Using MFC and Managed C++ to Realize Configuration Utility

Laiping Wu, Tao Lin, Libin Chen

Beijing Institute of Radio Measurement, Beijing, China

*wlp21@163.com

Keywords: IVI configuration utility; XML; Managed C++

Abstract.IVI Configuration Store is the key factor in the interaction between users and instruments. Usually, users need a GUI configuration utility to write the configuration information into IVI Configuration Store. But the IVI Foundation does not offer a specific configuration utility. In this paper, a configuration utility realized by MFC and Managed C++ is introduced. By loading “Ivi.ConfigServer.Interop” program sets, an interface-friendly and information-completed configuration utility is developed. The test results prove that the configuration utility can achieve intact configuration information of IVI-COM.

Introduction

To achieve the interchangeability between instruments without recompiling or re-linking, users have to identify those specific drivers and hardware resources should not be used in the test program. Thus, an external configuration store is required to dynamically load the configure information for specific drivers. In IVI, this store is IVI Configuration Store[1]. The IVI Foundation has defined two types of driver wrappers, IVI-C and IVI-COM. For ease of use, instrument driver suppliers always provide an IVI configuration utility to assist users in modifying the IVI configuration store. But as the IVI Foundation does not define an IVI configuration utility, thus, it is likely that multiple IVI configuration utilities can be attained. Then, how to configure the IVI-COM instrument drivers is a problem eagerly needs to be solved. This paper gives a solution to achieve a IVI-COM configuration utility with MFC and Managed C++.

MFC and Managed C++

The IVI Foundation has supplied “Ivi.ConfigServer.Interop” assembly for users to develop the configuration utilities, but the assembly can be only used in .NET framework. MFC is a library that wraps most of the Windows API in C++ classes, including functionality that enables uses to develop with default application framework. Moreover, MFC defines many handle-managed Windows objects,which are very flexible for developing controls. Managed C++ is based on standard C++, and used to design the .NET Framework applications by Visual C++ programmers. It not only keeps all the characters of Standard C++, but also support automated memory management and interoperability with other. NET languages [2]. This mixed programming method does well in accessing to the . NET assembly, and it can implement a friendly user interface easily.

In this paper, the developing environment is Visual Studio 2005. In order to use the “Ivi.ConfigServer.Interop” in MFC, several settings are required.

Firstly, add CLR(Common Language Runtime Support) in MFC as follows: Project->Properties->Configuration Properties->General-> Common Language Runtime Support (/clr).

Secondly, add Ivi.ConfigServer.Interop in the project. The step is: Project->References->Add New Reference->COM->Ivi Configuration Server 1.6 Type Library.

After these two steps, users can operate the functions of “Ivi.ConfigServer.Interop” during programming directly.

IVI Configuration Server

The IVI Foundation has defined "IVI-3.5: Configuration Server Specification". The IVI Configuration Server is responsible for providing system database services to IVI-based measurement system applications. Specifically, it provides system initialization and configuration information. Based on that, the configuration utility is developed[3] .

The data of the Configuration Server is stored as an XML configuration store file, which closely follows the design of the Configuration Server, named IVI Configuration Store, which is the kernel of the IVI Configuration Server. Furthermore, the IVI Configuration Store is a standard mechanism, which can offer these IVI resources to the measurement system. Figure 1 shows the simplified IVI configuration server class diagram. In the diagram , the IVI Configuration Server has six global classes, namely, IviLogicalName, IviHardwareAsset, IviSoftwareModule, IviSession, and IviDriverSession, IviPublishedAPI. The IviDriverSession class is inherited from the IviSession class[4]. All the objects of the global class in the Configuration Server are unique (by Name), except Published APIs.

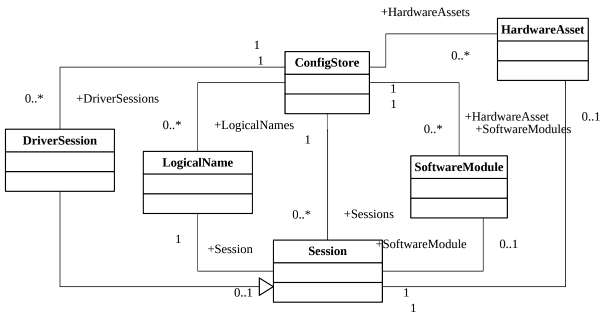
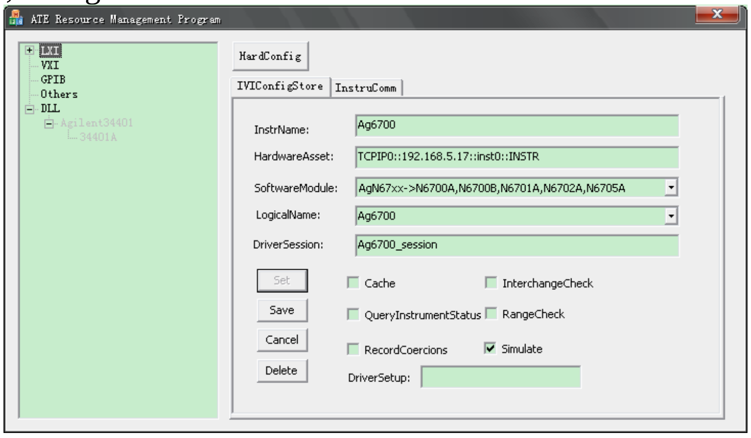


Fig.1 Simplified IVI configuration server class diagram

Development of IVI Configuration Store

To speak simply, the Configuration Store is just a database. This database holds information about IVI drivers installed on the computer and configuration information for the instruments in the test system. The Configuration utility facilitates the process of "manually" editing of the configuration store. In one way, the configuration utility described in this paper can be used to edit the configuration store via a graphical user interface. In the other way, it can also check the status of the instruments if needed, as Figure 2 shows.



InstrName: Ag6700
HardwareAsset: TCPIP0::192.168.5.17::inst0::INSTR
SoftwareModule: AgN67xx->N6700A,N6700B,N6701A,N6702A,N6705A
LogicalName: Ag6700
DriverSession: Ag6700_session

Cache ☐ InterchangeCheck ☐
QueryInstrumentStatus ☐ RangeCheck ☐
RecordCoercions ☐ Simulate ☑
DriverSetup:

Fig.2 The graphical user interface of the configuration utility

The configuration utility should display the configuration information automatically. It also allows users to add, modify and delete instrument information in the database. However, it should be noted that the configuration utility could not add or delete software modules in the master configuration store. After each operation, the configuration utility will automatically save an IVI Configuration file. The default file path is C:\Documents and Settings\All Users\Application Data\IVI Foundation\IVI\IviConfigurationStore.xml.

Display configuration

In the figure above, when users click the instrument on the left column, the information of the instrument will be displayed. If the instrument is not configured, a blank table is displayed for users to fill out. And if the instrument has been configured already, then detailed information of the the instrument is showed. This function is achieved by transmitting the Resource Descriptor. When users click the instrument, the auto-display module will search the Resource Descriptor. associated to the selected instrument in the HardwareAsset collection. If the Resource Descriptor exists, then all the other contents are displayed according to the HardwareAsset item. If not, it means that this instrument is not configured. If the instrument is online, then the Resource Descriptor of the instrument is displayed. If not, that is only the driver of the instrument can be found in the system, then all the instruments supported by the driver will be revealed in the SoftwareModule, as “AgN67xx” showed in Figure 2.

The flow chart of the auto-display module is listed in Figure 3.

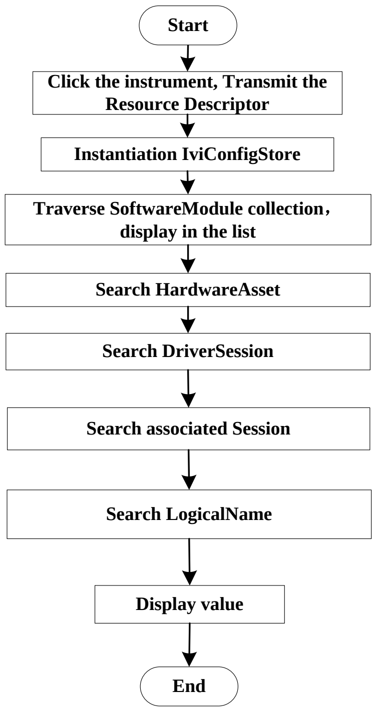

Start
Click the instrument, Transmit the Resource Descriptor
Instantiation IviConfigStore
Traverse SoftwareModule collection, display in the list
Search HardwareAsset
Search DriverSession
Search associated Session
Search LogicalName
Display value
End

Fig. 3 The flow chart of display configuration

In Fig.3, we can see that traversing SoftwareModule collection is used for users to add configuration. Cause of the logic relationship among HardwareAsset, DriverSession and LogicalName, both of DriverSession and LogicalName could be found through HardwareAsset.

Add configuration

In the figure 2, when users select the instrument which does not exist, then a blank table will be offered to fill out. Add configuration requires users to add the nodes “HardwareAsset”, “DriverSession”, “LogicalName” and “SoftwareModule”. “InstrName”node in the figure is a property of IviHardwareAsset class, and users can enter the name of the instrument in the node. In the “HardwareAsset” edit box, Resource Descriptor of the instrument will be filled. Under normal conditon, when the instrument is online, the edit box is automatically associated; but if the instrument is not online, users need to input the value, which must be the same with the Resource Descriptor of equipment being used. ”DriveSession” is inherited from “Session”, but it has more initialization information. Taking the integrity of configuration information into account,“DriverSession” is adopted in the design. The way of check box for users is used to select the initialization information.

In order to facilitate users to write the “SoftwareModule”box, when the system initializes, “SoftwareModule” item will display all the drivers and their support equipment types in the drop-down list, so that it can be true that users simply fill out configuration only select the corresponding instrument drive from the drop-down list.

Configuration information is always associated with each other. “HardwareAsset”, “DriverSession”, “LogicalName” and “SoftwareModule”, these four are not allowed to be left empty in the filling process (except “DriverSession” initialization information), and data from different instruments are also different from each other. Therefore, in the design, the empty and repeated data warnings are set, in order to ensure the information is correct.

According to the reference relations of classes in the IVI Configuration Store, “HardwareAsset” must be filled out first, because it is the only one with no reference to other types of nodes. Next is “SoftwareModule”, then is “DriverSession”, finally is LogicalName item. The flow chart of adding configuration shows in Figure 4.

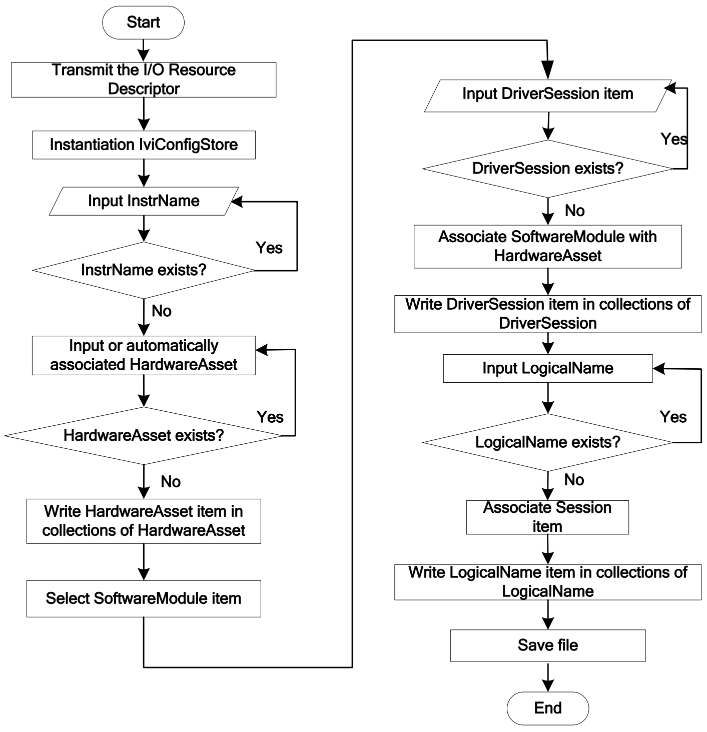

Start
Transmit the I/O Resource Descriptor
Instantiation IviConfigStore
Input InstrName
InstrName exists?
Yes
No
Input or automatically associated HardwareAsset
HardwareAsset exists?
Yes
No
Write HardwareAsset item in collections of HardwareAsset
Select SoftwareModule item
Input DriverSession item
DriverSession exists?
Yes
No
Associate SoftwareModule with HardwareAsset
Write DriverSession item in collections of DriverSession
Input LogicalName
LogicalName exists?
Yes
No
Associate Session item
Write LogicalName item in collections of LogicalName
Save file
End

Fig.4 The flow chart of adding configuration

Delete Configuration

If the parameters of the instruments in the system are changed, negated or need to be replaced, all the related configuration information must be deleted from the configuration store. The utility can delete “HardwareAsset”, “DriverSession” and “LogicalName” in the configuration store rather than the SoftwareModule. The “SoftwareModule” is the entrance of the instrument driver. It is added to the configuration store automatically during the installation of the driver program, and is removed during the Uninstallation of the driver program.

On the contrary, deleting configuration information should begin with the latest referenced class. That means when deleting configuration information, “LogicalName” ought to be deleted first, then comes “DriverSession”, and “HardwareAsset” is the last one. If a class is referenced by another class, it will not be deleted. For example, if “HardwareAsset” in the design is deleted, then a system error occurs. It is just because “HardwareAsset” is referenced by “DriverSession” and can’t be deleted. The selected “HardwareAsset”, “DriverSession” and “LogicalName” should be deleted together, and can’t be deleted one by one. That is to say, if one of these three is incorrect, all these items must be deleted and refilled.

The flow chart of deleting configuration shows in Figure 5.

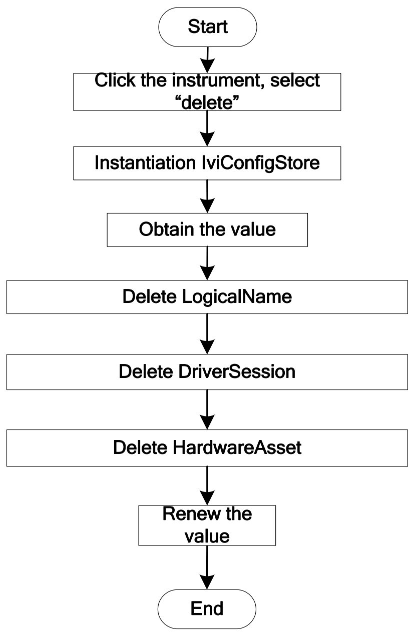


Fig.5 The flow chart of deleting configuration

Summary

IVI drivers have become prevalent in the test and measurement market over the past decade[5],especially for the LXI instruments, which defines IVI as a standard driver. Hence IVI configuration utility is urgently needed by the automatic test systems with LXI devices . This paper gives an overview of the process and product to achieve the IVI-COM configuration utility by using MFC and Managed C++. The configuration utility can display the information of the instruments automatically, and allow users to add, delete the information. Experiments show that the configuration utility developed in this paper can be easily carried out on the IVI-COM instrument drivers. The configuration files can be called correctly by IVI session factory, and the test systems can achieve instrument interchangeability in the same IVI class.

References

[1] IVI-3.5: Configuration Server Specification. Version 2.3. 2013, 15

[2] Ding Youhe. Series of application and development on Visual C++.NET -- Managed C++ program design. Computer Programming Skills & Intenance. 2002, (8),pp.1~2

[3] Mark Meldrum. Mapping Multiple Legacy Instruments and Associated Measurements into a Single Synthetic Test Environment. IEEE AUTOTESTCON 2008 PROCEEDINGS, 2008,pp.237~242

[4] Li Shicheng. Research of Key Technology in the Signal-interchangeable Virtual Instrument. Harbin Institute of Technology. 2009,pp.35~37

[5] Kirk Fertitta,Simplifying Test System Development with IVI.NET. IEEE AUTOTESTCON 2012 PROCEEDINGS,2012,135.