Implementation of the Model for Identifying and Referring Hardware in Complex Simulation Systems

Zhichun Zhang, Jingxin Xiao, Yun He, Zhenpen Zhao, Shengquan Tang, Jian Bu
Military Simulation Institute, Aviation University of Air Force, Changchun, 130022, China

***Abstract*—This paper focuses on complex simulation systems with a large-scale of hardware controlled items to resolve the problems about the identification, sampling , driving, transformation and reference of hardware items. A formal resolving model with its software supporting system is presented to decouple the interdependence between hardware driving developers and application developers, avoid repeatedly developing on similar works and enable the seamless integration of hardware driving routines and application programs. The controlled items are identified and referred in a two-dimensional notation and implemented by an encoding transmission technology to pack message packets. A reusable API(Application Programming Interface) system supporting the two-dimensional notation is realized. The interface system makes specific hardware devices, network medium and application deployment transparent based on external configuration files (hardware configuration files and a net configuration file). In addition, it supports the auto-detections on hardware detection online and precision maintenance by priority transmission packets. The notation and its reusable supporting system can be used in hardware-related systems especially in man-in-loop simulators to make the developing efficiently and the integrating seamlessly and easily.**

Keywords-simulation; hardware API; detection and maintenance; reusability.

I. INTRODUCTION

In many simulators, such as flight simulators, ship simulators, power station simulators, the trainee operates on the devices and the devices echo[1][2][3]. There are four types of programs in such systems: hardware program, communication code, simulation program and maintenance tool. A hardware program implements the conversation from analogue data to digital data and(or) from digital data to analogue data. The communication code transmits data between hosts in the system. A simulation program implements the simulation of objective system, such as aircraft engine, flight features. The maintenance tool can diagnose the hardware online and recover the hardware precision. Usually, a simulation system is developed as follows with some problems:

First, the hardware developer develops the hardware program to acquire the device data and describes the data in bit-based formats. How to identify the hardware items in a normal, simple, obvious notation in order to understand each other better for the developers is not resolved. A bit-based format is used in a system and a different format is used in another system, even in the same development organization.

Then the network developer develops the communication code to transmit these data to simulation programs. The communication code is based on the hardware items and host topology, thus these codes must be developed repeatedly according to different systems.

Finally, the simulation program developed by the simulation developer must parse these data in the bit-based formats to refer the hardware items. This bit-based reference is not obvious, efficient and often leads to code bugs. Even more important problem is that the parsed codes are different in different systems. The same problems exist for developing the maintenance tool as well.

To resolve these problems, this paper presents a method of separated the general and reusable components from the system that focuses on subjects as follows:

i) A simple, elegant notation to identify hardware items.

ii) Implementation of the notation in DLL(Dynamic Link Library) that can be used in different systems without any modifications needed.

iii) Auto maintenance detection supported by the DLL.

iv) To avoid network communication code development.

v) To decouple the interdependence with the developers and integrate their programs seamlessly.

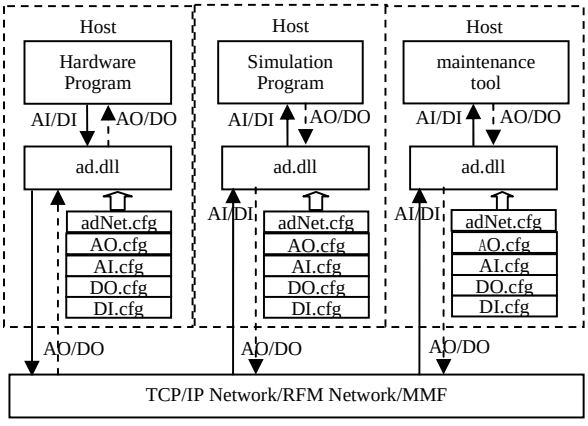

Figure 1. System deployment and data flows

II. OVERALL STRUCTURE

Figure.1 shows the system deployment and the hardware data flows between programs. The hardware program, simulation program, maintenance tool refer hardware items based on the interface system *ad.dll* and they communication each other performed by *ad.dll* based on the external configuration file *adNet.cfg* which describes the communication endpoints. The *ad.dll* supports three types of communication medium TCP/IP network[4][5], reflective memory(RFM) network[6] and memory-mapped file(MMF) [7] usually used in simulators. The *ad.dll* determines the hardware items based on the external configuration files *AO.cfg, DO.cfg, AI.cfg, DI.cfg*(see Section 3) that define the four types of hardware items in the system.

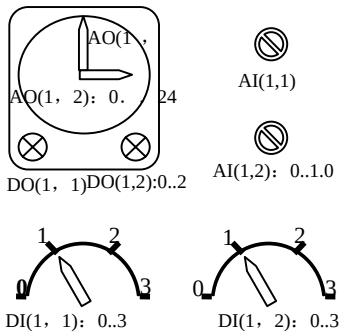

Figure 2. Hardware identifiers

III. TWO-DIMENSIONAL NOTATION

Each controlled hardware item on a device is identified showed in Figure.2.

An analogue output(such as a meter needle) is marked as *AO(i,j)*.An analogue input(such as a knob) is marked as *AI(i,j)*.An digital output(such as a light with two statuses:ON or OFF) is marked *DO(i,j)*. An digit input(such as a switch) is marked as *DI(i,j)*.

Where *i* is the number identifying the panel that the item on and *j* is the number identifying the item.

IV. INTERFACE SYSTEM APIS

The interface system *ad.dll* provides APIs(Application Programming Interface) for clients and some key APIs are listed as follows:

double& AI(int i, int j); //Returns the reference of hardware item *AI(i,j)*.

double& AO(int i, int j); //Returns the reference of hardware item *AO(i,j)*.

char& DI(int i, int j); //Returns the reference of hardware item *DI(i,j)*.

char& DO(int i, int j); //Returns the reference of hardware item *DO(i,j)* .

void DoNormal(); //Sets the system status in *NORMAL*.

void DoDetect(); //Sets the system status in *DETECTING*.

void DoMaintain(); //Sets the system status in *MAINTAINING*.

void DoRecv(); //Receives message packets from network.

void DoSend(); //Sends message packets through network.

V. HARDWARE ITEM REFERENCE IN PROGRAMS

A. A sample of the Hardware Program

Each hardware item can be sampled or drove in a C/C++ program as follows:

Step1: (Get current mode) *mode=DrvMode();*.

Step2: (Sample hardware items) Sample the hardware items and save their raw values in the variables *x1,y1*.

Step3: (Do precise operations) if *mode* is not *MAINTAINING* do precise operations on *x1,y1*.

Step4: (Update AI/DI) *AI(1,1)=x1; DI(1,2)=y1;* .

Step5: (Get driving AO/DO) Call *AO(int,int),DO(int,int)* to get driving values and save them in the variables *x2,y2*：*x2=AO(1,1); y2=DO(1,2);* .

Step6: (Do precise operations) if mode is not *MAINTAINING* do precise operations on *x2,y2.*

Step7: (Drive hardware items) Drive the hardware items with *x2,y2*.

Step8: (Do network communication) Call *DoRecv()* to receive *AOs/DOs* and Call *DoSend()* to send *AIs/DIs*.

B. A Sample of the Simulation Program

Step1: (Get sampled hardware items) *x1=AI(1,1); y1=DI(1,2)*; .

Step2: (Do simulation) Do simulation with *x1,y1* and the simulation results are saved in *x2,y2.*

Step3: (Update AO/DO) *AO(1,1)=x2; DO(1,2)=y2*; .

Step4: (Do network communication) Call *DoRecv()* to receive *AIs/DIs* and Call *DoSend()* to send *AOs/DOs*.

C. A Sample of the Maintenance Tool

Step1: (Set current mode) Call *DoDetect()* for detecting or Call *DoMaintain()*for maintaining.

Step2: (Get sampled hardware items) *x1=AI(1,1); y1=DI(1,2)*; .

Step3: (Display sampled hardware items) Display *x1,y1*; .

Step4: (Input driving data) Input *x2,y2* from the user interface(UI).

Step5: (Set driving AO/DO) *AO(1,1)=x2; DO(1,2)=y2;*.

Step6: (Do network communication) Call *DoRecv()* to receive *AIs/DIs* and Call *DoSend()* to send *AOs/DOs*.

VI. CONFIGURATION FILES

A. Hardware Configuration

The hardware configuration files includes *AO.cfg, AI.cfg, DO.cfg* and *DI.cfg*. These files are textual map of the diagram showed in Figure.2 so that the diagram can be identified by *ad.dll*. These files must match the following syntax:

ItemList:Item ItemList Item:AItem|DItem

AItem:AO(i,j)[:Type]|AI(i,j)[:Type]
DItem:DO(i,j)[:State.]|DI(i,j)[:States]
Type:float|double States:State..State
State:numberBasedZero i:numberBasedZero
j:numberBasedZero

B. Net Configuration

The net configuration file *adNet.cfg* is textual map of the deployment of hosts that receive and(or) send hardware item packets so that the deployment can be identified by ad.dll. These files must match the following syntax:

NetCfg: <Recv&Send=AUTOON |AUTOOFF>
<Recv>RecvCfgList</Recv>
<Send:WhatList>SendCfgList</Send:WhatList>
RecvCfgList:RecvCfg RecvCfgList
RecvCfg: Ethernet|Rfmnet| Mmfnet
WhatList: What[,WhatList]
What:AO|AI|DO|DI
SendCfgList: SendCfg SendCfgList
SendCfg: SendEthernet|SendRfmnet|SendMmfnet
SendEthernet: Ethernet Ethernet Period
SendRfmnet: Rfmnet Period
SendMmfnet: Mmfnet Period
Ethernet:<ip,port>
Rfmnet:<RfmCardID,BufAddress>
Mmfnet:<MmfFileName,FileSize，Offset>
RfmCardID:numberBasedZero
BufAddress:numberBasedZero
Period:numberBasedZero
MmfFileName:string
FileSize:numberGreaterThanZero
Offset:numberBasedZero

The functions *DoRecv()* and *DoSend()* perform communication based on this file *adNet.cfg*. For implementing a new communication system, the only task is to write one file *adNet.cfg* instead of the communication code development.

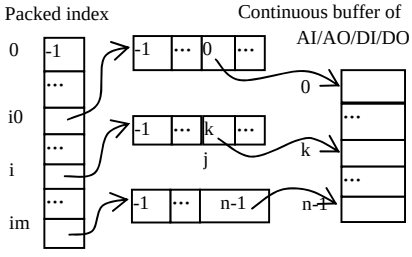

Figure 3. Packed index

VII. CONTINUOUS BUFFERS AND PACKED INDEX

The hardware items are continuously saved in the hardware item buffers in order to access any items in constant time *O(1)*. Figure.3 shows how to locate the hardware items in its buffer through the packed index. The hardware items lay on the panels from *i0* to *im*. The packed index is built based on the files *AO.cfg*, *AI.cfg* , *DO.cfg* and *DI.cfg* .

VIII. PACKET FORMAT

A packed packet consists of continuous sections ended in *NULL* section. There are five types of sections as follows:

i) *NULL* section

Byte 0: the type of the section, must be 0xff;

ii)AI section & AO section

Byte 0: the type of the section defines as follows:

0x01- normal AI; 0x11- detection AI; 0x21- maintenance AI;

0x02- normal AO;0x12- detection AO; 0x22- maintenance AO;

Byte 1- Byte N: the block sequence in the section and each block is presented as follows:

Byte 0-Bytes 1 (short): the index idx1 of the first hardware item in the current block when idx1 is not less than 0, otherwise idx1 is an end mark of the block;

Byte 2- Bytes 3 (short): when idx1 is not less than 0 these two bytes present the index idx2 of the last hardware item in the current block;

Byte 4-Byte M: the hardware items from idx1 to idx2 saved as double or float;

iii)DI section& DO section

Byte 0: the type of the section defines as follows:

0x03- normal DI; 0x13- detection DI; 0x23- maintenance DI;

0x04- normal DO; 0x14- detection DO; 0x24- maintenance DO;

Byte 1-Byte N: All DI or DO sequence and each item is saved in bit-based format packed based on the status number described in the file *DI.cfg* or *DO.cfg*.

IX. SYSTEM STATE MACHINE

The interface system *ad.dll* implements the state machine showed in Figure.4. The system does receiving operation *DoRecv()* and sending operation *DoSend()* in each state. The receiving operation includes receiving the packets, unpacking them and maintaining the system state based on the types of sections in the packets. The sending operation includes packing hardware items to the packets and sending them.

In *NORMAL* state, the packets are sent based on *adNet.cfg*. In *DETECTING* and *MAINTAINING* states, the sending operation is not based on *adNet.cfg* but the packets are only sent to the current maintenance tool detected by the interface system. In the other hand, in *NORMAL* and *DETECTING* states, the clients of the interface system *ad.dll* do precise operations. But in *MAINTAINING* state, the precise operations are avoided so that the maintenance tool can maintain the device precisions.

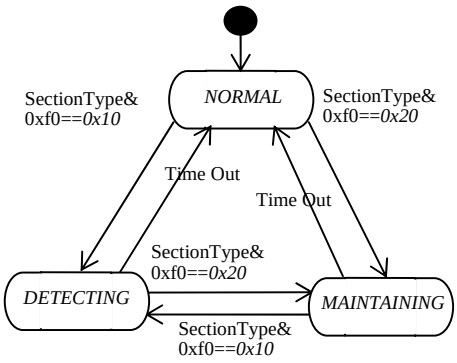

Figure 4. System state machine

X. CONCLUSIONS

This paper presents a two-dimensional notation to identify and refer the hardware items and implements an interface system *ad.dll* to support the notation. The notation is simple, elegant and is proper better for identifying a large-scale hardware items than a one-dimensional notation. The interface system *ad.dll* is generic components separated from the hardware-related system while the external configuration files *AO.cfg,DO.cfg, AI.cfg,DI.cfg* encapsulating the specific hardware and *adNet.cfg* encapsulating specific communication. The most tasks are implemented in *ad.dll* that can be used in different systems without any modifications needed. For implementing a new hardware-related system, the only task is to write four files *AO.cfg,DO.cfg, AI.cfg,DI.cfg* and *adNet.cfg* and no bit-based format parsing and no communication code developing needed. The *ad.dll* also supports the auto-detections on online hardware detection and precision maintenance by priority transmission packets and this will benefit the developing of maintenance tools and make the maintenance mode normalized. Obviously, the hardware item notation and its reusable supporting system *ad.dll* can be used in hardware-related systems especially in man-in-loop simulators to make the developing efficiently and the integrating seamlessly and easily.

REFERENCES

[1] LIU Xin-shun, YAN Jian-guo, Data communication and acquisition of mode calculation computer for UAV's hardware-in-loop simulation in VxWorks.*Modern Electronics Technique*, 35(01),pp.7-9, 2012.

[2] HE Bo-ling , ZHANG Zhi-chun, XU Wen, Modularized intelligent control method and its applications for a huge massive hardware.*Journal of Lanzhou University of Technology*，40（2），pp.106-109, 2014.

[3] XU Hai,CUI Lianhu,XU Guangyao, Research on realtime collection method of timing information in RTX environment. *Ship Electronic Engineering*,32(4),pp.59-61,2012.

[4] Buck Graham, *TCP/IP addressing : designing and optimizing your IP addressing scheme.*Morgan Kaufmann, San Diego, Calif. 2001.

[5] Michael J. Donahoo, Kenneth L. Calvert.,*TCP/IP sockets in C : practical guide for programmers, 2nd Edition.*Morgan Kaufmann Publishers, San Francisco 2009.

[6] SUN Yahong,Windows-based ditributed real-time simulation system. *Electronic Science and Technology*,25(03),pp.7-9, 2012.

[7] Jeffrey Richter, Christophe Nasarre, *Windows Via C/C++*. Microsoft Press, Washington 2011.