10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems. Geneva, 10 - 14 Oct 2005, WE4A.4-5O (2005)

OPC evolution toward UNIX (from Windows to World Wide Domination?)

M.Beharrell[1], R.Barillère[1]
[1]*CERN, Geneva, Switzerland*

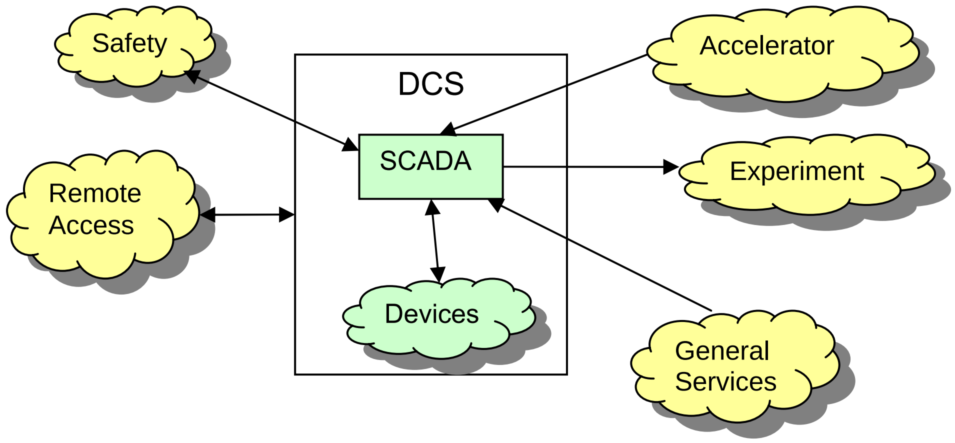
ABSTRACT

OLE for Process Control (OPC) is a middleware solution for integration problems found in an industrial environment. Typically used for connecting devices to higher level processes such as SCADA (Supervisory Control And Data Acquisition) systems, it has found wide spread acceptance in industry since its introduction in 1996. Today it is considered the de facto standard for supervisory / process control integration. A multitude of manufacturers produce for their devices OPC servers that provide their customers with a "plug and play" solution to the problem of integrating a device into their supervisory system.

Whilst significantly simplifying integrators' life, OPC was only available on MS Windows based platforms. Thanks to 1) the opening of the DCOM standard, 2) the introduction of XML OPC and 3) the introduction of OPC UA, this is no longer strictly the case. In this paper we will look at each solution, examining its performance and its respective advantages and disadvantages. We conclude the paper by examining the value of currently available solutions, in light of the strength behind OPC – its wide spread acceptance.

COMMUNICATIONS FOR DETECTOR CONTROLS

The figure below depicts an abstract view of data flows for a Detector Control System (DCS).



Typically, a Supervisory Control and Data Acquisition (SCADA) system obtains data from a wide variety of devices used in controlling the detector and summary information is passed to other systems of the experiment. Beam status information will come from the Accelerator and information will pass between safety systems and general services. There is additionally an increasing demand to access the DCS from remote sites. Information exchange between each of these communication domains takes place via one or more communication protocols. Substantial effort must be put into developing, debugging and maintaining these protocols.

This problem is particularly acute when communicating with hardware devices. In these situations it is normal to see different protocols used between 1) different device manufacturers, 2) models from the same manufacturer and 3) even different protocols for different versions of the same device.

Hence it is desirable wherever possible to use commercially available standard-based solutions that hide these differences in device access from the user of the device. One solution is to use middleware, which provides a common interface for access to (typically distributed) software entities. CORBA [1], Java RMI [2], DCOM [3] are just a just a few of the middleware solutions that exist. These however are intended to be general solutions and varying degrees of effort must be put into further developing these solutions for use for device access. These developments may be done in a dissimilar manner, hence introducing further protocols, each one requiring a client capable of handling this protocol.

Some effort has gone into the standardization of middleware for device access. In 2005 the Object Management Group introduced a specification specializing CORBA for data acquisition [4]. However, nine years prior to this the OPC foundation introduced the OLE for process control (OPC) set of specifications.

The OPC solution to device access

The OPC foundation was formed in May 1995 with the goal of producing a middleware solution suitable for the industrial setting. In August 1996 the OPC foundation produced version 1.0 of the OPC Data Access specification [5]. This describes a set of interfaces written in Microsoft's Interface Definition Language (IDL) that can be used to manage and access device data. It further detailed a data model in which not only is device data represented but also properties such as min/max values, data description, engineering units, time stamp etc. – thus making it easier for OPC users to understand the data they are receiving. Finally, the specification provided client/server and publish/subscribe communication models. Using these specifications developers are able write OPC-DA clients and servers.

The OPC-DA also had several weaknesses - only simple data types were supported, no support for commands, no explicit support for security, etc. Over the preceding years the OPC foundation has evolved the DA specification and additional specifications have been introduced to address these shortcomings.

Since its introduction OPC has become the de facto standard for industrial communication and it is difficult to find a SCADA system that does not contain an OPC client. It is thus common to purchase hardware which is accessed via an OPC server.

What this means in detector controls

The ability to purchase hardware and software to control it in a standard manner is an attractive one, reducing the manpower needed to develop and maintain middleware for the wide variety of devices used in a DCS. However, there is not a panacea.

OPC's reliance on DCOM for security leads to configuration problems – DCOM security can be difficult to understand and users often opt for little or no security as a result of this. The Windows' registry, on which DCOM relies, is proving fragile and prone to corruption. Windows upgrades (notably SP2) also add additional complexity in maintaining working OPC servers. Further, there is a tendency to use OPC servers to provide data for many thousands of data items in a time-constrained manner. Thus efficiency of the OPC server becomes paramount. Finally, but most notably, detector control systems tend to be heterogeneous in nature, typically, machines are Windows or Linux platforms based on x86 architectures, but, other combinations do exist. Because of the use of non-windows machines, requirements frequently exist for OPC servers on UNIX-based platforms. The standard answer is that these do not exist, although this is not strictly true…

OPC ON NON-WINDOWS PLATFORMS

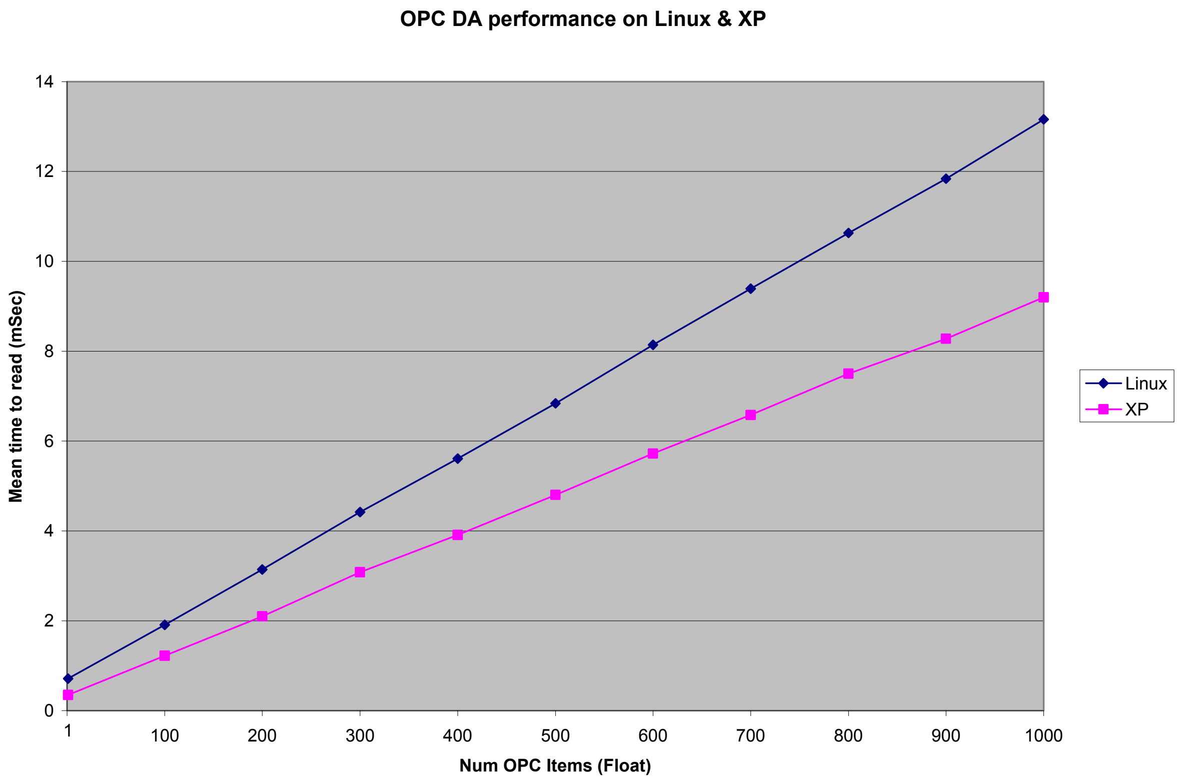
There are several solutions to implementing OPC servers on non-windows platforms (such as UNIX). We now briefly look at each solution in turn.

DCOM

Microsoft opened their DCOM specification in November 1996. This has resulted in a handful of DCOM implementations appearing for non-windows platforms such as Solaris, VXworks and Linux. These DCOM implementations are installed on the target platform in the form of libraries which developers use as a basis for their applications. With these DCOM libraries comes the ability to provide OPC servers on UNIX platforms.

We are interested in the efficacy of this, particularly with respect to the performance and stability of such solutions under Linux. To test this we ported the OPC foundation's sample implementation of the OPC DA 2.05 server to Linux using Software AG's EntireX DCOM libraries for Linux. We then compared the performance of the resulting implementation against the same server running on

Windows XP[1]. This test was performed by taking the mean of the time taken to perform 1000 reads of selected sets of OPC items (float data items) using the OPC-DA IOPCSyncIO read method (this allows the client to read one or more OPC items at the same time). The graph below compares the average time taken to read increasing numbers of OPC items from XP and Linux platforms.



We can see that whilst the Linux implementation is slower than the equivalent XP implementation, it still compares favorably to it. It is interesting to note the best end to end throughput of floats that the Linux implementation OPC appears to achieve is about 2.3 Mbps (remember other data such as timestamp and quality are also sent). Whilst this may seem slow, it represents data for over 76000 OPC items being read every second.

Our implementation had some stability problems: 1) there was a problem with the DCOM library routine for converting multibyte to wide strings, 2) the 'window registry' was prone to losing some data stored in it[2] and 3) the DCE-RPC server that DCOM relies on would suddenly become inaccessible (even though it was still running) - only reinstalling the DCOM libraries would solve this. These problems would have to be resolved before a usable OPC DA server on Linux can be achieved, but this is believed to be attainable. Alternatively, another DCOM implementation for Linux such as the Open Groups FreeDCE could be used, thus providing an OPC DA server on Linux that is equivalent to that found on Windows.

XML DA

In July 2003, the OPC foundation released the OPC XML DA specification. This is very similar to that used by OPC DA 3.0[3]. However, it is distinguished from previous OPC specifications in that it no longer uses DCOM as the transport layer and instead uses Web Services - basically remote procedure calls based upon HTML and SOAP. Depending on the implementation, an XML DA server may be

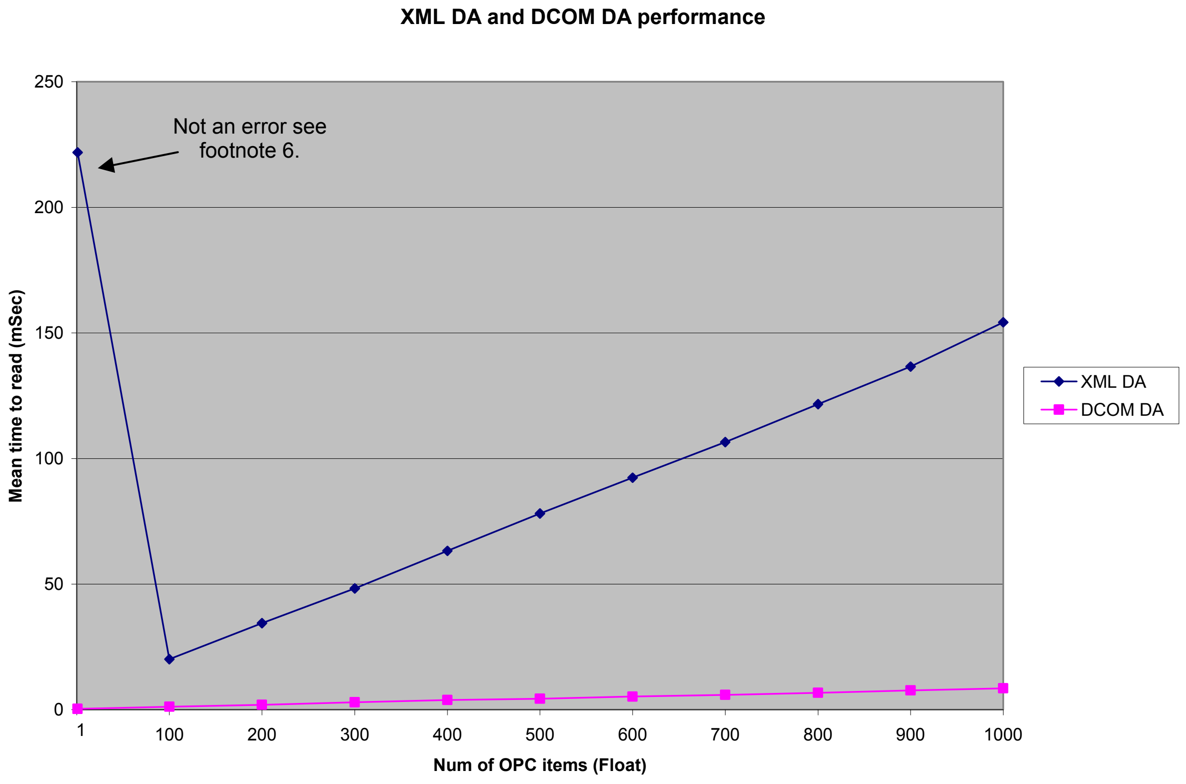
[1] The tests were performed with the servers hosted on a 2.66 Ghz P4 with 1.5GB RAM (dual boot Windows XP SP2 and CERN Scientific Linux 3) whilst the client was a hand written OPC client hosted on a 2.4Ghz P-4 with 1GB RAM running Windows XP SP2. Both client and server were connected via 100 Mbit Ethernet using a crossover cable.

[2] In particular the 'RunAs' settings for a DCOM component.

[3] Two notable differences are the lack of a true publish/subscribe communication model and no need to create OPC groups before reading OPC items.

embedded in a web server (such as Apache or IIS) or run as a standalone server.

Since HTML is a ubiquitous protocol, the possibility now exists to have OPC servers (that implement the XML DA specification) available on any platform capable of supporting HTML (or TCP). Because XML DA represents all its data as XML strings, there is the obvious concern about the efficiency of this protocol. To investigate this we followed the same methodology used in comparing DCOM implementations, this time reads were performed using the Read() request as defined by OPC XML DA v 1.1. We wrote the XML server and client using TechnoSoftware AG's evaluation versions of the OPC XML server framework C++ for windows[4] and OPC Client Framework.net[5] respectively. The graph below compares the results of our XML read tests with the results of the DCOM IOPCSyncIO read tests.



We see that there are substantial performance differences[6] between OPC XML DA and OPC DA. This can be improved by using other more efficient data encoding methods (for example base64 or hexadecimal encoding). However, these are not utilized by the XML DA specification.

Further efficiency concerns originate from the absence of a true publish/subscribe communicate model. This has been replaced by what is called the 'polled refresh', requiring the client to poll the server on a periodic basis to obtain fresh information regarding the OPC items it is 'subscribed' to. The OPC server may maintain lists of changed OPC items of interest to each of its clients – so as to avoid the loss of data between polls. If the client should poll slower than the prearranged rate, the server is free to de-allocate all resources for that client.

When building XML DA servers we found that tools used to parse the WSDL (Web Service Definition Language) files that describe the OPC web services did not always produce correct code[7]. Web services rely on XML, XML schema; SOAP and WSDL which are complex specifications. Any misinterpretation of these specifications can result in tools which produce incompatible Clients and

[4] Unfortunately, there was no evaluation version available for Linux. However, we feel that this implementation is representative of values that would be obtained using Linux.

[5] We wrote our own XML DA client in C++ based on GSOAP – unfortunately memory leaks that appear to originate for the GSOAP libraries meant we could not use the implementation for these tests.

[6] Note that the peak in the initial XML DA readings is not an error, rather a product of the latency introduced by the Nagle algorithm [6] used in TCP/IP.

[7] We found server side skeletons produced by GSOAP were one example of this.

Servers. This issue is not unique to OPC XML and compliance tests do exist to test compliance with specifications, but even these appear to miss some compatibility issues.

OPC UA

The OPC Unified Architecture is the newest product to come from the OPC Foundation. Introduced in April 2005, details of its specification are still very much in formation. Its goals are to:

- Enable the integration of structured 'plant floor' data at the level of the enterprise and internet.
- Provide secure, reliable and efficient services.
- Provide protocol and platform independence.

It represents an effort to produce a solution that not only works across multiple platforms but also integrates into enterprise systems and beyond. The architecture is a fusion of the existing OPC specifications into a single coherent specification. The objects (variables, properties and commands) are accessed via services. A service is a capability of the server and related services are grouped into service sets (examples of such sets are the data access services and the subscription services).

The UA aims to 'retire' DCOM, replacing it with performant SOAP-based alternative(s). The replacement interfaces are defined using WSDL with bindings to either HTML or TCP/IP. To address the performance issues associated with XML DA, the UA will encode data in a binary format. Initially it will use make use of base64 encoding to encode the data held in the body of the SOAP envelope. This will be replaced by Message Transmission Optimization Mechanism (MTOM) when its specification is finalized.

The performance of OPC UA is yet to be tested, but if UA meets its goals and overcomes the parser tool issues encountered with XML DA, it promises to be of great interest for use for communication within and outside of detector control systems.

SUMMARY

End to End performance

We have investigated the performance of the Linux DCOM and XML DA solutions to the problem of implementing OPC servers on UNIX systems. However, the performance of the OPC protocol is only one aspect that must be considered.

It is our experience that 'slow' OPC servers are not due to the OPC protocol itself, but rather the manner in which the OPC server access' its underlying devices – particularly network (Ethernet or Fieldbus) based devices. In these situations it is typical to find 1) an OPC server accessing multiple devices and 2) for a single network frame to contain multiple values. These two facts provide substantial opportunity of performance increase:

- Where the network capability permits, the Server can parallelize requests to multiple devices, instead of waiting for a response to a request before proceeding.
- The OPC server may group device reads according to the frame that provides that information, then instead of the OPC server requesting the same frame for each item (commonly seen), it requests the frame once, and updates the relevant OPC items from it.

It works but is it useful?

We have seen above that it is technically possible to implement both XML and DCOM-based OPC servers on UNIX like platforms. Whilst the technical solutions that work on UNIX are slower than pure Windows DCOM implementations, they still support a data rate that is likely to meet most DCS requirements. However, part of the advantage of OPC comes from its support from device manufacturers. It is 'a given' that if one purchases a device, one will also be able to obtain an OPC server for it – but this is a server that runs under Windows. Whilst OPC server and client toolkits exist for DCOM Linux and for XML OPC, they have yet to find acceptance by device manufacturers and end users, who prefer to stay with the Windows DCOM solutions which are known to work. The consequence of this is that if we want Linux-based OPC servers, they are likely to be custom developments. In doing this DCOM-based Linux servers are likely to be the preferred solution for the following reasons: 1) Performance & 2) the existence of a wide variety of OPC clients integrated into SCADA systems that can read these servers. Performance concerns and the lack of SCADA systems with integrated XML DA clients means it is likely that XML DA will only be used for wrapping existing DCOM based OPC servers giving them the ability to be easily accessed via the Internet.

An option that has not been explored in this paper, but whose potential warrants further investigating is that of implementing OPC DA clients in Linux. This should not be difficult to do and opens up the possibility of Linux-based SCADA systems have access to a multitude of available OPC DA servers.

Conclusion

For the immediate future we are likely to continue seeing Windows based OPC DA servers used in industry. XML OPC will be used in situations where OPC DA servers need to be accessed remotely, but will not be used in mainstream device access scenarios. OPC XML DA is likely to be surpassed by OPC UA which promises to be a more secure, robust, flexible and performant solution that can be used in many aspects of detector control system communication. Ultimately customer acceptance will determine whether OPC UA replaces OPC DA and becomes the new multi-platform de facto standard for integrating device data into supervisory systems, enterprise systems and beyond. In the mean time, if we want Linux-based OPC servers, we are likely to have to write them ourselves.

REFERENCES

[1] Object Management Group, Inc., http://www.omg.org.
[2] http://java.sun.com/products/jdk/rmi/
[3] http://www.microsoft.com/com/default.mspx
[4] "Data Acquisition from Industrial Systems Specification – Version 1.1", Object Management Group, Inc., June 2005
[5] "Data Access Custom Interface Standard", currently v 3.0, OPC foundation, March 2003.
[6] "Congestion Control in IP/TCP Internetworks", RFC 896, Network Working Group, John Nadle, 6 January 1984.