Design and Research on Real-Time Interface Management Framework of the General Embedded Testing System for Model Driven Testing

Xiaoxu Diao and Bin Liu

Beihang University,
38 Xueyuanlu Road, Beijing, China
dxx@dse.buaa.edu.cn, liubin@buaa.edu.cn

Abstract. This paper designed and realized an efficient, reliable and extendable interface management framework to fit new characteristics of general embedded testing system such as diversity, complexity and timeliness. This framework was designed to apply the principle of test in model driven architecture (MDA) to simplify the process and satisfy the requirement of their simulation. The article also provided the details of the structure of our framework and the relationship between the modules in it. At last, we applied it in real time operating system to test and verify our theory.

Keywords: interface management, MDA, testing system.

1 Introduction

The general real-time embedded testing system is an effective implement for the test of complicate products and devices. With the development of science and technology, great deals of new embedded products come out, which possess the characteristics such as time critical, complexity and multi-tasks. Therefore, the new testing system should be reliable and safety, which can acquire testing data efficiently and precisely and can be customized to test all kinds of devices. Meanwhile, the researches of Model Driven Testing (MDT) are put forward to abstract the test itself from variety hardware devices. This principle let the test developing staff to concentrate on the test case instead of the device driver or the task scheduling.

First, this paper described the theory and principle of MDT and summarized the requirements of the new testing system especially in the interface management framework. Second, the article describes the theory of MDT and makes a summary of the testing system for MDT. After that, the part 3 interpreted the function and architecture of the interface management framework in details. Moreover, the configuration and workflow of our framework is introduced respectively in part 4. At last, we applied the framework and conclude our theory in part 5.

Y. Wu (Ed.): ICHCC 2011, CCIS 163, pp. 359–365, 2011.

2 Model Driven Testing

Model Driven Testing (MDT) is a new kind of testing method based on the fundamental principles of Model Driven Architecture (MDA). MDT is devoted to solving the problem of reuse of artifacts produced during the stage of software development to the process of software testing. This makes testing start earlier and the errors produced in the process of software development could be detected and eliminated in time.

In the MDT, the principles are summarized as follows. Firstly, system test case is generated from the software requirement model by test case generation tool. These test cases are belong to Platform Independent Testing (PIT) model and do not rely on the specific platform. Secondly, the PIT is transformed into the Platform Specific Testing (PST) model to make the model executable. Finally, the test is performed by the interface framework which interpreted the PST.

2.1 Model

Models can be seen as an abstract of the device under test (DUT) or simulation. In general, there could be two kinds of models, the real device model and the simulation model. The real device model is used to present the special function or the device itself which should be tested. On the contrast, the simulation model is the function that the testing system should perform, which usually do some communication with the DUT.

2.2 Link

Link is used to describe the relationship between the models. In fact, it usually presents the direction of the signals such as input or output, and the physical interfaces such as Ethernet or RS-232. If there is more than one kind of interface between two models, there should be different kinds of links.

2.3 Variables

Variables are important components of Link. Variable is used to describe some definite data or some frames to deliver special meaning. For instance, a variable could be a UDP frame to deliver the current temperature of the DUT in Ethernet.

3 Interface Management Framework

As shown in Figure 1, the Interface Management Framework divided in three layers, the Interface Management Layer (IML), the I/O Channel Layer (IOL) and the Device Management Layer (DML).

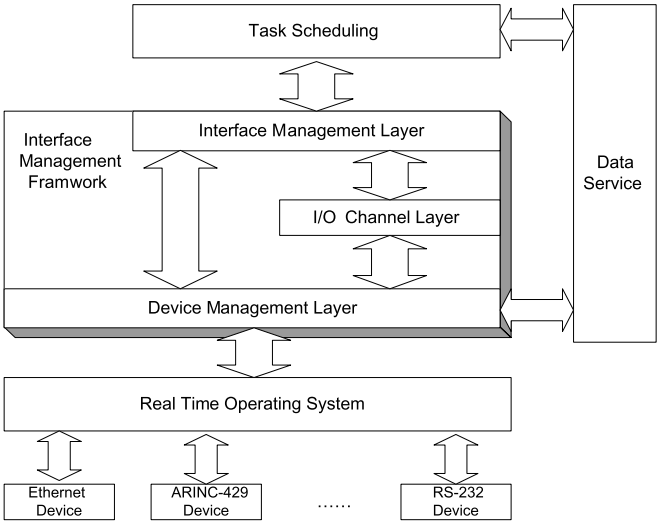

Fig. 1. Interface Management Framework Architecture

As we can see from the picture, the IML is the top layer. This module provides interface service to other modules in testing system such as Task Scheduling. In the process of test, Task Scheduling noticed the IML to handle the testing data. After that, the IML call the DML to send or receive the corresponding message.

I/O Channel Layer is a middle layer between the others. It is a useful part to manage the variety devices and the channels. Because of this layer, Driver Management could not care about the real number of channels on different kinds of devices but just use a logical number to identify them. The IOL holds the relationship between the logical channels and the practical channels. This information is used to check the validation of the user's configuration.

In the picture, the Device Management Layer is at the bottom. This layer transfers different drivers from several manufactures to a unified API. The other layers use these API to communicate with device drivers.

3.1 Driver Management Layer

Driver Management Layer is the interface of the framework which communicates with other modules in testing system. For instance, the task scheduling module uses the API from this layer to stimulate the process of sending and receiving data periodically.

This layer completes three main functions: static information management, dynamic information management and status management.

- **Static Information Management**

The static information management means the initialization after power on. As shown in Figure 2, the framework has lots of steps in initialization. First, after it was loaded by the operating system, the framework would find the configurations which were restored in a file. If the file could be read and contented valid information, the initialization would shifted to the next step. In the second step, the interfaces (such as Ethernet, RS-232) would be initialized according to the configurations. When all of the interfaces initialization finished, the framework would loaded the devices which provided the information of the configuration files and the driver files. Both fails in

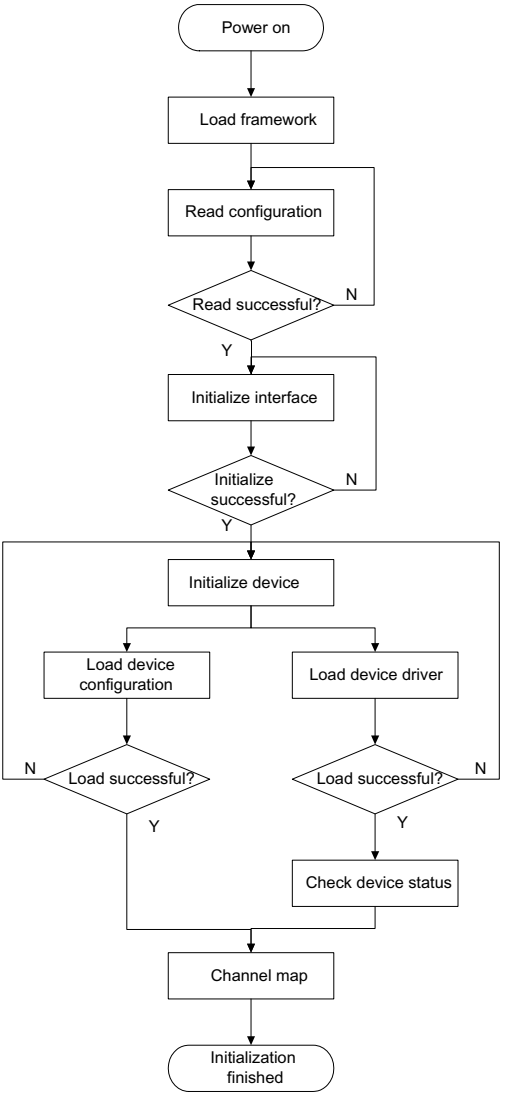

Fig. 2. Static Information Management Initialization

loading the files would lead to the device initialization fail. If the device driver was loaded successfully, the framework would use the state-checking function to acquire the device information (such as the current device number or whether the device was running). In the last step, the framework would record the information and marked the channels in the device was available.

- **dynamic information management**

The dynamic information management includes the variable registration, the device operation (such as open, close) and the test data operation (such as send, receive).

We use a structure to manage the information. When a variable is registered in the framework, it will acquire a handle to identify itself. This handle actually is a pointer to one structure in this kind.

The Figure 3 described the workflow of one test initialization. First of all, the model file will be loaded which content the information and construction of the models. If the file can be founded and holds the valid information, the models described in file will be analyzed and each of the variables will be registered to the framework. According to the variable registration, the first time of variable checking will be activated. The configuration of the variable will be compared with the corresponding one in the interface configuration file. If the variable is invalidating, the framework will record this information and report it to the end users. If all of the configurations are correct, the framework will bind the channel number of the variable to a real device channel. This process is called channel assignment.

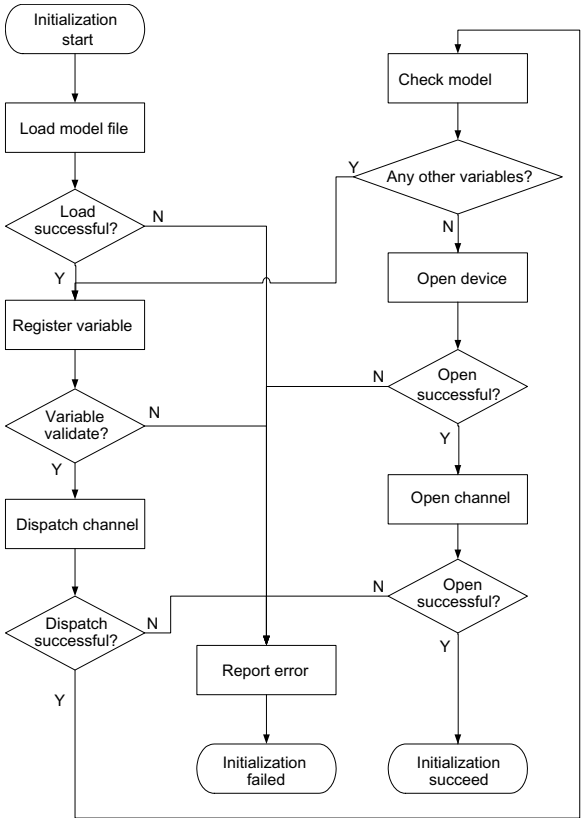

Fig. 3. Dynamic Information Management Initialization

After the entire variables are registered, the framework will open the devices using in test by calling the open functions in the drivers. Via the returns of the functions, the framework judges whether the open action succeed. The errors will also be record and reported. If the device is opened successfully, the configurations of the variables will be used to setup the device. This is the second time to check the validity. The inappropriate communication will result in the error returns from the setup functions. In the condition of all configuration passed, the framework will open the corresponding channel needed in test and the last time of variable checking is performed.

```
typedef struct _VarCtrl
{
    UINT iLabel;       // magic label
    UINT iBoardNum;    // real device number
    UINT iChnlNum;     // real channel number
    UINT iUsrChnlNum; // logic channel number
    ETHandle hMemID;  // used by data service
    IDrvBoard* pBoard;// pointer to device
}VarCtrl, *PVarCtrl;
```

In this structure, we firstly define a magic label which is used to confirm the handle is correct. In the label, we predefine a magic number and check whether the value is right. Next, we record the real device number, real channel number and the logic channel number for access the right device. The memory ID is used to access variable value in data service and is not cared in this article. In order to fast the process of operating data, we used a device pointer to keep the functions in device driver.

- **Status Management**

The status management predefines the system status as follow:

- ✓ Uninitialized
- ✓ Initialized
- ✓ Opened
- ✓ Running
- ✓ Paused

The uninitialized status means the initialization is not finished yet or failed. The initialized means the system is ready and can be used for test. "*Opened"* is a state that the models and variables are just analyzed and registered. In the *"Running"* state, the test is under processing. When a test paused, it can be resumed. If a test stopped, the system returns to the *"initialized"* state.

3.2 Dispatching Variables

The variables dispatching technology is used to evaluate the variable which receives the value from interrupt. When the device received a frame, an interrupt will be raised to notify the framework that there are values to be received. Then, the Interrupt Service Routine (ISR) reads the data from device buffer. After that, the variable dispatcher will be activated. It uses the predefined configurations to analyze the frame and determines which variable it belongs to. If the data received from device could not find a variable to dispatch, it will be seen as an illegal data and will be discarded.

4 Interface Information Management

The framework uses XML files to manage the devices and interfaces information including the configurations of channels. In general, there are two kinds of configuration files: the file for interface configurations and the file for device configurations. We describe both of them in details as follow.

4.1 Interface Configurations

The Interface Configuration File (ICF) is one of the core management files in the whole testing system. It records all kinds of interfaces and devices can be supplied by current system. It also contents the interface configurations and the path of the device configuration files respectively. Therefore, a real-time system should hold at least one ICF.

4.2 Device Configurations

The Device Configuration File (DCF) is used to describe a concrete device and records the configurations which could be customized by the end users. It mainly holds the number of channels in certain type of device and the channels' configurations. It also contents the path of the device driver files. Generally, one kind of device corresponds to one DCF.

5 Conclusions

The Interface Management Framework is coded in C++ and can be running in VxWorks real-time operating system now. It is an important element in a testing system named "Test Cube" which has applied to test a certain type of FCMS (Flight Control and Management System) and record a great deal of data for simulation and performance analysis.

References

1. Liu, B., Gao, X., Lu, M., Ruan, L.: Study on the embedded software reliability simulation testing system. Journal of Beijing University of Aeronautics and Astronautics 26(4), 59–63
2. Liu, B., Zhong, D., Jiang, T.: On modeling approach for embedded real-time software simulation testing. Journal of Systems Engineering and Electronics 20(2), 420–426 (2009)
3. Raistrick, C., Francis, P., Wright, J., et al.: Model Driven Architecture with Executable UML. Cambridge University Press, Cambridge (2004)
4. Duenas, J.C., Mellado, J., Cero, R., et al.: Model Driven Testing in Product Family Context. In: First European Workshop on Model Driven Architecture with Emphasis on Industrial Application. University of Twente, Enschede, the Netherlands (2004)
5. Hu, J., Hu, D., Xiao, J.: Study of Real-time Simulation System Based on Rtw and its Application in Warship Simulator. In: The Ninth International Conference on Electronic Measurement & Instruments, ICEMI 2009, pp. 3-966-3-970 (2009)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley/Pearson (2004)
7. Wind River Corporation, `http://www.windriver.com`