INSTRUMENT DRIVER DESIGN

Lan Vuu
Software Engineering Division (TISAD)
Ogden Air Logistics Center
Hill Air Force Base, Utah
84056
801-777-4480
Vuul@software.hill.af.mil

Amin Khan
Software Engineering Division (TISAD)
Ogden Air Logistics Center
Hill Air Force Base, Utah
84056
801-777-4585
KhanA@software.hill.af.mil

Abstract - Many organizations in Automatic Test Equipment (ATE) are involved in upgrading aging test stations or development of new test stations to state-of-the-art VXI plug&play technology. The instrument drivers for the F-16 Analog Test Station Sustainment (FATSS) were developed using the LABWindows/CVI environment with Object-Oriented (OO) programming design. The objective of this paper is to illustrate the driver development using this OO architecture.

INTRODUCTION

Object-Oriented (OO) Technology has been used throughout industry programming areas. The intention of the OO design toward the project is to make use of the benefits available through reuse of driver programs with minimum impact to the Test Program Set (TPS). Its benefit gives the ATE engineers the flexibility to reuse their code for the same instrument from different manufactures or the same code for multiple instruments. Traditionally, the engineers have to re-write entire programs to accomplish the above tasks. With the new approach, it would accelerate the driver development process in coding, improve developer effectiveness and productivity, and reduce document generation.

HARDWARE AND DISTRIBUTOR SELECTIONS

The OO approach to instrument driver development was applied to the upgrade effort of the aging F-16 Analog test station. Hardware and distributor selections were completed in the early stage of the development. Instruments were chosen according to the critical item product function specification for the test station itself. All instrumentation purchased utilized either VXI plug&play technology or GPIB. It was not always necessary to select instrumentation performance capabilities that conformed exactly to required parameters. Some hardware was purchased if software solutions were available to manage differences between instrumentation specifications and available equipment.

DEVELOPMENT ENVIRONMENT TOOLS

Drivers were developed in a common integrated development environment (IDE). The IDE used was selected because it supported development of drivers in text based computer language (C). The IDE also provided capabilities to generate dynamic link libraries (DLLs), use vendor provided object modules, and to use other common software libraries including Graphical User Interface (GUI). For a detailed discussion, please refer to "Using Windows Messaging To Control Automatic Test Equipment" [1].

DRIVER PROGRAMMING DEVELOPMENT

An OO approach provides software access to hardware through a single entry point called the public access layer. An Interface Control Document (ICD) was used by the programmer to specify the input and output parameters needed to communicate with the public access layer. The public access layer provides a link between the rest of the software world and the hardware driver layer that performs specific tasks associated with the hardware. The
hardware layer calls instrument manufacturers driver functions. If there's a change in instruments used by the test station, only the hardware driver layer needs to be modified. A test plan was developed and used to evaluate the effectiveness of the driver design.

Figure 1: Instrument Driver Design

PUBLIC FUNCTIONS LAYERS

The driver development starts with the design of the driver program structure (Figure 1). There are three levels in the driver architecture – public function, hardware driver, and vendor function layers. The purpose of the public function layer is to analyze input parameters and perform appropriate hardware driver layer function calls. It provides a single entry point for all access to the hardware it controls.

The ICD defines the bridge between parameters available in TPS instrument calls and the public function prototype. The driver programmer uses the ICD to determine the required input and output parameters available for communication with the public access layer. The public access layer is responsible for linking the identified parameters from the ICD to the hardware driver layer that performs specific tasks associated with the hardware.

All public functions Input / Output (I/O) data is passed through one of three structure pointers. The pointers reference structures that contain input information, output data, and error information. All data passed by the later two structures are filled in the public function layer. This generic I/O format was employed to provide a single common access format between all public functions and TPS code. The input information pointer contains all parameters used by TPS source code (ATLAS) utilized in driver operation. The output data pointer is to be filled by the public function with data returned from the driver layer. This is filled by functions that utilize sensor instrumentation to perform measurement. The error pointer is also filled by the public access function. The error pointer is required to return an integer and a string to the function caller when an error occurs. Often English language error strings are available through vendor layer functions.

Public functions were often required to provide a single interface to multiple instruments. The individual instrument utilized is specified by one of the input parameters. Handling of all specific instrument functions like initialization, configuration, and set-up are passed through to the hardware driver layer.

HARDWARE LAYERS

The hardware layer serves as a wrapper which passes information between the public function layer and the vendor driver layer. Hardware driver layer functions were designed to look and feel exactly like vendor driver functions. When instrument driver designs are done well only the hardware driver layer should be modified when instrumentation is replaced with hardware from a new vendor. Vendors frequently design instrumentation that is similar to their competitors. The vendor driver layer function may require only a new vendor function name, and a new order for passed parameters.

Software drivers that are supplied by the vendors normally have significant differences occur. It will require some form of tweaking to work properly. This layer is served just that, as a location to make software adjustments required by vendor driver layers. These adjustments are made to assure that detailed requirements of a specific vendor's hardware are met. This layer is used to provide software safe guards used to protect instrumentation from programmed requests for out of range operation.

The instrument selected to replace the legacy test station counter used different input signal filtering. The driver layer programmer resolved the problem by the
generation of new filtering handling code to compensate for the incompatibility of the instruments.

Another example is the pulse generator purchased required programming to a minimum rise time. The legacy TPS code and translator passed a value of zero if a minimum rise time was desired. In this case the instrument minimum rise time was forced into the vendor driver call at the hardware driver layer.

**VENDOR LAYERS**

Hardware vendors provided this layer. It was used to perform hardware interface functions. These functions were not modified unless they did not work properly.

**TEST PLAN**

The final process of the driver development is to generate a test plan; it is based on the critical item product function specifications. The developers used it to validate the effectiveness of the driver design.

**CONCLUSION**

The methods of an Object-Oriented Instrument Driver in ATE will greatly benefit the development of new test systems and the re-host of Legacy systems alike. The instruments will be interchangeable based on industry standards, reduced in learning curve, software maintenance, and improved the development process.

**REFERENCES**
