# Research on Static Comprehensive Detection Method for Mission Electronic System

Hang Lian
School of Computing
Beihang University
Beijing, China
e-mail: lianhang1008@gmail.com

Jianghua Lv
School of Computing
Beihang University
Beijing, China
e-mail: jhlv@nlsde.buaa.edu.cn

Shilong Ma
School of Computing
Beihang University
Beijing, China
e-mail: slma@nlsde.buaa.edu.cn

***Abstract*—In the weapon system, with the improvement of the equipment performance, the system complexity also increases. Functional failure will cause the system to malfunction or even become a disaster. Therefore, system-level comprehensive detection for Mission Electronic System is the urgent issue in the field of weaponry and equipment. First of all, this paper gives the definition of both static and dynamic problems, which are caused by the upgrade of the complexity. Secondly, we establish a formal model for static comprehensive detection, which include definitions of Mission Electronic System, system environment, consistency and adaption. Meanwhile, generic detection algorithm of consistency and suitability are put forward. Finally, we achieve the relevant system based in this model.**

***Keywords- Mission Electronic System; static comprehensive setection; formal; consistency detection***

## I. INTRODUCTION

With the improvement of avionics system in the degree of integration and the widely use of micro devices, fault diagnosis and self-testing are more important for the military aircraft's suitability. Mission Electronic System usually consists of multiple fire control systems and it is more complex compared with the same level of fire control system. Thus leads to the cost of maintenance occupies more proportion in in the entire life cycle. Relevant statistics show that, about 33% of faults can be detected, about 66% could not be detected in the factory acceptance test, and only about 5% of perennial faults are described in technical manual. A survey which is carried out in aviation division shows that if we conduct a routine inspection every 3 days for 8 airplanes, the number in Maintenance Group should be more than 70, and they often work overtime. Therefore, the auxiliary field testing for the electronic system must be taken into consideration as a part of the system design.

The interface of Mission Electronic System is complicated, which contains many interface element, so improper operations can also affect the accuracy of the system's behavior. The reason of failures can be attributed to the surrounding correctness and function usability of a kind of large-scale distributed software system. The correctness of the surrounding concludes the correctness of installation files and executable files of system modules, the adaption between modules and modules, and adaption between modules and environment. It can be down to system consistency and system adaption, and also be referred to as static problems. The usability of the function contains availability of the software module, usability of the subsystem and availability of overall system, which are also referred to as dynamic problems. The two types of problems are common and can't be avoided in Mission Electronic System, which are related to the availability of overall system. It cost a lot of manpower and resources to testing and maintenance. This article focuses on the comprehensive detection of static problems, in order to improve the availability of Mission Electronic System.

## II. MISSION ELECTRONIC SYSTEM DETECTION

At present, there are three main means of detection for electronic system, Automatic Test System (ATS) which is also called Automatic Test Equipment (ATE), Built-in Test (BIT) technology and Prognostics and Health Management (PHM) technology. ATS (or ATE) is to integrate all the necessary incentives into test equipment, complete all the controls of motivation and the response signals through the computer, and execute state monitoring, performance testing and troubleshooting automatically for the unit under test (UUT), U.S. Army's Integrated Test Equipment (IFTE) and U.S. Navy's Consolidated Automated Support System (CASS) are the most successful cases all over the world[1]. BIT technology is an automated testing capabilities which is afforded by internal testing and fault isolation. It integrates the automatic detection into the system under test (SUT). But to some extent, it will occupy the electronic system resources. PHM is the technology that improves on BIT and status monitoring. This development is the change from status monitor to the health management, which introduces a predictive capability. With this ability, we can identify and manage the occurrence of failure, plan maintenance and supply of security [2].

Currently, many researchers have proposed their models or systems for the detection of the electronic system. Dr. Zhang Hongyan, in Northwestern Polytechnical University, achieved a common avionics automatic detection system, which was based on signal oriented language ATLAS and detected all LRUs of the entire avionics system [3]. Dr. Zhang Zaide, a professor in The First Aeronautical Institute of Air Force, also completed an automatic testing system by encouraging feedback on the hardware unit. These two systems can be seen as testing systems which focus on electronic signals' test (hardware side) [4]. Dr. Willard A. Hansen also proposed a performance tracking methods and

decision support model. However, these models or systems are concentrated in the test process [5]. This paper will describe the model of static comprehensive detection from the perspective of formal, and implement a system based on the formal model, which is supposed to support the automation of static detection.

## III. Formal Model Of Static Comprehensive Detection For Mission Electronic System

### A. Mission Electronic System

Mission Electronic System consists of several subsystems. What static comprehensive detection wants to check mainly include the consistency of the software configuration file and the adaption of interfaces between system environment and Mission Electronic System. Therefore, from the view of the static structure, each subsystem is composed of configuration files, interfaces and corresponding indicators. The configuration file refers to the profiles which are installed in the corresponding subsystem. The interface means the API of the system resource. And the indicator are the requirement of system performance.

***Definition 1.*** The configuration file is defined as a set of triples, $f = \{(name, a_i, v_j) | a_i \in A, v_j \in V\}$, where

- name is the identification of the configuration;
- $A = \{a_1,a_2,\ldots,a_t\}$, a set of file attributes;
- $V=\{v_1,v_2,\ldots,v_u\}$, a set of attribute values;
- $v_i = attr(a_i)$.

For example, a file name "test.text", which is in the route "/home/test/test/", can be described as {(test, type, txt), (test, location, /home/test/test/text.txt), (test, creation time, 2013/7/30 10:02:50),…}.

***Definition 2.*** The indicator is a triple (r-Name, P, B), where

- r-Name is the identification of the resource;
- $P=\{p_1,p_2, p_3,\ldots,p_n\}$, a set of resource attributes;
- $B = \{b_1,b_2,b_3,\ldots,b_m\}$, a set of boundary values;
- bi=[lower, upper],lower means lower bound,upper means upper bound.

For example, the requirement of a subsystem software contains that the microprocessor frequency is more than 600MHz, the memory, the memory is not less than 512MB, storage capacity is more than 512MB, and the reading and writing speed is greater than 4.5MB / s. So this indicator is (CPU,{GHz},{[600, ∞]}), (Memory ,{capacity},{[512MB, ∞]}),(Disk,{capacity, R&W Speed},{[512MB, ∞], [4.5MB/s, ∞]}).

***Definition 3.*** The API is defined as a quintuple (API-Name, $I_{in}$, $I_{out}$ ,$B_{in}$, $B_{out}$)[6], where

- API-Name is the name of the API;
- $I_{in} = \{P_1,P_2,\ldots,P_m\}(m \in N)$, represents the input interface, Pi represents a parameter $I_{in}$;
- $I_{out} = \{P_1,P_2,\ldots,P_n\}(n \in N)$represents the output interface, $P_i$ represents $I_{out}$ parameter;
- $B_{in}$ represents a constraint set of $I_{in}$, $B_{out}$ represents a constraint set of $I_{out}$.

Remark: Use param($I_{in}$) to represent the parameter of $I_{in}$, Similary, param($I_{out}$) represents the parameter of $I_{out}$.

For example, DWORD WNetAddConnection2 (_In_ LPNETRESOURCE lpNetResource, _In_ LPCTSTR pPassword, _In_ LPCTSTR lpUsername, _In_ DWORD dwFlags), which is a creation interface of the network resource, can be described as (WNetAddConnection2, { lpNetResource, pPassword, lpUsername, dwFlags }, {return}, { LPNETRESOURCE, LPCTSTR, LPCTSTR, dwFlags }, {DWORD}).

***Definition 4.*** A subsystem, marked as SS, is composed of some configuration files, some outside API and corresponding indicators. SS is defined as a triple (SS-F,API,PI), where

- $SS\text{-}F=\{f_1,f_2,\ldots,f_m\}$ represents a set of system configuration files;
- $API=\{api_1,api_2,api_3,\ldots,api_n\}(n \in N)$represents a set of API, $api_i$ represents the ith interface of SS;
- $PI=\{pi_1,pi_2,\ldots,pi_s\}$ represents a set of indicators, $pi_i$ represents the ith indicator.

***Definition 5.*** Mission Electronic System, which is marked as ES, consists of several subsystems, so it can be expressed as a set of subsystems, i.e. $ES = \{SS_1, SS_2,\ldots,SS_l\}$.

### B. System Environment

***Definition 6.*** The resource R is defined as a tetrad (r-Name, P, V, F), where

- r-Name is the identification of resource R.
- $P=\{p_1,p_2, p_3,\ldots,p_n\}$ is a set of resource properties.
- $V = \{v_1,v_2,v_3,\ldots,v_m\}$ is a set of attribute values.
- $F=\{api_1,api_2,\ldots,api_s\}$ is a set of external APIs of appropriate resources.

For example, the memory resource can be expressed as (Memory Resource , {Total, Available, Remaining, Used}, {4.0G, 3.9G, 22%, 78%}, {(new, {size}, {return}, {size_t}, {{void*,NULL}}), (malloc, {size}, {return}, {size_t}, {{void*,NULL}})}, (relloc, {mem_address, newsize}, {return},{void*,unsigned int},{{void*, NULL}}), (free, …), … ).

***Definition 7.*** The system environment, which is composed of several resources, is defined as a set OS, i.e. $OS = \{r_1, r_2, r_3,\ldots,r_n\}, r_i \in R$.

***Definition 8.*** The status of system environment is defined as a set of tetrads $s = \{(r, p_i, v_k) | p_i \in P, v_k \in V\}$, where

- r is the resource name.
- $P = \{p_1, p_2, ..., p_t\}$ is the set of resource attributes.
- $V = \{v_1, v_2, ..., v_u\}$ is the set of attribute values

Remark: The system environment is usually the operating system environment. From the view of resource, operating system include two types of resources—hardware and software resources. State of system environment means the operating system version, CPU, memory, disk, the database state and the state of the network interface.

### C. Static Detection Behaviors

What the static comprehensive detection concentrates are system environment and configuration file. We collect data which can represent the state of current system, analyse these data and give out the result.

***Definition 9.*** For configuration files f1∈SS-F and f2∈SS-F, f1= {(name1, $a1_i$, $v1_j$)| $a1_i$∈A1, $v1_j$∈V1} and f2= {(name2, $a2_i$, $v2_j$)| $a2_i$∈A2, $v2_j$∈V2}, if they satisfy the following conditions:

- name1= name2
- A1=A2
- ∀$a1_i$∈A1, ∀$a2_i$∈A2, $v1_j$= $v2_j$ if $a1_i$ =$a2_i$

then f1 and f2 are consistent expressed as f1 == f2; otherwise inconsistent, expressed as f1! = f2.

***Definition 10.*** For a resource r=(r-Name, P, V, F) and a corresponding indicator pi = (r-Name, P, B), if r satisfies the following conditions:

- vi >= $b_i$.lower
- vi <= $b_i$.upper

then resource r satisfies indicator pi, expressed as r◎pi.

***Definition 11.*** For APIs api1 and api2, where api1 = (API-Name1, $I1_{in}$, $I1_{out}$ , $B1_{in}$, $B1_{out}$), api2 = (API-Name2, $I2_{in}$, $I2_{out}$, $B2_{in}$, $B2_{out}$), if they satisfy the following conditions:

- API-Name1 = API-Name2
- param($I2_{in}$) = param($I1_{out}$)
- $B1_{out}$ ⊆ $B2_{in}$
- param($I1_{in}$) = param($I2_{out}$)
- $B2_{out}$ ⊆ $B1_{in}$

then api1 and api2 is adapted, expressed as I1 ∆ I2.

For example, For API api1=(malloc, {size}, {return}, {[0,2G]}, {{void*,NULL}}) and API api2= (malloc, {return}, {size}, {{void*,NULL}}, {[0,512M]}), they satisfy all the five rules of adaption, so api1∆api2. If api2= (malloc, {return} , {size}, {{void*,NULL}}, {[0,2.5G]}), then they don't satisfy the 5th rule and they are not adapted.

### *D. Detection Process*

*1) Consistency Detection.* Detection is a process which checks an indicator whether it satisfies a given standard, and some data are to be pre-processed before the detection process. In the static detection process, the purpose is to ensure that current software configuration items are consistent with expected. Therefore, we need a standard library, expressed as SL, a library which pre-stores some normal data for comparison. A complete and secure set of software configuration item files are stored in SL. For each subsystem $SS_i$, its consistency depends on the corresponding difference between the standard library SL and the the actual system configuration files exist in the collection, which is expressed as CRT. Thus, we get the process of consistency detection:

**Algorithm 1.** The process of consistency detection.

Input：ES={$SS_1$, $SS_2$, …, $SS_n$}, SL, CRT
Output：CRT = {$CRT_1$, $CRT_2$,…,$CRT_n$}
for(i=1; i<=|ES|; i++)
{
for(j=1;j<|SS-F(i)|;j++)
{
if(f(j) is not match in SL(i))
{
put f(j) into CRT+(i);
}
}
for(j=1; j<|SL(i)|; j++)
{
if(sl_f(j) is not match in SL(i))
{
put f(j) into CRT-(i);
}
}
put CRT+(i) into CRT(i);
put CRT-(i) into CRT(i);
put CRT(i) into CRT;
}

*2) Adaption Detection.* Adaption detection mainly consists of two parts—the status adaption and API adaption. Status adaption detection is to ensure that the current state of the system environment OS meets the indicator of Mission Electronic System. The detection result for the resource state can be expressed as SDRT = {A, NA}, where A (Adaptive) indicates an adapter and NA (Non-adaptative) said they did not fit. So the result of state adaption detection for a resource r = (r-Name, P , V, F) can be expressed as SRT = {(r, $p_i$, $sdrt_k$)|$p_i$∈P, $sdrt_k$∈SDRT }. Here we get the state adaption detection process:

**Algorithm 2.**The process of state adaption detection.

Input：SS = (SS-F, Port, PI), s = {(r, pi, vk) |pi∈P, $v_k$ ∈V}
Output：SRT = {(r, $p_i$,$sdrt_k$)|$p_i$∈P, $sdrt_k$∈SDRT }
for(i=1; i<|PI|; i++)
{
for(k=1;k<|pi(i).P|;k++)
{
get s(j) from s by (r-Name, pi(i).p(k));
if(s(j).attr(p(k)) < b(k).lowwer || s(j).attr(p(k)) > b(k).upper)
{
put (r-Name,p(k), NA) into SRT;
}
else
{
put (r-Name,p(k),A) into SRT;
}
}
}

The API adaption means that the resource interface of system environment can satisfies requirements from Mission Electronic System. For example, DWORD WNetAddConnection2 (_In_ LPNETRESOURCE lpNetResource, _In_ LPCTSTR pPassword, _In_ LPCTSTR lpUsername, _In_ DWORD dwFlags) is a network connection function. From the view of OS, lpNetResource, pPassword, lpUsername and dwFlags are all input parameters, but from the view of Mission Electronic System, on the contrary, these are its output parameters, and the handle returned by the function is its input parameter. If and only if the API provided by OS and the API called by ES

are adapt, the system is useful. In the process of API adaption detection, first, put APIs required by Mission Electronic System into the library, and then get the APIs provided by OS, compare them and then get the result, which we call APIRT. So we get the process of API consistency detection:

**Algorithm 3.** The process of API consistency detection.

Input：SS = (SS-F, API, PI), OS = $\{r_1, r_2, r_3,\ldots,r_n\}$

Output：APIRT

```
for (i=1; i<|SS.API|; i++)
{
get api(j) from OS by API-Name;
if(param(api(i).Iin)= param(api(j).Iout) && api(j).Bout.contain(api(i).Bin) && param(api(i).Iout)= param(api(j).Iin) && api(i).Bout.contain(api(j).Bin))
{
put (API-Name, A) into APIRT;
}
else
{
put (API-Name, NA) into APIRT;
}
}
```

## IV. System-level Static Comprehensive Detection For Mission Electronic System

System-level static comprehensive detection for mission electronic system is achieved based on the former formal model. It mainly completes the detection and analysis of predetermined information extracted from electronic system software configuration items, operating systems, applications and databases. The purpose is to find the problem caused by internal and external reasons, develop the efficiency of detection, help the fault detection and protect the stability and reliability of detection.

The software consists of a static detection component and an analysis component, and link to the mission electronic system through local area network. From the view of function, this system includes a module of user login and authentication, a module of the system environment status detection, detection of the API information, collection of system configuration items.

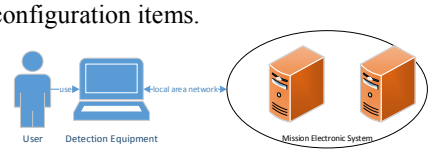


Figure 1. The pattern of usage

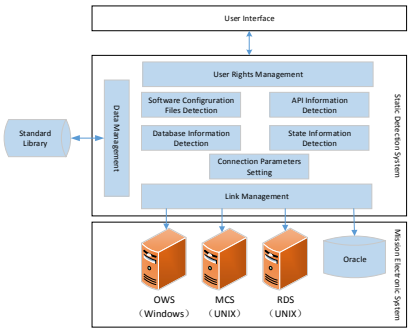


Figure 2. Function modules

## V. Conclusion

In this paper, we describe the complexity of mission electronic system and problems caused by that. Meanwhile, a formal model of static comprehensive detection are established, which provides a formal description to mission electronic system, the system environment and the detection of consistency and adaption. Base on this model , a system is achieved. This model can pave a foundation for the next study—run-time dynamic detection.

## REFERENCES


[1] Jia Zhi-jun, Yan Guo-qiang. Development Trend of Military ATE/ATS Technology Overseas[J]. Computer Measurement & Control, 2003, 11(1):1-4.

[2] Zeng Sheng-kui, Michael G. Pecht, Wu Ji. Status and Perspective of Prognostics and Health Management Technologies[J].ACTA AERONAUTICA ET ASTRONAUTICA SINICA, 2005, 26(5): 626-632.

[3] Zhang Ho ng yan, Li Yanjun, Zhang Ke. Design and Realization of Avionics General Automatic Test System[J]. Computer Measurement & Control, 2009,17(2):255-257.

[4] ZHANG Zai-de, HUANG Jia-cheng. The integrated intelligent test system of avionic device[J]. Journal of Xinyang Agricultural College, 2009, 19(2).

[5] W.A. Hansen, K.T. Fitzgibbon, N.S. Flann, L.V. Kirkland. A performance tracking methodology and decision support model[C]. IEEE AUTOTESTCON Proceedings. 2000.

[6] LI Xin-ke, YAN Lu-ming. Interface matrix—based detecting method for change of component[J]. Application Research of Computers, 2009, 26(4):1360-1362.