New Control System Aspects for Supporting Complex Data and High Performance System

Dae-Seung Yoo, Vu Van Tan, and Myeong-Jae Yi
School of Computer Engineering and IT, University of Ulsan,
San-29, Moogu 2 Dong, Namgu, Ulsan 680-749, Republic of Korea
{ooseyds, vvtan, ymj}@mail.ulsan.ac.kr

Received 27 June 2008; Accepted 29 August 2008

The data in automation and control systems can be achieved not only from different field devices but also from different OPC (OLE for Process Control) servers. However, current OPC clients can only read and decode the simple data from OPC servers. They will have some problems to acquire structured data and exchange the structured data. In addition to the large network control systems, the OPC clients can read, write, and subscribe to thousands of data points from/to OPC servers. Due to that, the most important factor for building a high performance and scalable industrial control system is the ability to transfer the process data between server and client in the most efficient and fastest way. In order to solve these problems, we propose a means to implement the OPC DA (Data Access) server supporting the OPC complex data, so that the OPC DA clients are able to read and decode any type of data from OPC servers. We also propose a method for caching the process data in large industrial control systems to overcome the limitation of performance of the pure OPC DA system. The performance analysis and discussion indicate that the proposed system has an acceptable performance and is feasible in order for applying to real-time industrial systems today.

Categories and Subject Descriptors: System and Architecture [**Distributed Systems**]

General Terms: Complex data, data caching, large control network, OPC data access

Additional Key Words and Phrases: High performance, industrial system, real-time

1. INTRODUCTION

In industrial systems the scalar data can present machine operating parameters from analogue measurements such as pressure, temperature, flow, level, and vibration, or discrete signals used to represent on/off state or abnormal condition [Holley 2004]. However, the OPC clients only might be able to read and decode the data as a simple data type. They have some problems to achieve the structured data and exchange the structured data between the collaborating applications. The OPC complex data specification was therefore proposed to overcome this limitation [Tan et al. 2006; TheOPCFoundation 2003]. The OPC complex data working group was making enhancements to the OPC DA specification based on requirements and feedback from other industrial groups to address additional types of data such as structures, arrays, binary, XML documents and dictionary. The OPC complex data provide a full way to read and decode any type of data from OPC servers. Actually, the OPC complex data are defined by OPC complex data items that are composed of a combination of the structured data, simple items, complex items, and so forth [TheOPCFoundation 2003; 2004]. The OPC Foundation[1] has provided some complex data implementation code example for developers and programmers in order to reduce the implementation cost and time on their system developments. Based on the OPC complex data specification, we propose an approach in order to implement the OPC DA products for supporting complex data structure.

The clients in the large network control can read, write, and subscribe to the thousands of data points from/to servers. Nowadays, one of the most popular standards is the OPC DA in real-time applications. OPC DA interfaces together with DCOM (Distributed Component Object Model) are used to easily establish a communication between network control system components running on various computer nodes. The OPC DA server exposes a set of OPC DA interfaces that have functions to *browse*, *read*, and *write* to variable values. But the OPC DA standard has a disadvantage that no mechanism for data compression is supported. This is one of the reasons, which may create performance problem for large data transfers in the control network [TheOPCFoundation 2003].

In use today the data caching is widely a classic and successful technique to improve the performance of software programs by exploiting temporal references and providing *high-bandwidth, low-latency access* to the data cache. In the client-server architecture model of software programs, the data caching was typically implemented on the server-side because it is simpler to implement and easier to synchronize and maintain. All programs running on the clients may equally use server-side cache whether they hopefully require a fast access to the process data. The main weakness of the server-side data caching is when the software programs running on the client require a large amount of the process data, the data transfer from server's cache to the clients may become a performance bottleneck as it is the case of the client-server applications using the OPC DA standard for large data transfers [Iwanitz and Lange 2006; Veryha 2005].

The client data caching has been widely used in DBMS (Database Management System) and Web applications [Cao and Liu 1998; Teng et al. 2005; Carpenter et al. 2001; Wang et al. 2006; Franklin 1996; Yang and Zhang 2001; Barish and Obracke 2000; Franklin et al. 2006]. [Cao and Liu 1998] compared three consistent approaches such as *adaptive time-to-live*, *polling-every-time*, and *invalidation* through their analysis and implementation. [Teng et al. 2005] proposed the Web Caching and Web Pre-fetching in the client-side proxies by using IWCP (Integration of Web

[1]http://opcfoundation.org/

Caching and Pre-fetching) algorithm. The data caching in client-server systems based on the data-shipping approach is represented [Franklin et al. 2006]. [Gopalan et al. 2002] proposed an approach to focus on the cache coherency that is ensured in cooperation with the server-side database agent where communication is established through the use of persistent socket connections between the client-side agents and the database agent. Changes to the database can be propagated to all client agents ensuring that only current information is stored in the client agents' caches.

[Veryha 2005] presented a solution to improve the performance of the pure OPC DA system by using cache technique. The data caching was implemented in the client-side in order for overcoming the limitation of the OPC DA performances. Unfortunately, the system performance related to the complex data has not evaluated. To overcome the limitation of the pure OPC DA performances, we also propose an approach to implement the data caching technique in both client-side and server-side. The proposed approach improves the limitation of performances of the pure OPC DA products, which are widely used for control and automation system, when OPC client requests a large amount of data points.

This paper is organized as follows: The next section reviews the OPC data access and complex data specifications, and provides problem statements under consideration of some related works. Section 3 introduces the design and implementation of the OPC DA system for supporting the OPC complex data. Section 4 presents a mechanism for embedding data caching into both OPC DA server and OPC DA client. Section 5 discusses the performance evaluation and analysis of the proposed system. They significantly demonstrate that the proposed system has an acceptable performance and is feasible to support real-time industrial systems. In summary, some conclusions and future work will be marked in Section 6.

2. BACKGROUND AND PROBLEM STATEMENTS

In order to provide the relevant OPC technologies and problem statements, this section first presents an overview of the OPC DA specification and the OPC complex data specification in Section 2.1 and Section 2.2, respectively. The problem statements under consideration of related works are then analyzed in Section 2.3.

2.1 OPC Data Access

Developed by an automation software and hardware vendor consortium, the *OPC Foundation*, OPC is the first automation-domain specific component standard for control systems. OPC standardizes the mechanism for communicating a numerous data sources, whether they are hardware I/O devices on the plant floor or databases in control rooms. OPC interfaces which are provided by OPC servers let any client access the server devices. In general, the OPC DA specification defines a set of standard COM objects, methods, and properties that specifically address interoperability requirements for the factory real-time automation, process control, and condition monitoring applications [TheOPCFoundation 2004].

In OPC client-server architecture model, server applications acquire, contain, and serve data to client applications effectively. The OPC servers provide a standard interface to OPC COM objects, letting OPC clients exchange data and control command in a generic way. The OPC client also can communicate with one or more OPC servers from different suppliers. Thus the OPC clients can access to the data in the same way, whether data are coming from an OPC server connected to a PLC (Programmable Logic Control system); industrial networks such as FOUNDATION Fieldbus, PROFIBUS, or DeviceNet; SCADA (Supervisory, Control, and Data Acquisition system); a product management system, and so forth based on DCOM leverages for permitting client-server applications to access plant floor data via an Ethernet network.

The operations for accessing items' values are *Read* and *Write*. Both operations allow to access several items with a single call. The OPC operations *Browse* and *GetProperties* are used to query OPC items which are available and to get the properties of the OPC items. Browse allows querying an OPC item's immediate successors including filtering and can return property values of the item found. Property values can also be retrieved with *GetProperties*. The operation *GetStatus* is used to retrieve the status of the OPC server for the OPC clients.

Several communication mechanisms are used to communicate between the OPC client and the OPC server such as *synchronous calls*, *asynchronous calls*, *refresh*, and *subscription*. The refresh and subscription are *callback* mechanism used to access predefined sets of data points on the plant floor. The communication between the OPC DA components and the behavior of the OPC server can be controlled by the OPC clients.

2.2 OPC Complex Data

The OPC complex data initiative will provide a way for the OPC client application to read and decode any type of data from OPC servers. Complex data mean an OPC DA item that has a defined structure. The OPC DA item includes read-only information, run-time status, and writable control points. The OPC complex data contains OPC complex data items [TheOPCFoundation 2003].

The OPC complex data items can include non-structured items, abstract elements, abstract items, structured items, strings, integers, sequences of bytes, XML data, OPC binary and so forth. Each data item is accompanied by a Data Type Description that defines the structure of the item and a Dictionary described each of the Data Types. The dictionary contains all information that the OPC client needs to be recognized the complex data item it is receiving.

The OPC complex data specification defined two type systems that provide the level of capability, *XML schema* and *OPC binary*. The XML schema describes complex data values that are presented in XML. The OPC binary which is used to present complex binary values has defined the format of OPC binary dictionaries.

2.3 Problem Statements

The data in the industrial systems can be normally acquired not only from different field systems and devices, but also from different OPC servers. Currently, OPC clients can only read and decode the data as a simple data type. However, the data from the hardware I/O devices are complex data such as pressure, temperature, flow, level, and vibration or discrete signals, and so forth [Tan et al. 2006; TheOPC-Foundation 2003; Iwanitz and Lange 2006]. In addition to the automation and control systems the OPC clients can read, write, and subscribe to thousands of data points from various I/O devices through the OPC servers. For a graphical presentation of the large data mounts of the process data on the OPC clients, a large data transfer from the OPC server to the OPC clients may consume a large amount of the network bandwidth and operating system resources. Due to these requirements, the additional improvements for large network control systems using the OPC DA standard are required to reach an acceptable performance.

There are a number of researches focused on COM technology^2 to demonstrate that the COM/DCOM model can guarantee real-time communication to industrial control systems [Chen et al. 1999; Liu et al. 2005; Kew and Dwolatzky 2001; Fischer 1998]. [Veryha 2005] has suggested the way of using the data caching on the OPC DA client for improving the performance of the pure OPC DA system. The client data cache using proprietary socket-based connectivity implemented in C++ is able to subscribe to server process data. The process data from the *Data Access Server* are stored in the client data cache as variables. However, the author has not investigated the OPC DA performances in which the OPC DA server can permit the OPC clients to read and decode the complex data from the hardware I/O devices or OPC servers.

[Kaghazchi et al. 2007] developed an OPC DA server for diagnostics tool capable of diagnosing multi-field bus. The implemented approach provides a more general model for the development of relevant OPC servers for each underlying field bus network. But the complex data are not included. [Line et al. 2008] have studied the test of OPC standard as part of process control systems. However, the OPC server implementation was developed without supporting the complex data.

[Liu et al. 2005] proposed an open, nonproprietary, plug-and-play system for real-time process monitoring and control based on the OPC DA standards. This system provides a method for individual process monitoring and process control software to interact and share the data. Nevertheless, for the large control system the system performance is not effective because of not using data cache.

[Chilingaryan and Eppler 2005] have developed a high speed protocol used for the communication between the OPC XML-DA server and its clients with the guarantee of the complex data. The binary data format was used in the messages in order to reduce the required size of the message and to ensure high data rate. The performance of the protocol is not provided due to only some parts of the system being developed.

Up to date, many software companies have developed their OPC DA servers with supporting the OPC complex data such as [TheAdvosolInc 2008; ICONICSInc 2008; Technosoftware 2008; SoftwareToolbox 2008] and so forth. Unfortunately, the technical documents related to their systems are not provided because of secret.

In order to solve the problems mentioned above, the design and implementation of OPC DA system to allow that the OPC client can read and write any type of data from/to OPC servers or the hardware I/O devices are introduced. The proposed system also guarantees the ability of supporting high performance using data caching technology. The following issues will be discussed and solved.

(1) *Design and implementation of the OPC DA server in guarantee of the complex data.* Both the design of system aspects and the design of Complex Data Modules for supporting the OPC complex data are presented.
(2) *Data caching implementation for both the OPC server and OPC client.* A popular and effective approach to caching the process data in a large control network in order to improve the performance of the OPC DA system when applying to the process monitoring and control systems is developed.

3. DESIGN AND IMPLEMENTATION

This section presents a mechanism to implement the OPC DA server for supporting the complex data. The designs of system aspects and modules for developing and implementing the proposed system are represented in Section 3.1 and Section 3.2, respectively. The evaluation of the OPC complex data specification to specify its limitation is discussed in Section 3.3.

3.1 The Design of System Aspect

As aforementioned the OPC complex data contains OPC complex data items and can be described as a dictionary. This dictionary defines any type of data from the hardware I/O devices. When the OPC clients either read or write the data from/to the Data Access Server[3], the *Complex Data Modules* will detect and convert the data with the data type and the output format correlatively. The design and implementation

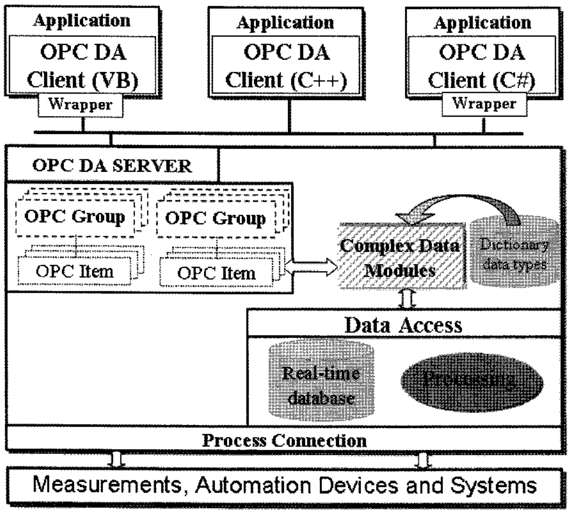


Figure 1. The design and implementation of OPC Complex Data Modules for the OPC DA server.

[2]The family of COM technologies includes COM+, Distributed COM, and ActiveX Controls.
[3]This term used in this paper is the same meaning with OPC DA Server.

of the OPC DA server to guarantee the complex data are proposed as shown in Figure 1.

The *Data Access Server* is notified and received the data from measurements, automation devices and systems, i.e., hardware I/O devices. After that the data are recognized and analyzed by the Complex Data Modules. Thus the data are then stored in an OPC item as a complex data item. Each OPC item contains the access path, complex data value, timestamp, etc. The OPC group is a collection of the OPC items that allow the OPC clients to access for reading or writing the process data. A respective OPC complex data item or a number of the OPC complex data items are accessed to read or write by the OPC clients that are interested in these OPC items.

In order to realize to the design strategy, when the hardware I/O devices generate the data, the data are processed by the Complex Data Modules with the Data Access Server and then are stored in the OPC item, including *access path*, *complex data value*, *timestamp* and so forth. The Complex Data Modules can support to recognize and represent both the XML data and OPC binary data. Hence, the OPC DA clients can understand and decode any type of data from the OPC servers or hardware I/O devices on the plant floor.

3.2 The Design of Complex Data Modules

The design of the Complex Data Modules that must guarantee to read and write both system types like *OPC binary* and *XML schema* is presented. The *OPCComplex-Data* class, which provides a means and methods for processing the complex data, aggregates two classes such as *OPCBinary* class and *OPCXMLSchema* class as shown in Figure 2. Both classes have defined attributes and operations used for initializing the Binary Dictionary Definitions and the XML Dictionary Definitions. The *OPCDA-*

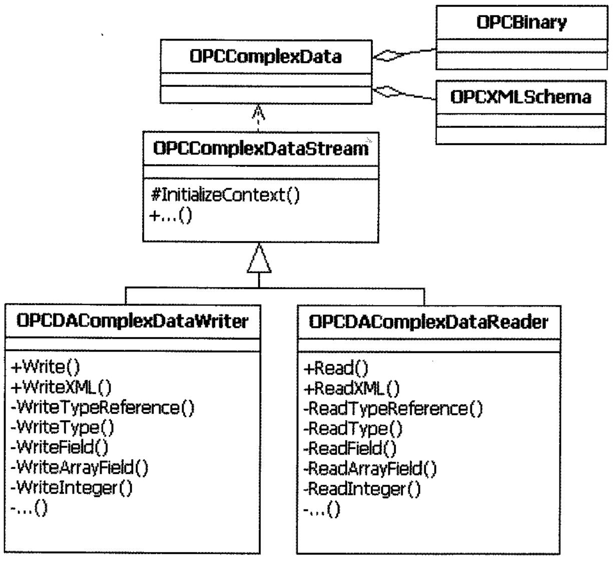


Figure 2. The class diagram of components of Complex Data Modules.

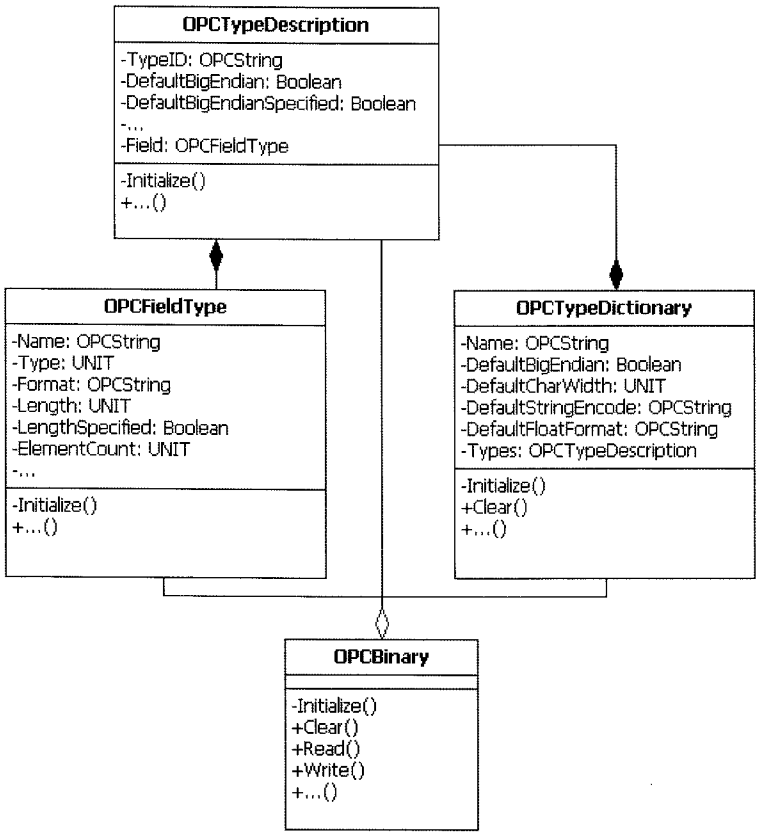


Figure 3. The class diagram of components of the OPCBinary class.

ComplexDataStream class initializes the serialization context for reading or writing the complex data item from/to buffer. Two derived classes from the *OPCDAComplex-DataStream* class are *OPCDAComplexDataWriter* class and *OPCDAComplexData-Reader* class. The *OPCDAComplexDataWriter* class provides functions for writing the complex data to the buffer. The *Write* and *WriteXML* functions are public functions used to write the OPC binary data or the OPC XML data according to the functionality. These two functions use some private functions such as *WriteType*, *WriteField*, *WriteArrayField*, etc. To illustrate the implementation of the proposed module for application developers and programmers, the function *Write* can be implemented in VC++ as the following fragment code.

```
Bool OPCDAComplexDataWriter::Write(
     OPCXMLType&  cValue,
     OPCTypeDictionary*  pDictionary,
     const OPCString&  cTypeID,
     BYTE**  ppBuffer,
     UINT*  pBufSize)
{
    //initialize the context in below command
    OPCContext cContext;
    bool bResult = InitializeContext(pDictionary, cTypeID, cContext);
    if(!bResult){
       return bResult;
    }
    UINT uBytesRequired = 0;
    bResult = WriteType(cContext, cValue, uBytesRequired);
    if((!bResult)||(uBytesRequired ==0)){
       return false;
    }
    //allow buffer
    cContext.Buffer = allow(BYTE, uBytesRequired);
    cContext.BufSize = uBytesRequired;

    //set buffer to zero
    memset(cContext.Buffer, 0, cContext.BufSize);
    UINT uBytesWritten = 0;
    bResult = WriteType(cContext, cValue, uBytesWritten);

    if((!bResult)||(uBytesWritten!= uBytesRequired){
       //free the buffer
       CoTaskMemFree(cContext.Buffer);
       return false;
    }
    *ppBuffer = cContext.Buffer;
    *pBufSize = cContext.BufSize;
    return true; //successfully write
}
```

The *OPCContext* in the fragment code above is a structure used for storing the current serialization context. The *OPCDAComplexDataReader* class provides the functions for reading the complex data from the buffer. Two main functions such as *Read* and *ReadXML* are used in this class. The *Read* function is used to read the complex data from the buffer and returns the data as binary data. The *ReadXML* function is used to read the complex data from the buffer and returns the results as XML data. To describe the components of the *OPCComplexData* class, an overview of these classes is then exposed. The *OPCDABinary* class is aggregated with three classes, *OPCFieldType* class, *OPCTypeDescription* class, and *OPCTypeDictionary* class as shown in Figure 3. With the *OPCFieldType* class, it is composed of the definitions of a field within a type description. The *OPCTypeDescription* class contains the description of a type within a dictionary. Finally, the *OPCTypeDictionary* class consists of a set of the complex type descriptions describing the complex data types. In the *OPCBinary* class, it contains the functions such as *Initialize*, *Read*, *Write*, and so forth which are used for initializing, reading, and writing the complex data from/ to the buffer, respectively.

The *OPCXMLSchema* class is developed, as Figure 2 shows, to provide a mechanism to present data in the XML data formats. It aggregates *OPCXMLElement* class and

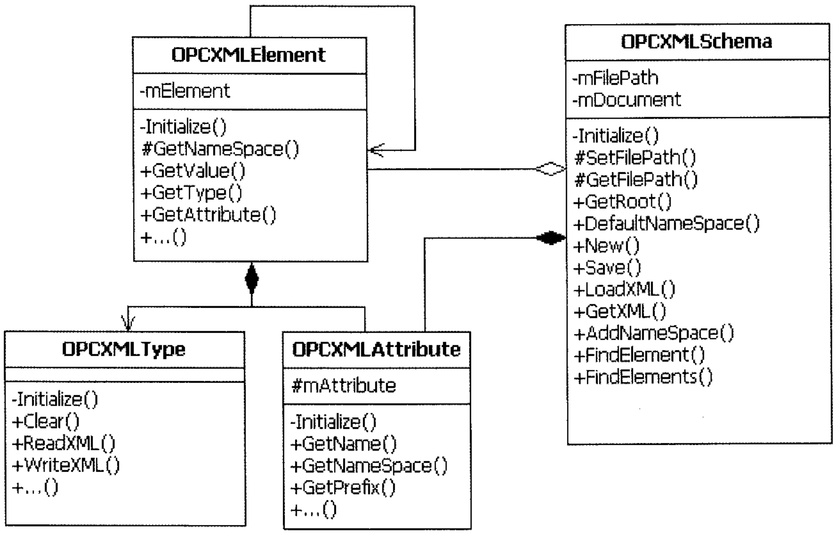


Figure 4. The class diagram of components of the OPCXMLSchema class.

also contains *OPCXMLAttribute* class. The *OPCXMLSchema* class consists of the needful functions that specifically support to perform and process the XML data. The *OPCXMLElement* class is composed of two classes as *OPCXML-Type* class and *OPCXMLAttribute* class. The *OPCXMLType* class has declarations for XML constants, types, and functions. There are two important functions in this class like *ReadXML* and *WriteXML*. The *ReadXML* function reads an object from an XML element or attribute. And the *WriteXML* function is used to write an object to an XML element or attribute. The *OPCXMLAttribute* class is to use for representing an XML attribute. Another is the *OPCXMLElement* class which facilitates manipulation of XML elements. It has the functions to perform processing of XML elements of the XML data. The *OPCXMLSchema* class is designed as shown in Figure 4. The design of the Complex Data Modules implemented into the OPC DA server has been represented. As a result, the OPC clients can read and decode any type of data from the OPC servers and the hardware I/O devices on the plant floor.

3.3 The Evaluation of the OPC Complex Data Specification

The proposed system is designed and developed according to the design strategy of supporting the complex data and high performance, respectively. According to the OPC complex data specification, it seems that this specification has some limitation for the implementation of an application. A number of evaluations of the OPC complex data specification are discussed. First, the OPC binary specification is limited to represent some few complex data types. Nevertheless, the data alignments are often required to transport the data in native representation. The OPC complex data based on the use of an XML representation require large amount of memory and high intensity of memory management operations [Chilingaryan and Eppler 2005].

On the other hand, the requirement to convert the data between the XML data representation and the binary data representation makes it difficult to determine the required size of the memory buffer to hold the whole data. In providing the ability to support high performance in the automation and control systems, the *data cache technology* is increasingly important and is an optimal mechanism to ensure the guarantee of the large control network. For effective memory space allocation, each representation of the data item is associated with dedicated cyclic buffer of enough size. The buffer should be used to store all memory blocks related to the current data item representation and the corresponding representations of all cached data item values.

Finally, the fast possible solution to transmit the large amount of data is to use the binary data representation rather than the pure XML data representation. However, different platforms even different compilers of the same platform use different binary data representation such as different floating point formats, different string formats, and so forth [Eppler et al. 2004]. This is the main drawback when using the OPC complex data.

4. DATA CACHING DESIGN

To solve the large control network problem that appears in industrial automation systems, we propose a mechanism for implementing the data caching technology to overcome the limitation of the pure OPC DA performances. As mentioned, the OPC clients in control and monitoring systems can often read, write, and subscribe to the thousands of data points from/to the OPC servers. In addition, the relation between the OPC servers and clients is *n-to-m*, i.e., an OPC client can simultaneously interact with several OPC servers and several OPC clients can access to the same OPC server. A large amount of data transfers from the OPC servers to the OPC clients may consume a large amount of the network bandwidth and operating system resources. Consequently, additional improvements of the OPC DA performances for large network control systems are required in order for reaching an acceptable performance. This section introduces the design of *Server Data Cache* for the OPC server and the design of *Client Data Cache* for the OPC client, respectively.

4.1 Server Data Cache

Several OPC clients can simultaneously access to read, write, and subscribe to the data from the OPC server. Therefore, caching process data in the OPC server-side is gaining increased importance in the data cache implementation of the OPC DA products. In the client-server system, the data caching is typically implemented on the server side because it is simpler to implement and easier to synchronize and maintain [Carpenter et al. 2001]. This section introduces a mechanism to implement the data caching into the OPC DA server. The cache server provides high performance caching and reduces the overall bandwidth. In addition to reducing the bandwidth, it can be easily employed in any distributed system, including on control and monitoring systems. Alternatively, the data from automation and control systems are complex data types, so that each OPC item contains the complex data value.

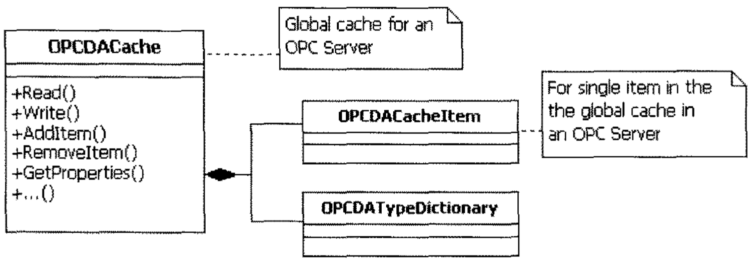


Figure 5. The class diagram of the OPC Server Data Cache.

Data caching is therefore made for each OPC item stored as a process data variable in the OPC server.

To effectively manage the OPC complex data items, the *OPCDADictionary* class that is used for managing the complex item types and complex type descriptions is created as shown in Figure 5. The *OPCDACacheItem* class used for single OPC item in the global cache in an OPC server is also created. The *OPCDACache* class is composed of such two classes above. When a hardware I/O device generates new data, an OPC server is notified and received the data from it. The data stored in an OPC complex item are identified as a process variable. An OPC server has a list of the OPC clients which are subscribed to the server data generated by the hardware I/O devices on the plant floor.

In responding and receiving the data element from the hardware I/O devices, the OPC server sends the process data over the network to a respective OPC client or to a list of the OPC clients which are interested in the data. To utilize the data caching performances on the OPC server side, the OPC DA Server Data Cache is controlled by a caching management that performs a removal of seldom used or unimportant OPC items to free some memory. As shown in Figure 6, the OPC complex items are placed as process variables in the OPC DA server in which they are variables to store the process data from the hardware I/O devices on the plant floor.

4.2 Client Data Cache

Data caching at the client side is indeed common for the implementation of a system in the client-server systems to reduce both database load and network traffic. We introduce an approach in order to implement a cache technology in the OPC client side. All OPC items contained in the Client Data Cache are further available as the OPC items using OPC DA interface that is provided for the client data cache. This approach is especially important for HMI (Human Machine Interface) components that allow clients to be able to subscribe to a large amount of the process data. The data available will be largely improved due to the lack of data transfer over the network between the client data cache and client HMI components. As discussed previously, an OPC item will contain such values as *complex data, timestamp*, and *quality* for reading, writing, and subscribing from/to the OPC servers acquired by the OPC clients. When the OPC server is notified and received the new data from the hardware I/O devices on the plant floors, it will send the changed data over the network to a respective OPC client or to each of a number of the OPC clients which

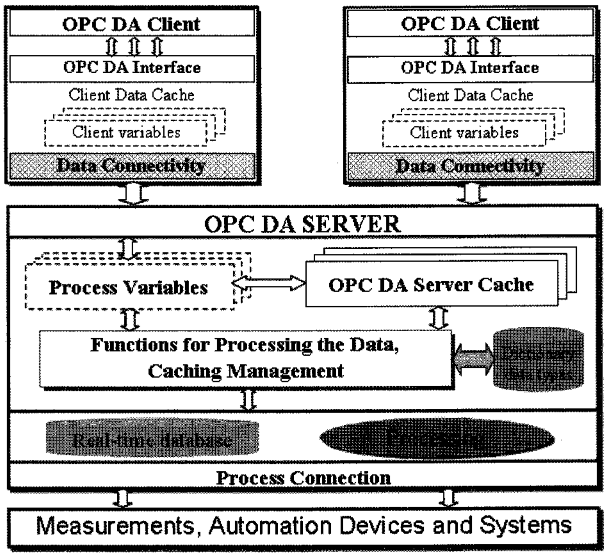


Figure 6. The architecture of the data caching design for the OPC client-server model used in process control and monitoring systems.

are interested in the data.

In the strategy of the caching technologies, the most important features in the OPC Client Data Cache are to allow customizing how the OPC items are cached and how long they are cached. Thus the OPC Client Data Cache needs a caching management that performs a removal of seldom used or unimportant items to free some memory. On the other hand, the caching management can be done during adding an OPC item into the OPC Client Data Cache. To apply the algorithm for solving these problems, some caching techniques can be applied such as *client polling*, *invalidation callback*, *time-to-live*, and so forth [Carpenter et al. 2001; Barish and Obracke 2000]. The idea is to partition the cache into expired and fresh data at the time when an eviction is to occur. If the expiration time associated with the version of the item is after next request, then the item is fresh; if it is before the request time, then the item is expired.

As Figure 6 shows, the data cache architecture model in the OPC client is presented. The data generated from the hardware I/O devices on the plant floor are updated to the OPC servers and then the OPC server will send its new data to the OPC clients which are interested in the data. The OPC clients will store the data in the OPC Client Data Cache as client variables that will be refreshed when respective OPC items are to expire. The algorithm of caching the process data in the OPC client is proposed as shown in Table I.

5. PERFORMANCE EVALUATION AND DISCUSSION

The proposed system first supports the OPC DA clients to read or write the data as complex data types from/to the hardware I/O devices on the plant floor and OPC servers. It allows applying to the real automation and control systems where the

Table I. The algorithm of caching the process data for the OPC client in the OPC client-server model.

ALGORITHM 1: CLIENT DATA CACHING

```
INPUT:
    • OPC items
OUTPUT:
    • Data caching for OPC items was added in the Client Data Cache

 1. if (OPC item is not in Cache) then
 2.   Fetching the values, i.e, complex data, timestamp, and quality, from original server and
      sending them to the client
 3.   if (Cache size is enough) then
 4.     Inserting this OPC item into the OPC Client Data Cache
 5.     Updating the data
 6.   else
 7.     Removing the seldom used or unimportant OPC items
 8.     Inserting this OPC items into the OPC Client Data Cache
 9.     Updating the data
10.   end if
11. else
12.   if (Expiration time is before the request time) then
13.     Fetching from the original OPC servers those hinted objects triggered by requesting
        new data
14.     Sending the values, i.e., complex data, timestamp, and quality, to the Client Data
        Cache
15.     Evicting expiration value, i.e., old values
16.     Updating the data
17.   end if
18. end if
```

data include simple and complex data. Moreover, the communication between the OPC clients and the OPC servers is *n-to-m*, i.e., an OPC client can simultaneously interact with several OPC servers and several OPC clients can access the same OPC server. The data caching in both the OPC servers and the OPC clients is therefore truly needed. With the data caching in the OPC server, the data are stored in the OPC items as process variables.

When the hardware I/O devices generate the data, these data will be acquired to store in the process variables if necessary. As a result, the OPC clients can subscribe to the data from the OPC servers easily and effectively. The caching management performs a removal of seldom used or unimportant OPC items to free some memory as well as possible. In addition to the data caching in the OPC clients side, the OPC Client Data Cache allows easily customizing how the OPC items are cached and how long they are cached, so that the life of the OPC items is effectively managed. The caching management can be done during adding an OPC item into the OPC Client Data Cache to indicate that the design is flexible. Moreover, the OPC item value in the OPC clients will be refreshed when the data from the OPC servers are changed

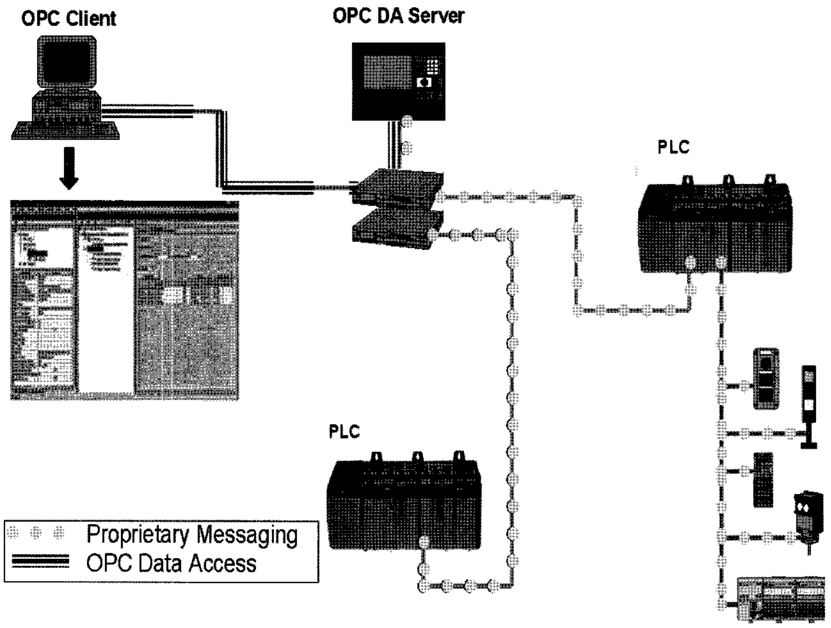


Figure 7. The illustration of deploying the proposed system to real-time industrial systems.

by calling the *callback* function in the OPC clients. By this way, the data from the hardware I/O devices are only updated to the OPC items when their data really need to be updated. Consequently, the amount of bandwidth and operating system resources are reduced as much as possible. The system performance indicated that the proposed approach has an acceptable performance to real-time industrial systems that strictly require high performance.

By comparing with the normal OPC DA product, i.e., without any performance improvement implementations, the performance of the proposed system with cache techniques is about several times higher than the normal OPC DA product's performance. In industrial systems today, the OPC technology-based applications normally control and monitor more than the hundreds or thousands of devices, so that the optimized software for control and monitoring systems is needed to be developed and deployed. The proposed solutions in this paper are to improve the performance of these systems when the number of the OPC items in the client for request is largely increased. The illustration of deploying the proposed system into real-time industrial systems for controlling and monitoring the hardware I/O devices on the plant floor is shown in Figure 7.

To compare the proposed system with others, it is difficult to conduct a fair comparison of different proposals and architectures because of their conceptual design and wide range of experimental environment setup. Thus the structural characteristics of the proposed system such as *ability*, *adaptability*, *scalability*, and so forth are used to compare it with others. Therefore, a qualitative comparison of the proposed system and the existing approaches could be made. The overall evaluation when comparing the proposed system with the existing approaches [TheOPCFoundation

Table II. The comparison between the proposed system and existing approaches according to the qualitative intention with the six factors.

	The proposed system	The OPC DA implementation	[Veryha 2005]	[Chilingaryan et al. 2005]
Complex data	✓			✓
Server data cache	✓	✓		✓
Client data cache	✓		✓	
Ability to reuse components	✓		✓	
High performance support	✓			✓
Scalability	✓			
Overall evaluation	good	none	fair	fair

2003; Veryha 2005; Chilingaryan and Eppler 2005] can be summarized in Table II. This overall result indicates that the proposed system is good and makes it easy in order to reuse and maintain the developed components.

6. CONCLUDING REMARKS AND FUTURE WORK

This paper has introduced the design and implementation of the OPC DA system in order to support complex data and high performance, applying to control and monitoring systems, e.g., SCADA. The proposed system is optimal for supporting OPC DA clients to read and decode any type of data from the hardware I/O devices on the plant floor. This solution is simple to implement and easy to maintain the implemented components in the OPC DA servers. Moreover, an effective approach to improve the performance of a system by using data cache techniques is presented. This approach is powerful and easy-to-use client's and server's data caching to overcome the limitation of the performance of the pure OPC DA systems. It has solved the problems when the large amounts of data are transferred between the OPC DA clients and the OPC DA servers. Additionally, it is easy to integrate to applications that achieve the process data through the OPC DA products today.

The system performance and analysis have indicated that the proposed system has a good performance and is feasible for supporting the real-time industrial control systems. The proposed system is optimal and affordable when comparing it with the normal OPC DA systems.

In addition to the control and monitoring systems, the applications have stringent real-time constraints, service-depend expiration times. Due to them, the real-time benefits of caching the process data from the plant floor must be carefully assessed in future work. On the other hand, the security for OPC DA standard-based systems is also strictly required. The security guarantee will be significantly integrated in the proposed system in the future work.

ACKNOWLEDGMENTS

The authors would like to thank the Korea Ministry of Knowledge Economy, Ulsan Metropolitan City, University of Ulsan, and the Network-based Automation Research Center (NARC) which partly supported this research. The authors also would like to thank the anonymous reviewers for their carefully reading and commenting this paper.

REFERENCES

BARISH, G. AND K. OBRACKE. 2000. World wide web caching: Trends and techniques. *IEEE Communication Magazine on Internet Technology Series* 38(5):178–185.

CAO, P. AND C. LIU. 1998. Strong cache consistency in the world wide web. *IEEE Transactions on Computers* 47:445–457.

CARPENTER, T., R. CARTER, M. COCHINWALA, AND M. EIGER. 2001. Client-server caching with expiration timestamps. *Journal on Distributed and Parallel Database, Academic Publishers* 10:5–22.

CHEN, D., A. MOK, AND M. NIXON. 1999. Real-time support in com. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences* 3:1–10.

CHILINGARYAN, S. AND W. EPPLER. 2005. High speed data exchange protocol for modern distributed data acquisition systems based on opc xml-da. In *Proceedings of the 14th IEEE-NPSS Real Time Conference*:352–356.

EPPLER, W., A. BEGLARIAN, S. CHILINGARIAN, S. KELLY, V. HARTMANN, AND H. GEMMEKE. 2004. New control system aspects for physical experiments. *IEEE Transactions on Nuclear Science* 51(30):482–488.

FISCHER, P. 1998. Real-time extensions to opc. *Real-Time Magazine-3Q98*, 76–82. http://www.omimo.be/Magazine/98q3/1998q3 p076.pdf.

FRANKLIN, M. 1996. Client data caching: A foundation for high performance object database systems. *The Kluwer International Series in Engineering and Computer Science 3.*

FRANKLIN, M. J., M. J. CAREY, AND M. LIVNY. 2006. Transactional client-server consistency: Alternative and performance. *ACM Transactions on Databases* 22(3):315–363.

GOPALAN, P., H. KARLOFF, A. MEHTA, M. MIHAIL, AND N. VISHNOI. 2002. Caching with expiration times. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*: 540–547.

HOLLEY, D. W. 2004. Understanding and using opc for maintenance and reliability applications. *IEE Computing and Control Engineering*:28–31.

ICONICSInc. 2008. http://www.iconics.com/.

IWANITZ, F. AND J. LANGE. 2006. Opc: Fundamentals, implementation, and application. *Huthig Verlag Heidelberg*, 3rd rev. Ed.

KAGHAZCHI, H., J. HAYES, AND D. HEFFERNAN. 2007. Development of an opc server for a fieldbus diagnosis tool. In *Proceedings of 5th IEEE International Conference on Industrial Informatics*:329–334.

KEW, S. J. AND B. DWOLATZKY. 2001. Real-time performance of opc in a feedback system. 1–10. http://www.cs.unisa.ac.za/saicsit2001/Electronic/paper45.pdf.

LINE, M. B., M. G. JAATUN, AND Z. B. CHEAH. 2008. Penetration testing of opc as part of process control systems. In *Proceedings of the 5th International Conference on Ubiquitous Intelligent and Computing, LNCS 5061*:271–283.

LIU, J., K. W. LIM, W. K. HO, K. C. TAN, A. TAY, AND R. SRINIVASAN. 2005. Using the opc standard for real-time process monitoring and control. *IEEE Software* 22(6):54–59.

SoftwareToolbox. 2008. http://www.toolboxopc.com/.

TAN, V. V., D. S. YOO, AND M. J. YI. 2006. Design and implementation of web service by using opc xml-da and opc complex data for automation and control systems. In *Proceedings of the 6th IEEE International Conference on Computer and Information Technology*: 263.

Technosoftware. 2008. http://www.technosoftware.com/.

TENG, W. G., C. Y. CHANG, AND M. S. CHEN. 2005. Integrating web caching and web prefetching in client-side proxies. *IEEE Transactions on Parallel and Distributed Systems* 16(5):444–455.

TheAdvosolInc. 2008. http://www.advosol.us/c-3-client-components.aspx.

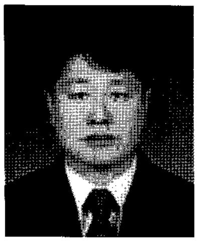
TheOPCFoundation. 2003. The opc complex data specification version 1.0. http://opcfoundation.org/Downloads.aspx.

TheOPCFoundation. 2004. The opc data access specification version 3.0. http://opcfoundation.org/Downloads.aspx.

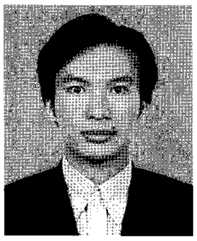
VERYHA, Y. 2005. Going beyond performance limitations of opc da implementation. In *Proceedings of the 10th IEEE Conference on Emerging Technology and Factory Automation* 1:47–53.

WANG, S. Y., L. ZHONG, AND Y. L. CAO. 2006. A data synchronization mechanism for cache on mobile client. In *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*: 1–5.

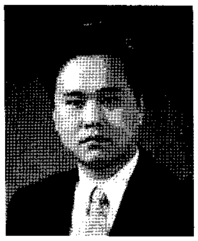
YANG, Q. AND H. H. ZHANG. 2001. Integrating web prefectching and caching using prediction models. *World Wide Web* 4(4):299–321.



Dae-Seung Yoo received the B.S. and M.S. degrees in Computer Engineering and Information Technology from the University of Ulsan, Republic of Korea, in 1998 and 2001, respectively. Now he is a guest professor in the School of Computer Engineering and Information Technology, University of Ulsan. He is a member of KIISE, KIPS, and IEICE. His main interests are software engineering, software for automation systems, and Internet technologies for industrial automation systems.



Vu Van Tan was born in Haiduong province, Vietnam, in 1981. He received the Eng. degree in Information Technology from the Hanoi University of Technology, Vietnam, in 2004. He has worked as design and analysis engineer in KhaiTri Software Company, Vietnam, for one year. He is currently a Ph.D student at the School of Computer Engineering and Information Technology, University of Ulsan, Republic of Korea. He joined in the Applied Software Engineering Lab, University of Ulsan, in 2005. His main interests are software engineering, software for automation systems, Internet technologies for industrial automation systems, and real-time communication systems.



Myeong-Jae Yi received the B.S. degree in Computer Science from the Seoul National University, Republic of Korea in 1987. He also received the M.S. and Ph.D degrees in Computer Science from the Seoul National University in 1989 and 1995, respectively. He was a part-time lecturer at the Department of Computer Science of the Seoul National University and the Sookmyung Women's University from 1991 to 1996. He is currently a professor in the School of Computer Engineering and Information Technology, University of Ulsan, Republic of Korea.
Prof. Yi is also a vice-Director of NARC (Network based Automation Research Center) at the University of Ulsan and is a member of KIISE and KIPS. His main interests are software engineering, software for automation systems, Internet technologies for industrial automation systems, mobile agent, and E-commerce.