HARDWARE INTERFACE INDEPENDENT SERIAL COMMUNICATION*

P. Kankiya, J. Jamilkowski, L. T. Hoff, BNL, Upton, NY, USA

Abstract

The communication framework for the in-house controls system in the Collider-Accelerator Department at BNL depends on a variety of hardware interfaces and protocols including RS232, RS485, GPIB, USB, Ethernet and CAN. IISC is a client software library which can be used to initiate, communicate and terminate data exchange sessions with devices over the network. It acts as a layer of abstraction allowing a developer to implement communication with these devices without having to be concerned about the particulars of the interfaces and protocols involved. Details of implementation and a performance analysis will be presented.

INTRODUCTION

A common problem faced when providing equipment control infrastructure for different accelerator branches such as vacuum, power supply, RF etc. is the diversity in communication protocols which is produced by the competing manufacturers. There are multiple factors such as cost, functionalities, and features involved in buying electronic equipment's and it is difficult to set a standard rule when choosing a device for a specific task. Hence the controls engineers are usually presented with varied choices of hardware interfaces. It would be ideal if somehow there was a way to generalize the software development effort involved when bringing each of these pieces of equipment on-line with a minimal code bloat and duplication.

Overview of Existing Systems

Particle accelerators are fitted with power meters, motion controllers, actuators, sensors etc. The control access to these instruments is provided by a logical device object known as ADO (accelerator device object). ADO is a C++ container type class which provides a software view of a collection of collider control points known as parameters. It is traditional practice to house the code for establishing connection with a device via a standard bus interface system, inside the ADO program. The diversity in physical interfaces makes the ADO development process some-what non-uniform.

This also means software developers, must have detailed know-how of every communication protocol in use. Although the nature of the communication is uniform mostly throughout the hardware interface protocols, their varying methods of implementation add a significant level of code complexity.

* Work supported by Brookhaven Science Associates, LLC under Contract No. DE-AC02-98CH10886 with the U.S. Department of Energy.

In absence of the abstraction layer, all bus specific IO operations are implemented inside the ADO code. It makes ADO code more error prone, and susceptible to memory leakages. Also adds the overhead of shared resource management.

There have already been efforts in the industry to standardize the hardware communication interfaces by using message protocols such as SCPI which is a standard for syntax and commands to use in controlling programmable test and measurement devices [1]. Since a large number of devices follow this command set, it will add a great amount of code re usability if the API sending these operation strings is also standardised. This helps in automating the process of basic communication with any kind of device. Later the application ADO can be customised to add specific intended functionality.

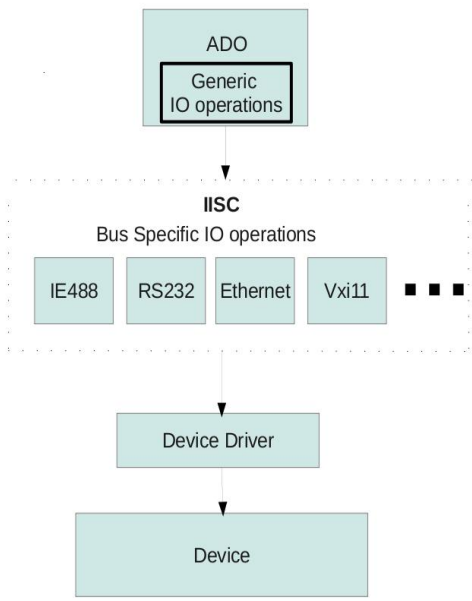

Figure 1: Addition of IISC layer in ADO code.

PROPOSED SOLUTION

In this paper we focus on defining an approach to a software abstraction layer called IISC which eliminates the code complexity due to the variety of physical bus interfaces.

The idea behind this solution is to base programming logic on the interfaces of the high level objects used, rather than on internal implementation details. IISC is a tool to provide a steady and homogeneous abstraction on top of accelerator devices whose hardware

ISBN 978-3-95450-139-7

implementation is heterogeneous and evolves over time[2].

SOFTWARE IMPLEMENTATION

The implementation mechanism adopted for IISC library is based on the design pattern called the factory method in C++[3]. The intention of factory method is to define an interface for creating an object, but let the sub-classes decide which class to instantiate. The goal of this design pattern is to redirect the normal way of instantiation of class X to somewhere other than the constructor without unacceptable consequences (memory leakages) or a performance hit. A factory pattern is used here because we need to create one of many possible objects based only on information that is known at runtime.

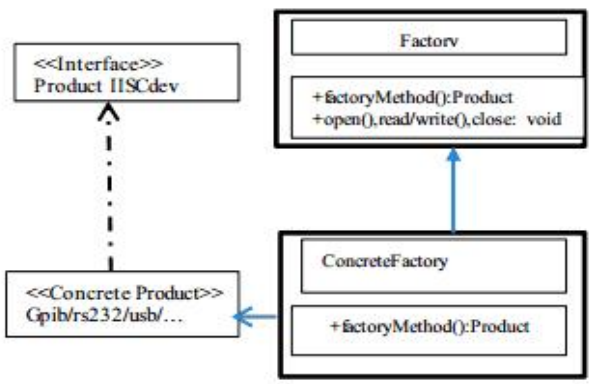

Figure 2: Class Diagram.

The UML diagram (Figure 2) explains the hierarchy of the classes and how the pattern works.

- **Product:** here is the IISCDev class which is the abstract class responsible for separation between the application and a family of bus interfaces, by introducing weak coupling instead of tight coupling hiding concrete classes from the application.
- **Factory:** declares the method FactoryMethod, which returns a IISC object.
- **ConcreteFactory:** overrides the generating method for creating concrete IISC product objects.
- **Concrete Product:** the subclass object returned by Factory. E.g. GPIBDev, ENETDev, etc.

The product class constitutes following fundamental member functions to establish communication: devOpen, devConfigure, devWrite, devRead, devClose, devReonnect. These functions are common to all bus interfaces. The support libraries (e.g. “ni488.h” for IE488 bus interface), provide variants of these member functions. These member functions in the base class IISCDev, are declared virtual, and will be overridden in the child class with respect to each bus interface class.

An IISCDev is a "handle" for accessing hardware interface services and for making low level system calls implemented by drivers. Therefore it defines a logical division of code between control parameters and device driver libraries[4].

It maintains the following information about the active communication irrespective of the protocol class that is instantiated through the factory method.

- IISCDevStatus - A class variable to describe the status returned by methods, could be errors.
- IISCDevException - A class variable to describe exceptions.
- IISCDevLock() - A method for shared resources .
- IISCDevInterface - A class describing the interface.
- IISCDevConfig - A class containing configuration parameters based on protocol in use.

A key function IISCDevConnectionMgr() is responsible for maintaining connection with hardware throughout the life of a program. This is attained by constantly polling the device for valid data, and implementing a reconnection mechanism in absence of valid data. Since this process is vital regardless of the protocol in use. This functionality can be easily embedded in the IISC layer without the software developer having to write code for each new application.

Member methods enlisted inside IISDev class are class is implemented as a wrapper function inside the subclasses. These wrappers are written on top of IO libraries which are used as, cross-platform C++ library[5] for network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach.

PRACTICAL IMPLEMENTATION

A use case for testing IISC library is a set-up for data acquisition from a delay generator device via IE488 standard interface.

An ADO program was written to provide user access to this data. With addition an abstraction layer (Figure 1) the task of IO operations are deferred from the ADO to the IISCDev class.

An instance of IISCDev class is instantiated in the ADO code. The runtime arguments of constructor of ADO provides the type of interface in use. Hence, the factory returned a GPIBDev type of object.

The generic IO method devOpen, is used to open gpib device in participation. The devOpen function is transient wrapper function which is overridden by the national instrument's ibdev() function call[6] inside the subclass. In similar manner other device IO operations are carried out.

This formulation of the ADO class resulted in prominently reduced code size due to omission of interface details. This also standardises the development effort.

ISBN 978-3-95450-139-7

One of the advantages of execution of layered architecture is, the ADO layer is allowed to make changes in physical interface implemented without any code alteration or recompilation . For example, the device used in this test set-up , is able to utilise Ethernet communication with only changing ADO constructor arguments and not internal coding algorithm.

Scope of Improvement

For future versions of IISC library, it will be beneficial to include call back routines for better and faster asynchronous data update. It will allow more than one client for a shared resource simultaneously.

Advantages and Limitation

With the help of IISC layer, the standard code interface with device drivers becomes easier to maintain. This library is absolutely portable because it has no dependence on the existing BNL controls framework. It is easily extensible to many more hardware bus interfaces by simply adding a new sub-class, and bus specific configuration parameters. IISC being a high level abstraction layer can add small amount of delays when exchanging messages with the device.

CONCLUSION

A software abstraction library has been implemented to hide details of implementation of standard communication bus interfaces. It is used to simplify and tailor the interface to a client code module ADO with the intent of making it more intelligible or relevant to the user. Features for future versions of the library are discussed and some performance limitations have been highlighted.

REFERENCES

[1] http://sdphca.ucsd.edu/Lab_Equip_Manuals/SCPI-99.pdf
[2] “EQUIPMENT SOFTWARE MODELLING FOR ACCELERATOR CONTROLS” Michel Arruat, Stephen Jackson, Jean-Luc Nougaret, Maciej Peryt, Accelerators and Beams Department, CERN, CH1211 Geneva 23
[3] “Design Patterns: Elements of Reusable Object-Oriented Software” By: Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides
[4] “EPICS: ASYNCHRONOUS DRIVER SUPPORT” Martin R. Kraimer, Mark Rivers,Eric Norum, Argonne National Laboratory, Argonne Illinois 60439 USA
[5] Boost Asio Library by Christopher Kohlhoff
[6] NI-488.2M™ Software Reference Manual.

ISBN 978-3-95450-139-7