Middleware Architecture for Open Control Systems in the Distributed Computing Environment

Wongoo Lee and Jaehyun Park

Abstract: The advance of computer, network, and Internet technology enables the control systems to process the massive data in the distributed computing environments. To implement and maintain the software in distributed environment, the component-based methodology is widely used. This paper proposes the middleware architecture for the distributed computer control system. With the proposed middleware services, it is relatively easy to maintain compatibility between products and to implement a portable control application. To achieve the compatibility between heterogeneous systems, the proposed architecture provides the communication protocols based on the XML with lightweight event-based service.

Keywords: distributed control system, real-time, XML, open architecture

I. Introduction

The improvement of computer and network technology enables the control systems to handle massive information in the distributed computing environments like other computing systems[1]. To develop such a distributed control system, a middleware that provides various services such as ORB (Object Request Broker) or event services is required. Using a middleware reduces the development cost and provides the compatibility between the heterogeneous control systems[2]. There are some well-known commercial middlewares currently available: CORBA (Common Object Request Broker Architecture) from OMG (Object Management Group), EJB (Enterprise JavaBeans) from Sun Microsystems, and COM (Component Object Model) from Microsoft[8][9]. Even this kind of the ordinary commercial middleware provides various services to developers, it is still difficult to write domain-specific applications such as distributed control and monitoring applications because they are originally designed for the general distributed applications. With these basic middleware services only, it is really hard to maintain the compatibility between the control applications from different vendors.

To achieve the openness within the control domain, there are a couple of consortiums in work. OPC (OLE for Process Control) is one of those consortiums to use COM-based middleware for the control domain [3]. It, however, defines only interfaces between the components rather than provides the runtime services [4]-[7]. Although OPC is an example for keeping openness and compatibility, lots of efforts are required for implementation and maintenance of a control application with it.

This paper proposes a middleware for the control systems based on the COM architecture provided by Microsoft Windows. It provides the specification of the control equipments, the distributed event service, the communication message format, and the communication protocol. This paper also demonstrates an implementation example to visualize the openness and flexibility of the proposed middleware. The target domain used in this paper is the chip-manufacturing domain based on the SEMI's GEM protocol [10].

Section 2 introduces the middleware for components and communications. Section 3 explains the services for the control domain and the configuration of whole system. And the implementation of the prototype for GEM protocol is in section 4. Finally, a conclusion is remarked in section 5.

Manuscript received: Nov. 18, 2000., Accepted: Mar. 14, 2001.
Wongoo Lee: Automation Engineering, Inha University.(iwongn@resl.inha.ac.kr)
Jaehyun Park: School of information and communication, Inha University.(jhyun@inha.ac.kr)
※The work reported in this paper is partially supported by Inha University under grant 1999-20695.

II. Underlying Technologies

Before describing the middleware proposed in this paper, the underlying technologies are introduced in this section.

1. Component-base architecture

Although object-oriented methodology makes it possible to design and implement the huge and complex systems, it also has some problems such as code exposure, versioning difficulty, and deploying. Furthermore, its white-box model inherently lacks compatibility and reusability. To overcome these shortcomings of the object-oriented method, the component-based methodology that uses the black-box model is widely used to isolate interface from implementation. The framework for the middleware also moves from the object-oriented to the component-based methodology [13].

To develop a distributed system in the past, the lower layer API's such as socket or RPC have been widely used. But these procedural methods require more time and cost to develop and maintain than the object-oriented methods. Since most of the component models support RMI (Remote Method Invocation), the component itself can be distributed over the network environment. This means that developers can write distributed control applications more easily using a component-based middleware. Another trend in designing a modern middleware is to support T/P monitor feature, often called as CTM (Component Transaction Manager) that provides load balancing, resource management, message queuing, and event service [12].

2. XML (eXtended Markup Language)

XML is a meta-language to describe other languages. Since the underlying philosophy of XML aims the flexibility and portability, it can be used in any platform. Thanks to its portabiliby, the platform-independent RMI such as SOAP (Simple Object Access Protocol) appears recently. Most of modern platforms provide the DOM (Document Object Model) parser

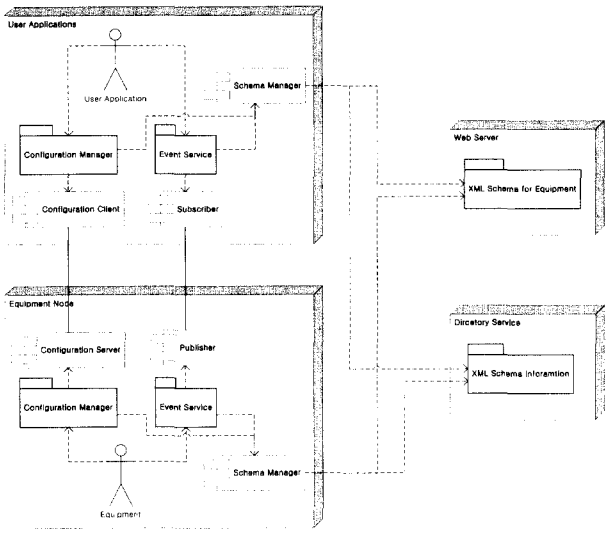


Fig. 1. Deployment diagram for system configuration.

and SAX (Simple API for XML) engine[14,15]. In addition, XML is adequate to model a hierarchical data structure.

3. UML (Unified Modeling Language)

This paper contains various diagrams in UML (Unified Modeling Language), the standard object-oriented modeling language accepted by OMG in 1997. The UML is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system[11]. It especially is useful to model a distributed system that has many components and protocol scenarios. *Use Case diagram*, *Class diagram*, *Component diagram*, *Sequence diagram,* and *Deployment diagram* are used for visualizing the proposed middleware.

III. Middleware Architecture

1. Structure of proposed middleware

Fig. 1 shows the global structure of the distributed control environment and proposed middleware in it. It shows three major groups: (1) server node, (2) client node, and (3) global service node. The server node produces data for control or monitoring and the client node consumes the produced data. For example, in a distributed monitoring system, a sensory device is a server node because it produces a measured data and the monitoring computer is the client node. Both nodes have the same middleware components: *event service, schema manager and configuration manager.* Besides these common components, there are symmetric server/client components: configuration server/client and data publisher/ subscriber. Two global service nodes, web server and directory server, are also included for providing XML schema for equipment and its information.

Data exchange between the client and server nodes goes through the publisher and subscriber components. However, before real data exchange, client and server nodes should negotiate their configuration through configuration manager after establishing a session between them. To establish a session, the applications must know the counterpart's physical location. In the proposed middleware architecture, the global service nodes are responsible for session management and the mapping between logical name and physical location. In addition to session establishing services, it provides an event service that is a useful data exchange method in control applications. Following subsections described each module and services of the proposed middleware in detail

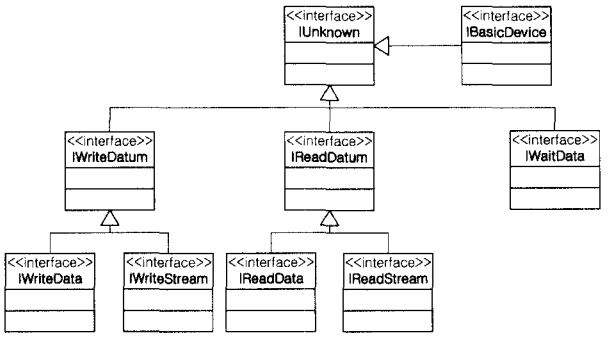


Fig. 2. Class diagram of data read/write function.

2. Equipment abstraction

An important requirement for the control systems is the precise specifications to describe the characteristics of the equipments. It is, however, very difficult to model all the equipment functions under single interface. In the proposed middleware, instead of modeling all equipments with single interface, each function of equipment is modeled with a different interface, and, in turn, equipment is modeled as a group of functions. In the proposed middleware, *IBasicDevice* is only one mandatory interface to support common features of all equipment; (1) initialization, (2) configuration, and (3) acquisition of component information. Because querying all interfaces to check whether a component supports the specific interface or not is very inefficient, a component category is used in this paper. Component category is a standard mechanism to advertise interfaces that is available in a component under COM environment. A specific equipment component just registers it interface to specific category and the middleware takes care of the rest. Fig. 2 shows a sample interface hierarchy for the read/write functions. An application program can use these functions by a query mechanism.

3. Data format

Two kinds of data format should be defined. The first one is the format for transferring data packet, and the second one is the configuration format of the equipment. Since the first is likely to be transferred in network frequently, the overhead of caused by XML is not negligible. To reduce this overhead, binary-encoded format is used. *XML Information Set* (Infoset) is defined for this purpose. Because XML Infoset is not supported, however, by all of the platforms yet, the proposed middleware implements a simple interface to convert the ordinary XML to binary format. Fig. 4 shows Deployment diagram on this conversion. Besides this conversion interface, the middleware must identify the data type and their length in the binary packet. For this, the same specification in XML Infoset is used. The specification defines almost all of the types used in programming language such as C, C++. By using XML

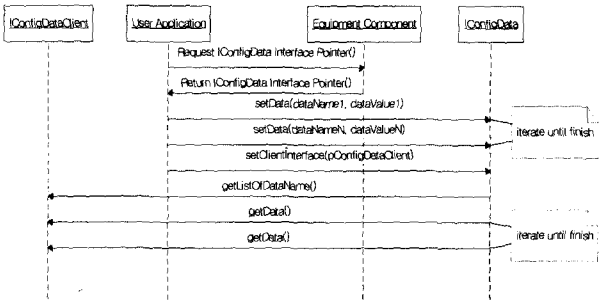


Fig. 3. Sequence diagram for component configuration.

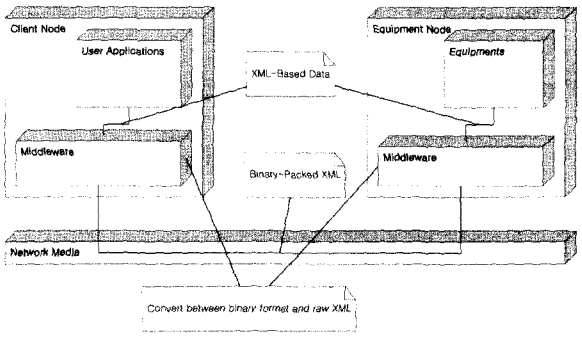


Fig. 4. Conversion between binary format and raw XML.

specification, the system can manage data with standardized manner for the complex data format such as RPTID message of GEM protocol (described later).

4. Dynamic configuration

The equipment component should be able to configure itself dynamically. There are two kinds of configuration data: the equipment global data and the client-specific data. The equipment global data can be stored in the same location of the component. But to store the client-specific data in the same location is very inefficient and difficult to implement from equipment's view. The client-specific configuration data must be stored in custom location by client application. To support this configuration scenario, the middleware defines two standard interfaces and their sequences as shown in Fig. 3. If there is no client-specific data, the *setClientInterface* call and

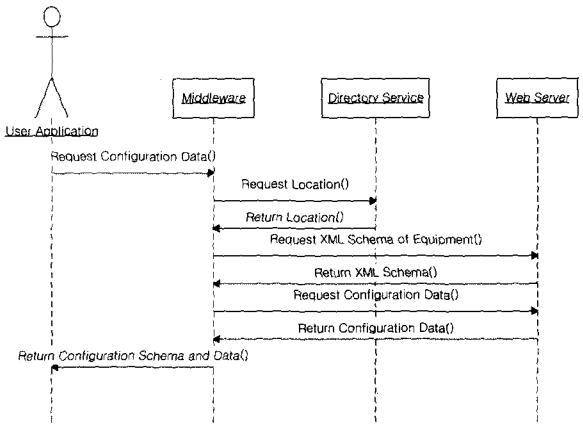


Fig. 5. Sequence diagram for configuration data.

subsequent sequences can be omitted. When user application calls setClientInterface, it passes the pointer of *IConfigData-Client* interface. With this interface, equipment component can configure itself with user application specific data.

Table 1. Properties for the XML Schema.

Name	Type	Description
Name	String	The name of Equipment
UniqueID	GUID	Globally unique ID
Location	URL	The location of XML Schema

Since configuration data is transferred only during the initialization or modification period and it requires very complex structure, usually in hierarchy structure, XML is an effective scheme for this purpose. To use XML with complex structure, XML Schema must be defined to advertise the configuration data of equipment at runtime. To store schemas in the public place, directory service, Active Directory in our case, is used. It makes easy to implement the stable and public storage through auto replication and its accessibility. Table 1 shows schema properties that are designed for equipments within a directory service. As shown in table 1, because the location type is an URL, it can be viewed XML browser directly. It means client application can be written based on commercial web-browser. Fig. 5 depicts the sequence that user application gets configuration schema data.

Fig. 6. Mandatory elements of configuration data.

Although the mandatory properties exist in configuration data that can be acquired with standard manner, the data can be different to the equipments. In spite of these diversity of configuration data, XML can be parsed with standard DOM parser or SAX engine that is provided underlying platform. Fig. 6 shows mandatory properties in equipment configuration data.

5. Data acquisition

There are two basic interfaces for data acquisition, *IRead-Data* and *IWaitData*. They are used for periodical querying and notification from equipment respectively. Even these two are mandatory interfaces, not all equipments support both read and write functions. For example, sensors have only IWaitData interface. For this kind of equipments, the proposed middleware provides default implementation of the connection com-

ponents. Fig. 7 shows a sample connection component and underlying operation. It uses aggregation mechanism of COM. Fig. 7 shows only IWaitData-to-IReadData component, but IReadData-to-IWaitData component can be defined in the reverse sequence.

Event service is a mechanism for the publishers to notify subscribers of their data without any queries from subscribers, which is used to implement IWaitData interface. COM technology provides two mechanisms for event services; *Connection Point* and *COM+ Event Service*. Connection Point is not suitable to be used out of machine boundary because it is originally designed to be used within single process by standard *Automation* interface. And it requires the overlapping of execution between publisher and subscriber. Since it, however, supports Automation mechanism, it can be used by script environment such as VBScript and ASP. COM+ event Service provides useful services such as transaction processing and reliable queuing. But it may impose large overhead to system.

With considering these natures of event services, this paper proposes a simple event service that has lightweight overhead and supports script environment. It uses the TCP-based proprietary protocol, named MS2DC (Monitoring Server-To-Data Collector) across the machine boundary and Connection Point and windows messages within machine boundary. The TCP-based protocol makes the whole middleware portable onto the various RTOS (Real-time Operating System). Fig. 8 shows the activity diagram of three threads at equipment side: (a) is the implementation of IreadData, (b) is the IWaitData, and (c) is a garbage collection algorithm.

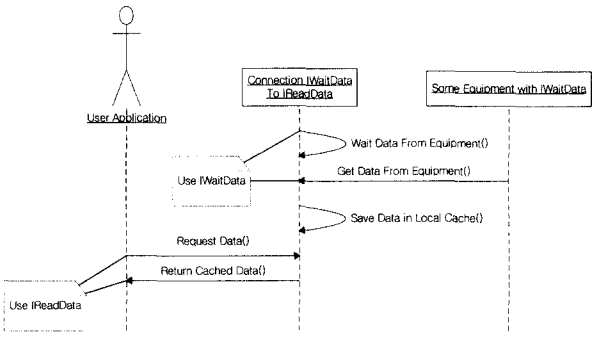
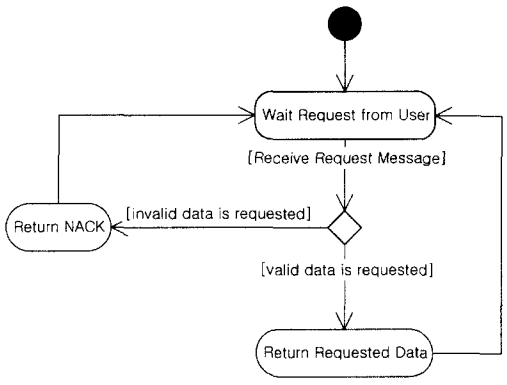
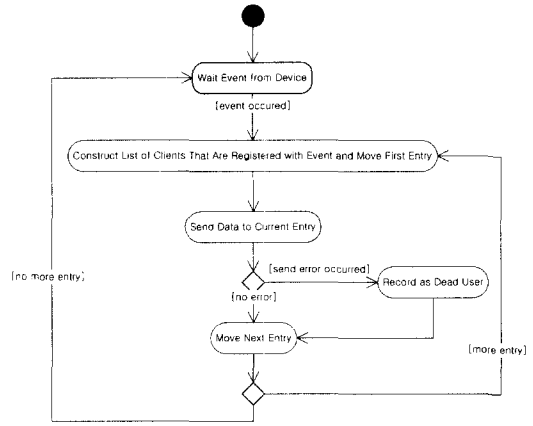


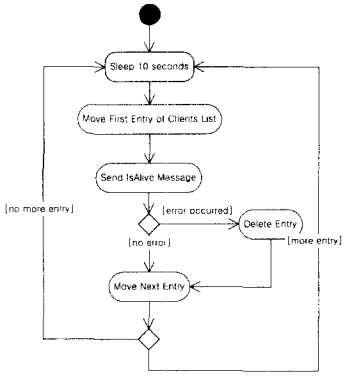
Fig. 7. Sequence diagram for operation of connection component.



(a) Response to data request messages



(b) Publish data that come from device



(c) Detect dead client and delete

Fig. 8. Activity diagram for equipment side of MS2DC Protocol.

At the user side, the application program can query data and register itself as a subscriber for a specific data. When the middleware implemented as a local daemon receives this registration request, it searches the location of equipment and transfers the registration-request message to remote equipment node. When the local middleware receives the data from the equipment nodes, it hands over them to applications by Windows message. The middleware enables applications use the Connection Point mechanism rather than Window message for the pure object-oriented programming. To translate Windows message to the COM event, STA (Single Thread Apartment) is used. Fig. 9 describes components and their communication methods.

Table 2. The used SECS-II messages.

Name	Stream	Function	Direction
Are You There (AYT)	1	1	H↔E
Status Collection	1	3	H→E
Establish Communication	1	13	H↔E
Request Online	1	17	H→E
Equipment Constant	2	15	H→E
Enable Collection	2	37	H→E

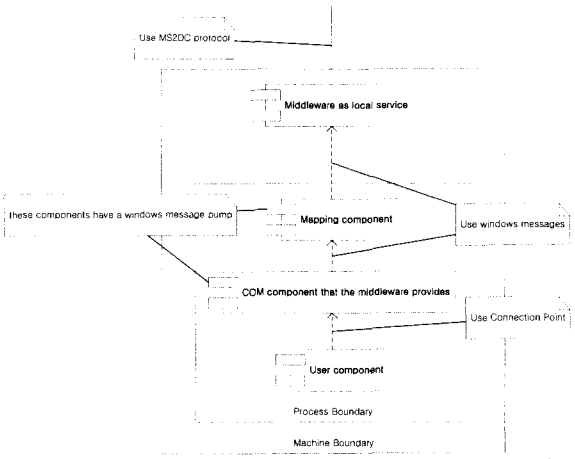


Fig. 9. Component diagram for message delivering and communication protocols.

IV. Implementation

To demonstrate the flexibility and portability of the proposed middleware architecture, the prototype middleware is implemented for the standard protocol in semiconductor industry, SECS-I, SECS-II, HSMS(High-Speed SECS Service), and GEM(Generic Model for Communications and Control of SEMI Equipment). Using the prototype middleware, simple control and monitoring application is developed for the commercial chip mounter (Samsung CP-40).

GEM is a standard protocol that is used for general modeling of semiconductor equipments and SECS-II is communication data format for GEM. SECS-I and HSMS are standard communication protocols based on RS-232 and TCP/IP respectively. Without any middleware, user application should contain a code to support GEM and SECS/HSMS., The applications coded with a commercial protocol stack of GEM and SECS/HSMS does not flexibility nor openness.

The prototype middleware is implemented using XML-based components, with which GEM data in a hierarchy are handled efficiently. This prototype system does not support all of the GEM/SECS-II commands. Table 2 summarizes the message types used in the prototype system. The GEM component uses CEID messages for event delivery, and it can read or write status variables by SVID messages. All the application can operate within a standard web browser because CEID, SVID and RPTID messages can be delivered using XML format on the top of implemented middleware.

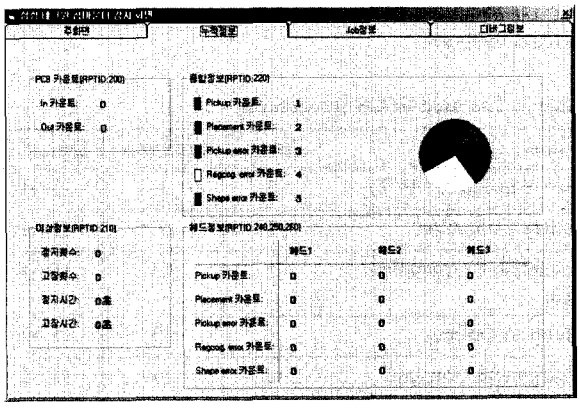


Fig. 10. Sample snapshot of the implemented system.

V. Conclusion

This paper proposes and implements a middleware that is designed for the distributed control domain. Using the proposed interfaces and runtime services increase the compatibility between the heterogeneous control applications from multiple vendors, and, as a result, managing the heterogeneous equipments can be standardized. The event services implemented in this paper provide easy and lightweight infrastructure for the event-driven control applications. Combining the proposed event services with real-time network with QoS (Quality of Service), developers are able to control and monitor equipments in real-time manner transparently. The proposed system provides XML and TCP/IP-based protocol for the communications between the different platforms as in Internet.

References

[1] Jerry Z. Gao, Cris Chen, Yasufumi Toyoshima, David K. Leung, "Engineering on the internet for global software production", *Computer*, vol. 32, no. 5, pp38-47, May, 1999.

[2] R.Orfali, D. Harkey and J. Edwards, "The essential client/server survaval guide", *John Wiley & Sons*, 1996.

[3] OPC Taskforce, "OPC Overview", http://www.opcfoundation.org, Oct. 1997.

[4] OPC Taskforce, "OPC Common Interface", *http://www.opcfoundation.org*, Oct. 1997.

[5] OPC Taskforce, "OPC data access custom interface specification 2.04", *http://www.opcfoundation.org*, Oct. 1997.

[6] OPC Taskforce, "OPC alarms and events custom interface specification", *http://www.opcfoundation.org*, Oct. 1997.

[7] OPC Taskforce, "OPC batch custom interface specification 1.0", *http://www.opcfoundation.org*, Oct. 1997.

[8] Robert Orfali, Dan Harkey, Jeri Edwards, "Instant CORBA", *John Wiley & Sons*, 1998.

[9] Richard Monson-Haefel, "Enterprise JavaBeans", *O'Reilly & Associates, Inc.*, 1998.

[10] SEMI, "SEMI E30-0299 Generic Model for Communications and Control of SEMI Equipment (GEM)", *http://dom.semi.org*, Feb. 1999.

[11] Grady Booch, James Rumbaugh, Ivar Jacobson, "The unified modeling language user guide", *Addison Wesley*, 1999.

[12] Y. P. Shan and R. H. Earle, "Enterprise computing with objects from client/server to the internet", *Addison Wesley*, Dec. 1997.

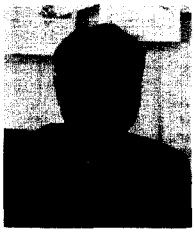
[13] Peter M. Maurer, "Components: what if they gave a revolution and nobody came?", *Computer*, vol. 3, no.6, pp28-34, Jun. 2000.

[14] Don Box, "Essential XML", *Addison Wesley*, 2000.

[15] Martin Naughton, "The joy of SAX", *http://msdn. Microsoft.com/workshop/xml/articles/joyofsax.asp*, Jun. 2000

Wongoo Lee

Received BS and MS degrees in automation engineering from Inha University. He is interested in open control architecture and distributed computer control software.



Jaehyun Park

He received BS, MS, and Ph.D degrees in control and instrumentation engineering from Seoul National University. He works for Inha University since 1995. His research area includes realtime computing, network, embedded systems, and open control architecture.