IDC Technologies - Books - 1031 Wellington Street West Perth WA 6005

Phone: +61 8 9321 1702 - Email: books@idc-online.com,publishing@eit.edu.au

OP-E - Practical Fundamentals of OPC

Availability: In Stock

Price: $139.94

Ex Tax: $127.22

Short Description

OPC has come a long way in making the engineers' dream of plug and play compatibility in automation engineering achievable. OPC is an industry-wide standard that breaks this proprietary lock by allowing open connectivity based on the principles adapted from widely accepted and applied 'Microsoft Windows' integration standards. OPC capabilities have been demonstrated in many practical applications and it is now a well-established approach for different competing manufacturers.

Description

OPC has come a long way in making the engineers' dream of plug and play compatibility in automation engineering achievable. OPC is an industry-wide standard that breaks this proprietary lock by allowing open connectivity based on the principles adapted from widely accepted and applied 'Microsoft Windows' integration standards. OPC capabilities have been demonstrated in many practical applications and it is now a well-established approach for different competing manufacturers.

It is now easily considered to be the standard interface in the Windows environment. If you are serious about reducing your costs of installing and maintaining your automation systems you need to use OPC. If you have only briefly heard about OPC and want to get to grips with its tremendous power and

apply this to your plant and application, then this manual will give you the necessary tools.

You will receive a valuable overview of OPC and understand why it is the standard of choice for data access in automation systems. You will be exposed to and understand the essential components of OPC based on typical applications. You will understand how OPC servers are installed and how they are then accessed by OPC clients (which could be SCADA systems). The OPC specification is discussed in great detail and you will be exposed to the development of OPC servers and clients.

Table of Contents
Download Chapter List

First Chapter
Overview - Practical Fundamentals of OPC

1 OPC Overview

1.0 What is OPC?

OPC originally stood for ‘OLE (i.e. Object Linking and Embedding) for Process Control.

Since OLE has been superseded by ‘ActiveX’ this acronym has become meaningless and consequently ‘OPC’ has become just a name. It is sometimes said to represent ‘Openness, Productivity and Connectivity’.

OPC is an open, standards-based infrastructure for the exchange of process control data and specifies a set of *software* interfaces for several logical (software) objects, as well as the methods (functions) of those objects. It is a software standard supported by all major process control hardware vendors and most OPC products (clients and servers) operate very much in an ‘out-of-the box, plug-and-play’ fashion. If they so wish, users can develop their own clients and servers by means of developers’ toolkits. It is even possible to develop simple HMI systems without any programming knowledge at all.

The scope of OPC is to focus primarily (though not exclusively) on the exchange

of ‘raw’ data, i.e. the efficient reading and writing of data between an application and a process control device. In other words, OPC is a ‘window’ through which plant data can be observed. It accomplishes this by specifying the ‘rules’ for the data exchange in sufficient detail to allow vendors to implement low level (i.e. software) interfaces to process data.

There are, however, certain implementation issues not addressed by OPC (albeit deliberately). These include:

- Hardware interconnection. The protocol issues (drivers etc) for the I/O subsystems are not covered by the OPC specifications, and have to be taken care of by the server vendors
- How clients and servers are to be implemented. Apart from specific guidelines in the OPC specifications, and strict adherence to the interface definitions, vendors can implement clients and servers in any way they want. As a result, OPC products differ in terms of speed, reliability and interoperability

The goals of the original OPC Task Force was to develop a way to facilitate easy access to process data. The system had to be easy to use, easy to implement (especially on the server side), had to operate efficiently in terms of the usage of system resources (such as CPU usage), had to have a reasonably high level of functionality, and had to be flexible enough to accommodate the needs of multiple vendors. These goals were met, and the original; ‘OPC Task Force’ (1995), representing Fisher-Rosemount, Intellution, Intuitive Technology, Opto22, Rockwell Automation, Siemens, and Microsoft has grown into the present OPC Foundation with several thousands of members.

Figure 1.1

OPC Foundation logo

1.2 The problems addressed by OPC

1.2.1 Process control architectures

Most of the problems addressed by OPC can be traced back to the use of a multi-layered process control architecture and a heterogeneous computing environment.

Automation systems provide users with three primary services namely control, configuration, and data collection. Control involves the exchange of time-critical data between controlling devices such as PLCs and I/O devices such as actuators and sensors. Networks that are involved in the transmission of this data must provide some level of priority setting and/or interrupt capabilities, and should behave in a fairly deterministic fashion, depending on the specific application.
The second type of functionality, namely configuration, typically involves a personal computer or a similar device in order for users to set up and maintain their systems. This activity is typically performed during commissioning or maintenance operations, but can also take place during runtime, e.g. recipe management in batch operations.

The third involves the collection of data for the purposes of display (e.g. in HMI stations), data analysis, trending, troubleshooting or maintenance.

Figure 1.2

Hierarchy of plant levels

Figure 1.2 shows a generic view of an automation system architecture. At the device level, information is exchanged primarily between devices and networks deployed on the plant floor. Fast cycle times are required, networks at this level are bit-or byte oriented, and data packets are fairly small. Examples are HART, AS-i, DeviceNet, PROFIBUS DP and Foundation Fieldbus H1. These are mostly bit-or byte oriented network technologies and are generically referred to as ‘field buses’.

At the control level data is primarily exchanged between HMIs and PLCs. At this level speed is less critical and the amount of data exchanged in a packet is, generally speaking, bigger. These are systems at this level is said to be message oriented, and examples are ControlNet and Foundation Fieldbus HSE.

At the information level we find larger computers (e.g. mainframes) often networked via Token Ring or Ethernet, often at up to 10 Gigabit speeds. Here, determinism is not a factor. However, at this level there could be a large number of different Operating Systems, such as Windows, UNIX and LINUX.

The current trend is for vendors to re-package their older systems by running them via a TCP/IP stack over Ethernet, and these systems are used at the device

as well as the control level. An example is the ODVA's Ethernet/IP, which is basically DeviceNet running on TCP/IP and Ethernet.

Field buses are deployed on the 'plant floor' and should be able to operate in a harsh environment (temperature, vibrations, EM-disturbances, moisture, etc.). They should be robust and should be easily installed by skilled people. Other desirable attributes are a high degree of integrity (i.e. no undetected errors), high availability, with a redundant topology if required, a deterministic type of operation, a built-in system supervision and diagnostics, support of fairly large distances (up to a few kilometers if required), support of non-real-time traffic for commissioning and diagnostics, and the option of intrinsic safety for some applications within explosive environments.

1.2.2 Problems arising from the process control architecture.

Various categories of personnel, from operators to the CEO, need to access data across all three layers at various points in time. This is difficult or almost impossible due to various factors, relating to the data sources, data formats, data users, the interconnection between data sources and data users, and Operating Systems. These will now be discussed in more detail.

1.2.3 Data sources

Data sources could include (but are not limited to):

- Computers
- Databases (e.g. SQL)
- Programmable Logic Controllers (PLCs)
- Distributed Control Systems (DCSs)
- Stand-alone (e.g. PID) controllers
- Remote Terminal Units (RTUs)

Each of these would use a different protocol at layer 7 of the OSI model. Even if they were all interconnected via the same physical network, the interrogating (client) system would need to be able to converse via several protocols.

1.2.4 Data format

Data format is another problem area. The data source (e.g. server) could, for example, make its data available in one of the following formats:

- Boolean (single bit)
- Character (signed 8-bit)
- Word (unsigned 16-bit)
- Short (signed 16-bit)
- Dword (unsigned-32 bit)
- Long (signed-32 bit)
- BCD (2-byte packed BCD)
- LBCD (4-byte packed BCD)
- Float (32-bit floating point number)
- Double (64-bit floating point number)
- String (null terminated ASCII string)

The client software would need to accommodate all of these.

1.2.5 Data users

The plant data obtained from various sources could end up being used by multiple clients. These could include:

- Human-Machine Interfaces or HMIs (a.k.a. MMIs)
- Supervisory Control and Data Acquisition (SCADA) systems
- Trenders
- Archivers
- Electronic Resource Planning (ERP) or Manufacturing Resource Planning (MRP) systems
- Spreadsheets
- Process Historians
- Machine condition monitors

This data might be used for functions such as

- Reporting
- Operator interfacing
- Trending
- Alarming
- Control strategies

Depending on the application, data could be accessed in groups or per individual items. It could either be polled by the client at constant intervals or be forwarded by the server to the client on an exception basis.

1.2.6 Interconnections

Interconnections, especially between Server host and I/O device, could be varied. Technologies and standards employed could include:

- Point-to-point links. Here, options include fiber, RS-232, RS-422, Bell-202 type modem links and 900 Mhz/2.4 GHz microwave links, to name but a few
- Multi-drop buses. Options here include RS-485 and IEC 61158
- Options include Modbus, DNP3 and DF-1, to name but a few.
- Fully-fledged industrial networks, implementing all (or at least layers 1, 2 and 7) of the OSI model stack include Ethernet/IP, PROFIBUS and Foundation Fieldbus. There are several dozen contenders ('field buses') at this level

1.2.7 Operating systems

This is another problem area. The various operating systems used server as well as client hosts could vary between Windows (95/98/NT, 2000, XP, CE, 2003 etc), UNIX, LINUX and various real-time operating systems for embedded controllers.

1.2.8 Drivers

Historically, the products that go into a process control system (such as field instrumentation and control components) have been proprietary in nature. Gradually, several hardware and signal exchange standards evolved in the field of analog and digital instrumentation to ensure that the components manufactured by different vendors can communicate with each other, at least at a low level. For example, most analog field devices such as transmitters communicate using standard signal values in the range of 4 to 20 mA.

The control system (or indicating instrument for that matter) to which this transmitter is connected can thus compute the actual flow rate based on this commonly understood, standard, signal. This means that the control equipment need not be custom-designed for each transmitter, but can be designed and manufactured irrespective of the parameter being measured, indicated or controlled. In effect, a single measuring or control device can be used for flow, pressure, temperature or any other conceivable parameter of any possible measurement range.

Another benefit is, of course, the convenience of having a vendor independent system. A transmitter supplied by a given vendor can thus be used with an indicating instrument or a PID controller from any other vendor, provided they use the same 4-20 mA standard. The output devices in the field, such as control

valves, are also based on the same standard. This means that a control system that requires the control valve regulating the controlled parameter to be kept fully open will simply send a signal of 20 mA to the valve.

While the example of a transmitter/controller/control valve typifies a hardware standard, widespread use of computers for process control has given rise to a need for standardization of software components as well. We are not talking about the control software of a particular system component here; that remains proprietary. In fact, the control software itself is of no relevance to anyone except the manufacturer of that hardware or to a developer writing applications specific to that hardware device. (Here too, standardization efforts have been taken up, such as the standard for PLC Programming as per IEC-61131-3, but such standardization may not be feasible for all types of devices). What is important is that, when it comes to exchange of data between systems, a common standard is absolutely essential.

Figure 1.3 shows a simplified example of a high-speed rotating machine control system.

Figure 1.3

Control system example

The lower portion of the diagram shows three independent control systems, each serving as a data source. They are:

- A PLC for interlocking operations and for supervision of vital operating parameters
- A vibration instrument that measures machine vibration
- A vendor-supplied performance calculation engine

The PLC will be connected to several I/O devices for control of the machine, using its own standard interfaces and communication protocols. All these are essentially self-contained devices that may include their own HMI hardware or other form of output devices such as hardwired indicators, annunciators and so on. However, they are also required to communicate with other devices provided for overall control of the process itself, of which this particular machine is a component.

The upper portion of the diagram shows three such devices:

- An HMI used by the process control operator for obtaining the system status, the alarm conditions when there is an undesirable deviation from specific threshold values, and often for control of the process
- A data archive for storing various measured values, events etc.
- A machine condition monitoring system

Assume that all data sources and data users shown in this example are proprietary systems. We can see right away that there is a need for each of the data user devices to obtain data from each of the data sources shown here. For example, the HMI will need status and alarm data from the PLC. It will also require vibration readings and critical alarms in the event of excessive vibration values for operator information. It will need interaction with the calculation engine to display critical calculated parameters to the process operator and use the results as inputs to say, an expert system application for operator guidance. The same will be true of the data archive device and the machine monitoring system also (refer Figure 1.4).

Figure 1.4

Communication between proprietary systems using separate drivers

With all these systems being proprietary in nature, direct communication between any two devices is not possible without some customized interfacing. The HMI will need an interface to access the data residing in the PLC. Similarly, it will need another interface to access the data from the vibration measuring instrument; and so on. Each interface is a special purpose software application known as a device driver (simply referred to as a ‘driver’). Each connection between two devices thus calls for two dedicated custom-developed drivers, one at each end. These cannot be used with any other device, and even between a pair of devices the driver may need to be rewritten if the data source or the data-using device is replaced or upgraded.

There is also another less obvious problem. Each data source device has to communicate through multiple drivers to the data users. This means that identical data may be requested by multiple user devices and has to be communicated separately to each by the data source. With slow serial protocols such as Modbus Serial, such a requirement puts a severe strain on the communication infrastructure. Thus:

- We need nine device drivers in this scenario

- Each will have to be custom-developed
- Each may become useless if any of the corresponding devices is replaced or even upgraded

The result is a lack of interoperability, getting locked-in to specific vendors, and time consuming software development and on-going management. Such a situation is neither in the user's interest, because of the difficulties cited above, nor in the manufacturers' interest, because they are the ones expected to supply the drivers. It is thus obvious that we can arrive at a 'win-win' situation by making all devices talk in a common language. Let us now modify the example in Figure 1.4 with another, representing communication through such a common language. This is shown in Figure 1.5.

.

Figure 1.5

Communication between proprietary systems via OPC

In this scenario, each data source has a software component for making its data available to other systems (let us call it the 'server') and each data user has a software component to access the server for data (let us call this the 'client'). We have therefore created a 'software bus'. By adopting a common data interface standard, it is possible for any client to request data from any server in a format that the server can understand. The server, in turn, makes the requested data available to the client in a format that the client can understand. As long as the client and server applications are all based on a common specification, any client and any server can thus exchange data thereby ensuring interoperability.

Since there is no proprietary component in this data exchange mechanism, it automatically ensures vendor independence. Scalability, too, is not an issue since a new data source following the same data exchange specification can easily be added. Device upgrades pose no problem either, since the data exchange mechanism will still be based on the same common standard. Server vendors are also not required to develop drivers to suit various client products, but simply have to implement the required OPC interfaces.

1.3 The logical object model

It is, strictly speaking, inappropriate to talk about a logical (i.e. software) object model for OPC as each specification has a different object structure. In other words, there is a DA object model, an AE object model, etc. This is, in fact,

considered a shortcoming and is being addressed by the new UA standards. We will clarify the concept be looking at the most poplar specification to date, the DA (Data Access) specification. Initially this was known simply at the ‘OPC Specification’.

There can be a one-to-one relationship between Client and Server, or, alternatively, there can be a one-to-many or many-to-one relationship between them as shown in the following figure. However, although multiple Client-Server connections are possible, this is dependent on specific vendor implementations and hence not always possible.

Figure 1.6

Basic relationship between OPC Clients and Servers (courtesy OPC Foundation)

OPC Clients and Servers are simply COM objects (in fact, all software programs on a Windows-based computer exist as COM objects and this can be verified with ‘OLE Viewer’ software. The connection between client and server is via well-defined interfaces, specified in Microsoft Interface Definition Language (IDL) and typically implemented in C++. A typical interface specification looks like this and only makes sense to a programmer. The example is for the CloneGroup method of the IOPCGroupStateMgt interface:

IOPCGroupStateMgt::CloneGroup

```
HRESULT CloneGroup(

        [in, string]LPCWSTR szName,

          [in]REFIID riid,

          [out, iid_is(riid)] LPUNKNOWN * ppUnk

          (:
```

The infrastructure to carry the data between the Client and Server interfaces is supplied by DCOM, described in the next chapter.

A more complex Client-Server relationship is shown in the following figure.

Here, the Client (A) obtains data from the server (B) via an OPC interface. The Server, in turn, obtains I/O data from a Device (e.g. a PLC), shown as C. The channel between the Server and the Device could, for example, be Ethernet or RS-485, and the Server vendor would develop the Device drivers in consultation with the Device vendor. In this specific scenario there is also s SCADA system (D) that obtains its own data via an I/O subsystem (D). If the SCADA system is OPC compatible, i.e. it has a built-in OPC server, then OPC server B could obtain data directly from the OPC server on the SCADA system via OPC Bridging software. Obviously this indirect method of obtaining data would be an issue if fast access to the plant data is a prerequisite.

Figure 1.7

More complex relationship between OPC Clients and Servers

When ‘looking inside’ an OPC Server such as an OPC DA Server, the internal structure of the server becomes evident. The structure varies from specification to specification, but there is always a single Server control object at the top. In the case of DA this object has the very creative name of OPCServer (note the absence of a space after OPC). Below this are one or more OPCGroup objects, and below each of these is one or more OPCItem objects. The Interfaces of the OPCServer and OPCGroup objects extend to the outside world, but those of the OPCItems do not, and as a result a Client can only be interrogated the items indirectly via the OPCGroup interfaces.

Figure 1.8

Internal structure of DA Server

Incidentally, one of the drawbacks of the first generation of OPC specifications is that they are not built around a coherent data model, i.e. the object hierarchy of a DA server is different to that of an AE server. This has been addressed by the new Unified Architecture (AU) - refer to Annexure A.

1.4 OPC Specifications

The OPC Foundation grew out of the OPC Task Force founded in 1995. With the decision to base OPC on existing Microsoft OLE/DCOM technologies, the OPC

specification version 1.0 was already available by August 1996. This was possible because the focus was on essentials, and the approach of not ‘re-inventing the wheel’.

The first update was issued in September 1997. Now it was no longer called the ‘OPC Specification’, but rather the Data Access (DA) specification 1.0A. However, the need for acquiring data from a process goes beyond routine operating data to more specific varieties of data such as events, alarm conditions and historical data. Additional industry requirements and further developments in Microsoft DCOM technology led to the next version called Data Access specification 2.0.

Meanwhile, separate working groups were formed to cover the requirements of exchanging information on events and alarms and the acquisition of historical data. The OPC Alarms and Events (AE) specification was published in January 1999 (Version 1.01) and the OPC Historical Data Access (HDA) specification in September 2000. A separate OPC Security specification was published in September 2000 covering the security policies to be used by OPC components. The specific data exchange needs of batch process control applications were covered in a separate document called the OPC Batch specification.

As the number of OPC specifications increased, it was obvious that there are elements common to all the specifications. This led to the issue of two additional specifications. The first is known as the OPC Overview and contains only the explanatory aspects, while the second was called the OPC Common Definitions and Interface Specification and contains normative definitions. Other functionalities for extending the scope of the Data Access specification include the use of XML technology for enabling the use of OPC components for accessing data over the Internet and in operating systems that do not support DCOM. A separate working group called OPC DX deals with communication between servers without involving a client.

OPC is a relatively new technology and is changing and expanding rapidly. The ‘first generation’ OPC standards shown below are all based on Microsoft DCOM technology and (at October 2006) comprise the following specifications. All specifications relate to the ‘Custom Interface’ (i.e. for use with C++) unless indicated as ‘Auto’ for VB/VBA.

- OPC XMLDA (XML Data Access) 1.01
- OPC Commands 1.00 Specification 0.29
- OPC HDA (Historical Data access) 1.20
- OPC Complex Data 1.00

- OPC DX (Data exchange)1.00
- OPC DA (Data Access) 3.00
- OPC AE (Alarms & Events) 1.10
- OPC Common 1.10
- OPC DA 2.05a
- OPC Batch 2.00
- OPC Batch Auto 1.00
- OPC HDA Auto 1.00
- OPC Security 1.00

DCOM, although a Microsoft invention, is not a proprietary solution that 'locks in' users to Microsoft as it has been implemented by several other vendors and is available for UNIX and LINUX as well. It is very efficient, as any object accessing another object can do so with a single Remote Procedure Call (RPC) to the other object, irrespective of the location of the other object. Provided that there is the required network connectivity, there is hardly any difference in Client-Server communication whether they are located on the same computer, or on opposite sides of the world.

DCOM does, however, have shortcomings and, in particular, is not entirely firewall-friendly and as a result it runs very well on a LAN, but not across on the Internet. DCOM operation is subject to the security mechanism on a computer, and recent security measures as implemented on Windows XP Service Pack 2 plays havoc with DCOM. It is also prone to time-outs, a situation that is undesirable in process control applications. As a result, the OPC foundation decided to migrate OPC to the Microsoft .NET ('dot-NET') technology. A recent decision was also to combine all the OPC functionalities in the so-called Unified Architecture or 'UA' specifications.

UA is a very new concept and at this time (October 2006) only the UA Concepts document has been released to the public. The other UA specifications are still only available to OPC Foundation members, and some are still in draft form. AU will therefore only be addressed in detail in the next revision of this manual, but as an interim measure an overview is given in Annexure A.

The current UA specifications (October 2006) are:

- OPC Test Lab RC1.00
- OPC UA Part 1 Concepts 1.00
- OPC UA Part 2 Security Model 1.00
- OPC UA Part 3 Address Space Model 1.00
- OPC UA Part 4 Services 1.00

- OPC UA Part 5 Information Model 1.00
- OPC UA Part 6 Mappings RC0.93
- OPC UA Part 7 Profiles Draft 0.23
- OPC UA Part 8 Data Access RC1.31
- OPC UA Part 9 Alarms Draft 0.62
- OPC UA Part 10 Commands draft 0.62
- OPC UA Part 11 Historical Data Access Draft 0.991