DEVELOPMENT OF GENERALIZED DEVICE LAYER FOR THE COACK SYSTEM

Masakatsu Mutoh, Yoshinobu Shibasaki and Isamu Abe*
Laboratory of Nuclear Science, Graduate School of Physics, Tohoku University
1-2-1 Mikamine, Taihaku-ku, Sendai 982-0826, Japan
*High Energy Accelerator Research Organization (KEK)
e-mail: mutoh@LNS.tohoku.ac.jp

Abstract

A generalized device layer for the COACK system has been developed. Software architecture for accelerator control systems commonly consists of three layers. COACK is a generalized control kernel having highly useful functions, and is installed in a middle-layer of the three layers. In regard to the lower device layer, a unique device layer has been developed that is adaptable to the target control system under various conditions. In the accelerator control system, PLCs or instruments using GPIB are employed to control various accelerator components such as magnet power supplies, RF components, vacuum components, beam monitors, and so on. If configurations of the accelerator components, interface specifications and control procedures and protocols could be standardized, a generalized device layer could be constructed. In order to realize the generalization of the device layer, XML is used to express the control information for device layer as hierarchical data. The device layer is installed automatically according to its control information. Consequently, the developed device layer can be applied to various control systems by means of only rewriting files described the control information in XML without any modifications.

1 INTRODUCTION

Software architecture for accelerator control systems commonly consists of three layers: a human interface layer such as an operation console, a control layer that functions as a control manager in the control system and a device layer that behaves like an active interface between accelerator components and the control system. Since adopting such a three-layer system provides a clear definition of each function, it becomes possible to distribute the control functions through a high-speed computer network to realize an accelerator control system at the most suitable scale. If generalized layers could be developed and made available, it would eliminate the need to design individualized programs for accelerator control systems, shorten the time frame for development, and reduce the overall development burdens, which include construction budget and manpower. Moreover, the generalized software would be used in many universities and/or institutes undertaking large experiments, so that it could be grown into more sophisticated software in collaboration with researchers. For these reasons, we have developed a generalized device layer for accelerator control.

In order to design a generalized device layer, it is important to document control information, such as configurations of the accelerator components, interface specifications and control procedures and protocols. We employed XML (eXtensible Markup Language) to express the control information. The control information is structured as hierarchical data in XML, and can be clearly shown in detail using the advantageous elements of XML. A proper tool named "Driver builder" is provided to make the XML document required in the device layer. The XML document is simply generated with basic control information entered from the display window of the driver builder. IOC (Input/Output Controller) works as a core of the device layer; when IOC starts up, it loads the XML document. Driver objects corresponding to each control action for accelerator components are created; then, the initial interface processes are executed in line with initialization commands described in the XML document. The Driver objects in IOC control the accelerator components.

COACK (Component Oriented Advanced Control Kernel)[1][2] is a generalized control Kernel developed to apply not only to accelerators, but to various other large physics equipment, as well; it is mainly used in the middle layer of the control system. Component-oriented software technology is adopted, and each function of COACK is designed as a software component. COACK has spread gradually to universities and institutes through applications such as controlling accelerators or large experiments. Until the development of such a unique device layer, adaptable to target control systems under various conditions, enormous expenditures of time and money were involved in the creation of such systems. That generalized control software reduces development burdens has been proven by the arrival of COACK. The device layer we have developed can be used with COACK as the middle layer; we expect that the combination of COACK and the developed device layer will be useful for the construction of flexible and effective accelerator control systems.

2 STRUCTURE OF THE DEVICE LAYER

A block diagram of the device layer is shown in Figure 1. The device layer controls the accelerator components such as magnet power supplies, RF components, vacuum components, beam monitors, and so on; it consists of the IOC and the driver builder.

The device layer supports widely-used PLCs and GPIB devices as interfaces between the control system and the

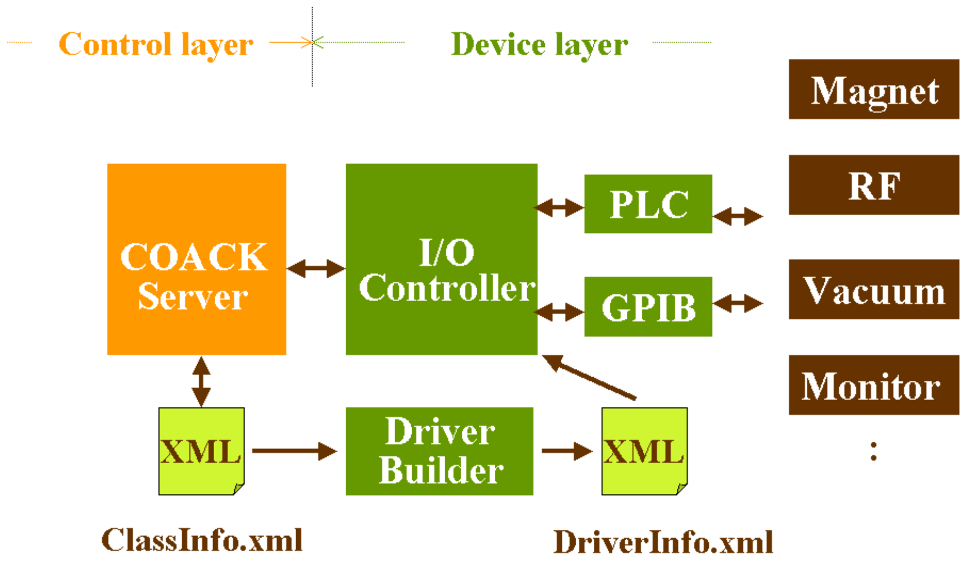

Figure 1: Block diagram of the device layer.

Table 1: Element and attribute list in XML document.

Element	Attribute	Example
Driver		
Interface	DriverClass	PLC/GPIB
Instrument	PLC	
	InstrumentID	YEW01
	IpAddress	130.34.61.97
	PLCType	YOK.FA.E
	GPIB	
	InstrumentID	HP01
	BoardNo	0
	GpibAddress	1
	Initialize	*RST;:INIT:CONT ON
Node	NodeID	ND01
	DeviceClass	Linac.Magnet.Q
	NodeName	Q1
Instance	PLC	
	InstanceID	INS01
	Property	Current
	AccessMode	Write/Read
	PropertyValue	
	RelayAddress	D00001
	DataType	Binary+Sign
	BitLength	16
	RelayValue	
	DeadBand	0.05
	RapidTime	1000 (mSec)
	RapidValue	0.1
	GPIB	
	InstanceID	INS02
	Property	Voltage
	AccessMode	Write/Read
	PropertyValue	
	DeadBand	0.1
	Command	:MEAS:CURR?

accelerator components. PLCs connected only with Ethernet can be controlled right now. Recently, not only measurement devices such as frequency counters and precision digital multi-meters, but also power supplies for magnets have been integrated as GPIB devices into accelerator control systems.

Here's how the system works. Control commands sent from COACK in the middle layer are accepted and processed by IOC, and are transmitted the accelerator components through PLCs or GPIB devices. IOC in turn observes accelerator operations, and informs COACK of the accelerator's operation status.

COACK possesses the control information on each accelerator component; control information is also treated as an XML document named "ClassInfo.xml". ClassInfo.xml is used to install a data cache in COACK for temporally saving the control commands and operation status messages from the accelerator. The driver builder adds the information on PLCs and GPIB devices to the information derived from ClassInfo.xml on the accelerator components controlled by the device layer, and generates the hierarchically structured XML document.

2.1 Structure of the XML document

The relevant information can be expressed as a hierarchically structured data by the nest function of XML. Addition and rearrangement of attributes in the element are flexible, therefore making it possible to add new interfaces to the device layer.

The configuration of the accelerator components, the interface specifications, and the control procedures and protocols are described as the XML document.

The hierarchically or "tree"-type structured data is composed of the following elements and attributes, as shown in Table 1:

- Driver-element; root element.

- Interface-element; the attribute designates interface type PLC or GPIB.
- Instrument-element; the attributes specify property values of the PLC or GPIB device and initial process commands in the case of GPIB devices.
- Node-element; the attributes show the accelerator components controlled by the driver layer.
- Instance-element, the lowest element; the attributes show contents of access commands to control the accelerator components, and they designate various control parameters needed to execute the commands. The attributes also specify a relay address, a data type, and a bit length of data in the case of PLCs.

The lower elements inherit attributes of upper elements.

An example of the XML document generated by driver builder is shown in Figure 2.

2.2 Driver builder

The driver builder that generates the XML document consists of a main program to designate a particular configuration of the accelerator components and two sub programs to designate the interface specifications and control commands for PLC and GPIB devices respectively.

When the driver builder is executed, the main program loads ClassInfo.xml, which describes all of the accelerator components, and displays the configuration of the components on a tree-view of a main window. Properties of a selected component on the tree-view are shown on a property list of the window. The component's properties are, for example, current, voltage, ON/OFF status, and so

```
<?xml version="1.0" encoding="Shift_JIS"?>
-<Driver>
 -<Interface DriverClass="GPIB">
  -<Instrument InstrumentID="HP1" BoardNo="0" GpibAddress="2" Initialize="*RST;:INIT:CONT ON">
   -<Node NodeID="ND01" DeviceClass="Linac.Gun" NodeName="Grid">
      <Instance InstanceID="INS01" Property="VoltRef" AccessMode="Write" PropertyValue="" DeadBand=""
        Command="VOLT ###"/>
      <Instance InstanceID="INS02" Property="Voltage" AccessMode="Read" PropertyValue="" DeadBand="5.0"
        Command="MEAS:VOLT?"/>
      <Instance InstanceID="INS03" Property="HvOnOff" AccessMode="Write" PropertyValue="HvOn"
        DeadBand="" Command="OUTP ON"/>
      <Instance InstanceID="*****" Property="HvOnOff" AccessMode="Write" PropertyValue="HvOff"
        DeadBand="" Command="OUTP OFF"/>
    </Node>
   </Instrument>
  </Interface>
 -<Interface DriverClass="PLC">
  -<Instrument InstrumentID="YEW1" IpAddress="130.34.61.97" PLCType="YOK.FA.E">
   -<Node NodeID="ND01" DeviceClass="Linac.Magnet.Q" NodeName="Q1">
      <Instance InstanceID="INS01" Property="CurrRef" AccessMode="Write" RelyAddress="D00004"
        DataType="Binary+Sign" BitLength="16" PropertyValue="" RelyValue="" DeadBand="" RapidTime=""
        RapidValue=""/>
      <Instance InstanceID="INS02" Property="Current" AccessMode="Read" RelyAddress="D00001"
        DataType="Binary+Sign" BitLength="16" PropertyValue="" RelyValue="" DeadBand="0.05"
        RapidTime="1000" RapidValue="0.1"/>
      <Instance InstanceID="INS03" Property="Output" AccessMode="Write" RelyAddress="I00001"
        DataType="Bit" BitLength="1" PropertyValue="On" RelyValue="$True" DeadBand="" RapidTime=""
        RapidValue=""/>
      <Instance InstanceID="*****" Property="Output" AccessMode="Write"
        RelyAddress="I00001" DataType="Bit" BitLength="1" PropertyValue="Off" RelyValue="$False"
        DeadBand="" RapidTime="" RapidValue=""/>
    </Node>
   </Instrument>
  </Interface>
 </Driver>
```

Figure 2: An example of the XML document (DriverInfo.xml).

Table 2: Method list of PLC and GPIB class.

Class	Method	Parameter	Definition
Plc	Initialize	Nothing	Initialize PLC
	ToPlc	Property value as variant	Transfer command to PLC
	ToCoack	Changed operation value as variant	Transfer changed operation value to COACK
Gpib	ToGpib	Property value as variant	Transfer command to GPIB
	ToCoack	Nothing	Read operation value and transfer it to COACK

on. In the next step, either PLC or GPIB is selected as the interface correspondent to each property. Then the display window is switched to sub windows for either PLC or GPIB devices. In the sub windows, the attributes concerning the Instrument-element and Instance-element in Table 1 can be entered. Attributes entered in advance are inherited by means of designating an InstrumentID for the Instrument-element and/or for the NodeID of Node-element. Finally, the attributes of the Instance-element are entered, and the XML document named "DriverInfo.xml" is generated. Anyone can design an XML document using the driver builder tool; knowledge of XML grammar or XML editor is not required.

2.3 IOC

IOC functions as the manager in the device layer, controlling accelerator components by following commands from COACK. As IOC starts up, DriverInfo.xml is loaded, and driver objects equivalent to each Instance-element are created from each driver class, the PLC or GPIB class. The attributes of each element correspond to the properties of each object. Each driver object has methods, "ToPlc" and "ToGpib", to execute the commands for the accelerator components. "ToCoack" is the method for transferring the operation status from the accelerator components to COACK. The differences between the interfaces, the differences between the accelerator components and the differences between the control protocols can be assimilated by the properties of each driver object. Table 2 shows methods of the PLC and GPIB classes.

The allocations of the driver objects and data flows in IOC are shown in Figure 3. Since COACK is component-oriented software, COM/DCOM is used to communicate between the components. The commands sent from COACK are received as DCOM events in IOC. The received command is checked to determine the target driver object, and the command is delivered to ToPlc or ToGpib of the target object. Concerning a "read" operation on PLC, if an operation value of the accelerator components exceeds a permissible range set up in advance, PLC generates to IOC an event including information such as an identification code on the event

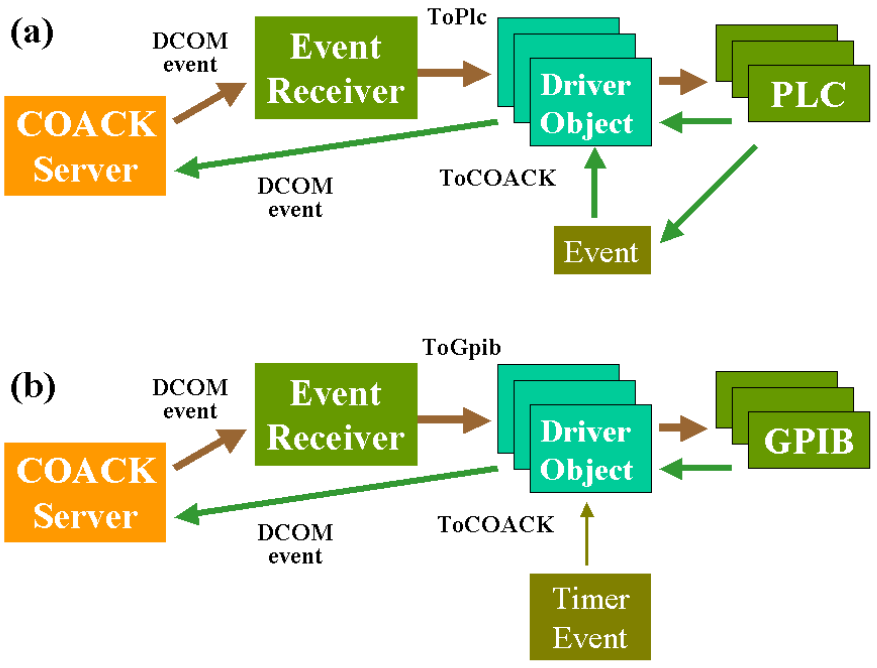

Figure 3: Data flow in the device layer; (a) is for PLC, (b) is for GPIB

source and the changed operation value. Then IOC sends the event information to ToCoack of the related driver object, and the operation value of the accelerator is transferred to COACK. Concerning GPIB operation, a “write” operation command received from COACK is processed the same as with PLC, but a timer event is used instead of an event from the interface in the “read” operation. The GPIB driver object is driven by timer events generated at intervals of designated time; its ToCoack method reads the operation value of the accelerator components. The ToCoack method of GPIB driver object then compares the new operation value with the last one, and if there is a discrepancy between the two values, the new operation value is transferred to COACK.

A product named “Fa-Engine”[4] offered by Robotic Ware Company is used to communicate between IOC and PLC. The product is an ActiveX component for Visual Basic. The library, NI-488.2M[5], offered by National Instrument is also used for GPIB communication.

3 CONCLUSION

Our generalized device layer was developed using an XML document, which makes it possible to construct an accelerator control system with high flexibility and expandability. The developed device layer will be tested soon at the 300MeV electron linear accelerator control system using COACK[3] at Tohoku University. After running a trial, the device layer will be improved, after which it will be registered in the COACK library and opened to the public.

As a next step, we plan to add an alarm monitor function to the device layer in order to enhance the power of COACK system. We also plan to add special software components to process particular control functions for the accelerator.

In a final note, we wish to express sorrow over the sudden passing of Isamu Abe on June 2nd of this year. As a project leader, he made a great contribution to the development of COACK. We have overcome the sadness of his death, in part by continuing to grow COACK. May his soul rest in peace.

4 REFERENCES

[1] I. Abe, et al., “Project on PC based accelerator control kernel (COACK-II),” ICALEPS 1999, Trieste, Italy

[2] I.Abe, et al., “Recent Status of COACK and It’s Function,” PCaPAC 2000, Hamburg, Germany

[3] Y.Shibasaki, et al., “New Control System for the LNS Linac,” PCaPAC 2002, Frascati, Italy

[4] “Industrial Automation Tool FA-Engine 3.5 User’s Manual,” RoboticsWare

[5] “NI-488.2M^{TM} User Manual for Windows,” NATIONAL INSTRUMENTS