraft test procedure}, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <s, \text{spacecraft test procedure}, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <s', \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma', \omega'>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\text{spacecraft test procedure}, \sigma, \omega>
\]

\[
\text{eva}(\text{logic expression}) = \text{true}
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\text{spacecraft test procedure}, \sigma, \omega>
\]

\[
\text{eva}(\text{logic expression}) = \text{false}
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\text{spacecraft test procedure}, \sigma, \omega>
\]

\[
\text{eva}(\text{logic expression}) = \text{true}
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\text{spacecraft test procedure}, \sigma, \omega>
\]

\[
\text{eva}(\text{logic expression}) = \text{false}
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

\[
<\text{spacecraft test procedure}_{\text{test}}, \sigma, \omega> \rightarrow <\Phi, \sigma, \omega>
\]

where $\text{eva}(\cdot)$ represents the value of this statement $s_i$. 

...
(C) Statement

\[
\begin{align*}
&<\text{statement}_1, \sigma, \omega > \rightarrow < \Phi, \sigma, \omega > \\
&<\text{statement}\_\text{assignment} \sigma, \omega > \rightarrow < \text{assignment} \sigma, \omega > \\
&<\text{statement}_1, \sigma, \omega > \rightarrow < s, \sigma', \omega' > \\
&<\text{statement}\_\text{statement}\_\text{statement} s, \sigma, \omega > \rightarrow < s', \sigma', \omega' > \\
&<\text{statement}\_\text{statement}\_\text{statement} s, \sigma, \omega > \rightarrow < \Phi, \sigma', \omega' > \\
&<\text{statement}\_\text{statement}\_\text{statement} \sigma, \omega > \rightarrow < \Phi, \sigma', \omega' > \\
&\text{eval(logic expression)} = \text{true} \\
&<\text{if} \text{logic expression} \text{then} \text{statement}, \text{else} \text{statement}, \sigma, \omega > \rightarrow < \text{statement}_1, \sigma, \omega > \\
&\text{eval(logic expression)} = \text{false} \\
&<\text{if} \text{logic expression} \text{then} \text{else} \text{statement}, \sigma, \omega > \rightarrow < \text{statement}_1, \sigma, \omega > \\
&\text{eval(logic expression)} = \text{true} \\
&<\text{while} \text{logic expression} \text{do} \text{statement}, \sigma, \omega > \rightarrow < \text{statement}, \sigma, \omega > \\
&\text{eval(logic expression)} = \text{false} \\
&<\text{while} \text{logic expression} \text{do} \text{statement} \sigma, \omega > \rightarrow < \Phi, \sigma, \omega > \\
&\text{spacecraft test atom} \sigma, \omega > \rightarrow < \text{spacecraft test atom}, \sigma, \omega > \\
\end{align*}
\]

(D) Assignment statement

\[
< \text{assignment} \text{variable} = \text{expression}, \sigma, \omega > \rightarrow < \Phi, \sigma', [\text{variable} \mapsto \text{expression}], \omega > 
\]

(E) Spacecraft test atom

\[
< \text{spacecraft test atom}, \sigma, \omega > \rightarrow < \text{atom identifier}; \text{atom property}_1; \cdots; \text{atom property}_n, \sigma, \omega > 
\]

(F) Send discrete telecontrol command statement

\[
\begin{align*}
&\text{time(lstc\_response)} \leq \text{response time} \text{ and } \text{opfb(lstc\_response)} = \text{complete} \\
&<\text{send discrete telecontrol command statement}_{\text{bltc}} \sigma, \omega > \rightarrow \\
&<\Phi, \sigma' \text{(state} \mapsto \text{successful)}, \omega' \text{(situ} \mapsto \text{situ'} , \text{ro} \mapsto \text{lstc}'); \text{discrete command id, so} \mapsto \text{lstc\_response}) > \\
&\text{time(lstc\_response)} \leq \text{response time} \text{ and } \text{opfb(lstc\_response)} = \text{fail} \\
&<\text{send discrete telecontrol command statement}_{\text{bltc}} \sigma, \omega > \rightarrow \\
&<\Phi, \sigma' \text{(state} \mapsto \text{fail)}, \omega' \text{(situ} \mapsto \text{situ'} , \text{ro} \mapsto \text{lstc}'); \text{discrete command id, so} \mapsto \text{lstc\_response}) > \\
&\text{time(lstc\_response)} > \text{response time} \\
&<\text{send discrete telecontrol command statement}_{\text{bltc}} \sigma, \omega > \rightarrow \\
&<\Phi, \sigma' \text{(state} \mapsto \text{timeout)}, \omega' \text{(situ} \mapsto \text{situ'} , \text{ro} \mapsto \text{lstc}'); \text{discrete command id, so} \mapsto \text{lstc\_response}) > \\
\end{align*}
\]

The execution of this statement is characterized by the above three process status conversion rules, which explains the status transition when the statement is executing under the storage status \(\sigma\) and the tested object status \(\omega\). time(lstc\_response) is the response time of the command and opfb(lstc\_response) is the return status of the command. 1) means that the tested object accepts the command and the discrete command identifier and returns the lstc\_response message. If the response message is received within the response time and the command is complete, the status of the statement is successful; 2) means that the status of the statement is failure if the command fails; 3) means that if the response message is not received within the response time, the status of statement is timeout.

(G) Send telecontrol command statement with proportion parameters

\[
\begin{align*}
&(\text{time(bltc\_response)} \leq \text{response time}) \text{ and } \text{opfb(bltc\_response)} = \text{complete} \\
&<\text{send telecontrol command statement with proportion parameters}_{\text{bltc}} \sigma, \omega > \rightarrow \\
&<\Phi, \sigma' \text{(state} \mapsto \text{successful)}, \omega' \text{(situ} \mapsto \text{situ'} , \text{ro} \mapsto \text{lstc}'; \text{discrete command id; proportion word; pulse select; number; period; width, so} \mapsto \text{bltc\_response}) > \\
\end{align*}
\]
(II) Send telecontrol command statement with data block

\[\text{time(bltc\_response) \leq response\ time} \quad \text{and} \quad \text{opfb(bltc\_response) = fail}\]

\[< \text{send telecontrol command statement with proportion parameters}_{\text{bltc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{fail)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{bltc'}; \text{proportion command id}; \text{proportion word}; \text{pulse select}; \text{number}; \text{period}; \text{width}, \text{so} \leftrightarrow \text{bltc\_response} > \]

\[\text{time(bltc\_response) \rightarrow response\ time}\]

\[< \text{send telecontrol command statement with proportion parameters}_{\text{bltc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{timeout)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{bltc'}; \text{proportion command id}; \text{proportion word}; \text{pulse select}; \text{number}; \text{period}; \text{width}, \text{so} \leftrightarrow \text{bltc\_response} > \]

\[\text{time(bltc\_response) \rightarrow response\ time}\]

\[< \text{send telecontrol command statement with data block}_{\text{kutc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{successful)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{kutc'}; \text{command id}; \text{command description}; \text{parameters list}; \text{first block id}; \text{command block number, so} \leftrightarrow \text{kutc\_response} > \]

\[\text{time(kutc\_response) \leq response\ time} \quad \text{and} \quad \text{opfb(kutc\_response) = complete}\]

\[< \text{send telecontrol command statement with data block}_{\text{kutc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{timeout)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{kutc'}; \text{command id}; \text{command description}; \text{parameters list}; \text{first block id}; \text{command block number, so} \leftrightarrow \text{kutc\_response} > \]

\[\text{time(kutc\_response) \rightarrow response\ time}\]

\[< \text{send telecontrol command statement with data block}_{\text{kutc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{failure)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{kutc'}; \text{command id}; \text{command description}; \text{parameters list}; \text{first block id}; \text{command block number, so} \leftrightarrow \text{kutc\_response} > \]

\[\text{time(kutc\_response) > response\ time}\]

\[< \text{send telecontrol command statement with data block}_{\text{kutc}, \sigma, \omega} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{timeout)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{kutc'}; \text{command id}; \text{command description}; \text{parameters list}; \text{first block id}; \text{command block number, so} \leftrightarrow \text{kutc\_response} > \]

\[\text{time(kutc\_response) > response\ time}\]

\[< \text{general test equipment set command statement}_{4)} \rightarrow \]

\[\text{(situ(test\ equipment) = ready) and} \quad \text{opfb(setp\_response) = complete}\]

\[< \text{general test equipment set command statement}_{4)} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{successful)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{setp'}; \text{command word}, \text{so} \leftrightarrow \text{setp\_response} > \]

\[\text{(situ(test\ equipment) = busy) or} \quad \text{opfb(setp\_response) = fail}\]

\[< \text{general test equipment set command statement}_{5)} \rightarrow \]

\[< \Phi, \sigma'\text{(state} \rightarrow \text{failure)},  \omega'\text{(situ} \rightarrow \text{situ}', \text{ro} \rightarrow \]

\['\text{setk'}; \text{command word}, \text{so} \leftrightarrow \text{setk\_response} > \]

\[\text{(situ(test\ equipment) = busy) or} \quad \text{opfb(setk\_response) = fail}\]

\[< \text{special test equipment set command statement}_{\text{opfb(setk\_response) = complete}} \rightarrow \]

\[< \text{special test equipment set command statement}_{\text{opfb(setk\_response) = fail}} \rightarrow \]

\[< \text{connect equipment set command statement}_{\text{connect\_response) = complete}} \rightarrow \]

\[< \text{connect equipment set command statement}_{\text{connect\_response) = fail}} \rightarrow \]

The execution of the general test equipment set command statement is characterized by the rules of process status conversion mentioned above, which explains the status change of the statement under storage status $\sigma$ and the tested object status $\omega$. 4) means that the tested object accepts the command setp and the input of command word, and returns the setp\_response message. If the test equipment is idle and the command is finished, the execution status of the statement is successful. 5) means that the execution status of the statement is failure if the test equipment is occupied or the command does not execute normally.
\( (\omega.\text{situ}(\text{test equipment}) = \text{busy}) \) or (\( \text{opfb}(\text{connect} \_\text{response}) = \text{fail} \))

\(< \text{connect} \text{ equipment set command statement} \_), \_\omega >\rightarrow \< \Phi, \sigma' \rangle \text{state} = \text{fail}, \omega'(\text{situ}_\text{test equipment} \mapsto \text{situ}'_\text{test equipment}, \text{ro} \mapsto '\text{connect}', \text{so} \mapsto \text{connect} \_\text{response}) >

(L) Disconnect equipment set command statement
\( (\omega.\text{situ}(\text{test equipment}) = \text{busy}) \) and (\( \text{opfb}(\text{disconnect} \_\text{response}) = \text{complete} \))

\(< \text{disconnect} \text{ equipment set command statement} \_), \_\omega >\rightarrow \< \Phi, \sigma' \rangle \text{state} = \text{successful}, \omega'(\text{situ}_\text{test equipment} \mapsto \text{ready}, \text{ro} \mapsto '\text{disconnect}', \text{so} \mapsto \text{disconnect} \_\text{response}) >

(M) Parameter statement
\( \text{opfb}(\text{get} \_\text{response}) = \text{complete} \)

\(< \text{parameter} \text{ statement} \_), \_\omega >\rightarrow \< \text{assignment} \text{ statement} \)
\( \omega'(\text{situ} \mapsto \text{situ}', \text{ro} \mapsto ' \text{getvalue} / \text{getrawvalue} / \text{getframevalue}'; \text{parameter} \text{ identifier}, \text{so} \mapsto \text{get} \_\text{response}) \)

\( \text{opfb}(\text{get} \_\text{response}) = \text{fail} \)

\(< \Phi, \sigma' \rangle \text{state} = \text{fail}, \omega'(\text{situ} \mapsto \text{situ}', \text{ro} \mapsto ' \text{getvalue} / \text{getrawvalue} / \text{getframevalue}'; \text{parameter} \text{ identifier}, \text{so} \mapsto \text{get} \_\text{response}) \)

The execution of the parameter statement is characterized by the rules of process status conversion mentioned above, which explains the status change of the statement under storage status \( \omega \) and the tested object status \( \sigma \). 6) means that the tested object accepts the getvalue, getrawvalue or getframevalue command and input of parameter identifier and returns get_response message. If the command is finished, the status of the statement is successful and a variable is assigned with the return value. 7) means that the status of the statement is failure when the command fails.

(N) Verify statement
\( (\text{lower limit} \leq \text{eva}(\text{verify} \_\text{response}.\text{value}) \leq \text{upper limit}) \) and (\( \text{opfb}(\text{verify} \_\text{response}) = \text{complete} \))

\(< \text{verify} \text{ statement} \_), \_\omega >\rightarrow \< \text{assignment} \text{ statement} \)
\( \omega'(\text{situ} \mapsto \text{situ}', \text{ro} \mapsto ' \text{verify}'; \text{parameter} \text{ identifier}, \text{so} \mapsto \text{verify} \_\text{response}) >

\langle (\text{eva}(\text{verify} \_\text{response}.\text{value}) < \text{lower limit}) \text{ or } (\text{eva}(\text{verify} \_\text{response}.\text{value}) > \text{upper limit}) \rangle \)

\( \langle \text{opfb}(\text{verify} \_\text{response}) = \text{complete} \rangle \)

9)
\(< \text{verify} \text{ statement} \_), \_\omega >\rightarrow \< \text{assignment} \text{ statement} \)
\( \omega'(\text{situ} \mapsto \text{situ}', \text{ro} \mapsto ' \text{verify}'; \text{parameter} \text{ identifier}, \text{so} \mapsto \text{verify} \_\text{response}) >

\( \text{opfb}(\text{verify} \_\text{response}) = \text{fail} \)

10)
\(< \text{verify} \text{ statement} \_), \_\omega >\rightarrow \< \Phi, \sigma' \rangle \text{state} = \text{fail}, \omega'(\text{situ} \mapsto \text{situ}', \text{ro} \mapsto ' \text{verify}'; \text{parameter} \text{ identifier}, \text{so} \mapsto \text{verify} \_\text{response}) >

The execution of the verify statement is characterized by the rules of process status conversion mentioned above, which explains the status change of the statement under storage status \( \omega \) and the tested object status \( \sigma \). 8) means that the tested object accepts verify command and input of parameter identifier and returns verify_response message. If the command is finished and the value of the return parameter is between the upper limit and the lower limit, the status of the statement is successful and the boolean variable is assigned true. 9) means that if the command is finished but the value of the return parameter is not between the upper limit and lower limit, the status of the statement is successful and the boolean variable is assigned false. 10) means that the status of the statement is failure when the command fails.

(O) General test equipment data statement
\( (\omega.\text{situ}(\text{test equipment}) = \text{ready}) \) and (\( \text{opfb}(\text{get} \_\text{p} \_\text{response}) = \text{complete} \))

\(< \text{general} \text{ test} \text{ equipment} \text{ data} \text{ statement} \)_
\( \langle \text{general} \text{ test} \text{ equipment} \text{ data} \text{ statement} \)_
\( \text{get} \_\text{p}, \_\omega >\rightarrow \langle \text{assignment} \text{ statement} \)
\( \omega'(\text{situ}_\text{test equipment} \mapsto \text{situ}'_\text{test equipment}, \text{ro} \mapsto ' \text{get} \_\text{p}', \text{so} \mapsto \text{get} \_\text{p} \_\text{response}) >

The execution of the general test equipment data statement is characterized by the rules of process status conversion mentioned above, which explains the status change of the statement under storage status \( \omega \) and the tested object status \( \sigma \). 8) means that the tested object accepts getp command and input of test equipment transfer server identifier, test equipment identifier and returns getp_response message. If the command is finished, the status of the statement is successful and a variable is assigned with the return value.
(\omega, \text{situ}(\text{test equipment}) = \text{busy}) \text{ or } (\text{opfb}(\text{getp}_\text{response}) = \text{fail}) \\
< \text{general test equipment data statement} \quad \text{getp}_\text{response} \quad \sigma, \omega 

< \Phi, \sigma'(\text{state} = \text{fail}), \omega'(\text{situ}_1 \text{test equipment transfer server identifier, test equipment identifier} = \text{busy}) \text{ or } (\text{opfb}(\text{getp}_\text{response}) = \text{fail}) \\
\text{situ}'_\text{test equipment transfer server identifier, test equipment identifier} \quad \text{ro} \mapsto \text{test equipment identifier} \quad \text{so} \mapsto \text{getp}_\text{response} > 

(P) Special test equipment data statement 
\text{opfb}(\text{getk}_\text{response}) = \text{complete} \\
< \text{special test equipment data statement} \quad \text{getk}_\text{response} \quad \sigma, \omega 

< \text{assignment statement} \quad \text{data identifier} = \text{eva}(\text{getk}_\text{response}_\text{value}), \sigma'(\text{state} = \text{successful}), \omega'(\text{situ}_2 \mapsto \text{situ}', \text{ro} \mapsto \text{getk}, \text{so} \mapsto \text{getk}_\text{response} > \\
\text{opfb}(\text{getk}_\text{response}) = \text{fail} \\
< \Phi, \sigma'(\text{state} = \text{fail}), \omega'(\text{situ}_2 \mapsto \text{situ}', \text{ro} \mapsto \text{getk}, \text{so} \mapsto \text{getk}_\text{response} > 

(Q) Time wait statement 
\text{time} = \text{clock} - \text{time}(s_\text{p}) = \text{wait time value} \\
< \text{time wait statement} \quad \text{wait} \quad \sigma, \omega 

< \text{time wait statement} \quad \text{time} = \text{clock} - \text{time}(s_\text{p}) < \text{wait time value} 

where \text{s}_\text{p} \text{ is the previous statement of time wait statement.}

(R) Time condition statement 
(time(clock) = \text{absolute time value}) \text{ or } (time(clock) - time(s_\text{p}) = \text{relative time value}) \text{ or } (\text{eva}(\text{condition expression}) = \text{true}) \\
< \text{time condition statement} \quad \text{until} \quad \sigma, \omega 

(time(clock) \neq \text{absolute time value}) \text{ and } (time(clock) - time(s_\text{p}) \neq \text{relative time value}) \text{ and } (\text{eva}(\text{condition expression}) = \text{false}) \\
< \text{time condition statement} \quad \text{until} \quad \sigma, \omega 

(S) Control statement 
< \text{control statement} \quad \text{start statement} \quad \sigma, \omega 

< \text{control statement} \quad \text{halt statement} \quad \sigma, \omega 

< \text{control statement} \quad \text{end statement} \quad \sigma, \omega 

< \text{control statement} \quad \text{continue statement} \quad \sigma, \omega 

< \text{control statement} \quad \text{call statement} \quad \sigma, \omega 

< \text{start statement} \quad \sigma, \omega 

< \text{mid statement} \quad \sigma, \omega 

< \text{end statement} \quad \sigma, \omega 

< \Phi, \sigma'(\text{time}(v_\text{all}) = v_\text{all}), \omega > \\
(v_\text{all} \text{ represents all the variables in the procedure}) 

< \text{call statement} \quad \sigma, \omega 

\text{The above is the semantic of spacecraft test language based on the model of operational semantic, and the proof about the completeness and correctness can be referred to Ref. [20].}

6. Comparison

The current spacecraft test languages are basically aimed at the spacecraft test processes, while the CATAL is a spacecraft test language system, including CATOL-PR, the spacecraft testing procedure language. As far as the comparability is concerned, the following two examples are used to compare CATOL-PR with GOAL and ETOL, the other two spacecraft testing languages, and analyze the differences among them.

Case 1 For a stable data value in the network, and the test host is idle and can accept the command, send a command and verify a parameter.

The implementation of the case by three languages is shown in Table 1.

<table>
<thead>
<tr>
<th>Language</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATOL</td>
<td>setk TC:NET:on setk TC:power.on</td>
</tr>
<tr>
<td>GOAL</td>
<td>Step1 open TC; Step2 turn on power; Step3 verify &lt;P008&gt; is between 1 PSIG and 1 PSIG; Step4 close TC; Step5 terminate;</td>
</tr>
<tr>
<td>ETOL</td>
<td>Configure the MTGP, add the parameter P008, and settle the upper and lower limits [1.1] TC on TC Z4 Verify P008 TC off</td>
</tr>
</tbody>
</table>
Compared with the two languages, CATOL provides the parallel way of test program, test procedure and test statement, so it is easy for CATOL to perform multiple test programs.

Case 2  Wait until a parameter satisfies a constraint (e.g. the upper limit is 10 and lower limit is 1), and do next.

The implementation of this case by three languages is shown in Table 2.

<table>
<thead>
<tr>
<th>Language</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATOL</td>
<td>Verify the static parameter atom: P008 1, 10</td>
</tr>
<tr>
<td>GOAL</td>
<td>Step1 verify &lt;P008&gt; is between 1 PSIG and 10 PSIG else go to Step1; Step2 terminate.</td>
</tr>
<tr>
<td>ETOL</td>
<td>Configure MTGP, add the parameter P008, and settle the upper and lower limits [1,10] Integer N=P008; while(N&lt;1</td>
</tr>
</tbody>
</table>

According to the analysis of the test procedure, CATOL abstracts the basic test method in test procedure, namely spacecraft test atom. Calling these testing atoms can help to enhance the reusability of the testing procedure, and make it easy to write a test program. At the same time, it provides us a good maintainability and scalability.

Therefore, compared with other spacecraft test languages, CATOL presents a systematic organization and has a better function abstraction and feasibility.

7. Implementation of CATOL Prototype System

Based on the syntax of CATOL-PR, testers can write the spacecraft automatic test program. The following is a test program instance which is defined by CATOL-PR. The spacecraft test executing system, through understanding the operational semantics, can parse and run the instance. In this way, the test program can be performed automatically.

MSProcess {
  Sprocess {
    SProcess1 set the equipment status {
      double value = getvalue sp003;
      boolean bool = verify <sp003> <10> <20>;
      SubsProcess1 {
        Lstc 0032 60;
        monitor PK [PK9, PK] ;
        int i = 0;
        While ((i + i)<10) do {
          setk ts00+i tc:001;
          double value = getvalue TC03;
        }fi
        monitor PK time difference [PK17, time difference 60 s];
      }
    }
  }
}

We have already developed the prototype system of CATOL. It has a friendly graphical user interface for editing and implementation. Fig.3 shows the implementation interface of the system. Via monitoring the status of the procedure, the tester can know the processes of testing clearly, and thus can deal with the test results in time. The figure shows the test procedure SProcess1 which is under implementation. When running the loop, the executing engine will calculate according to the conditional expression to determine what will be done next. This loop will implement two test atomic processes in ten cycles, and then get out of circulation to implement the last test atomic process "monitor PK time difference" in the routine testing. When finishing the processes above, the user can implement the atomic process "fill in the conclusion". After that, the execution of this test program is completed.

Fig.3 Prototype system of CATOL.

8. Conclusions

(1) According to the analysis of existing typical spacecraft test languages, combining with the current advanced network and computing technology, especially resource abstraction technique and service computing technology, we design a high-level spacecraft test language CATOL based on resource abstraction technique. The syntax and semantic model of the sub-process definition language CATOL-PR is defined in detail.

(2) The language has been used in the practical test and enhanced the efficiency of spacecraft test. The performance of the language running in some spacecraft automatic test system is good.

References

[1] Yamada T. Generic software for spacecraft testing and operations based on a spacecraft model. Proceedings of
SpaceOps 2006; 2006.


Biographies:

YU Dan: Born in 1979, she received her Ph.D. degree from Wuhan University in 2007 and has been doing postdoctoral research in Beihang University since then. Her main research interest is spacecraft test language and safety-critical system. E-mail: yudan@buaa.edu.cn

MA Shilong: Born in 1953, he received his M.S. degree from Beihang University in 1984 and is now the professor of School of Computer Science and Engineering in Beihang University. His main research interest is spacecraft application system, network computing and so on. E-mail: slma@nlsde.buaa.edu.cn