WHITE PAPER

How OPC UA Servers Facilitate Efficient SCADA Device Data Management

Charles ZK Chen
Moxa Product Manager

MOXA®

Abstract

A modern SCADA system communicates directly with an OPC server which itself communicates with PLCs, RTUs, and/or Moxa ioLogik remote I/O units to pass sensor readings and control signals back and forth between the OPC server and devices. Whereas the more traditional OPC DA server uses a polling method, which can use quite a bit of network bandwidth, the newer OPC UA server uses a "report by exception" methodology to reduce the amount of information that the OPC server needs to send to the SCADA software. The combination of OPC UA with Moxa's patented Active OPC server technology provides users with a seamless communication solution that can save an impressive amount of bandwidth.

In this white paper, we explain the difference between "updating data by polling" and "updating data by exception," give some general rules of thumb you can follow to decide which method is suitable for your various I/O devices, and introduce Moxa's new MX-AOPC UA server solution.

Released on December 23, 2014

Moxa is a leading manufacturer of industrial networking, computing, and automation solutions. With over 25 years of industry experience, Moxa has connected more than 30 million devices worldwide and has a distribution and service network that reaches customers in more than 70 countries. Moxa delivers lasting business value by empowering industry with reliable networks and sincere service for automation systems. Information about Moxa's solutions is available at www.moxa.com. You may also contact Moxa by email at info@moxa.com.

How to contact Moxa
Tel: 1-714-528-6777
Fax: 1-714-528-6778

Reliable Networks ▲ Sincere Service

Introduction

For more than half a century, SCADA systems have given operators located in a central control room the ability to monitor and control multitudes of devices spread out over a wide geographical area. The structure of a modern SCADA system, as depicted in Figure 1, has the SCADA software at the top, monitored devices at the bottom, and an OPC server in between. PLCs, RTUs, and/or Moxa ioLogik remote I/O units are used to pass sensor readings and control signals back and forth between the OPC server and devices. The PLC/RTU/ioLogik units provide the remote locations with a certain amount of autonomy, and are smart enough to implement local control schemes independent of the SCADA software itself.

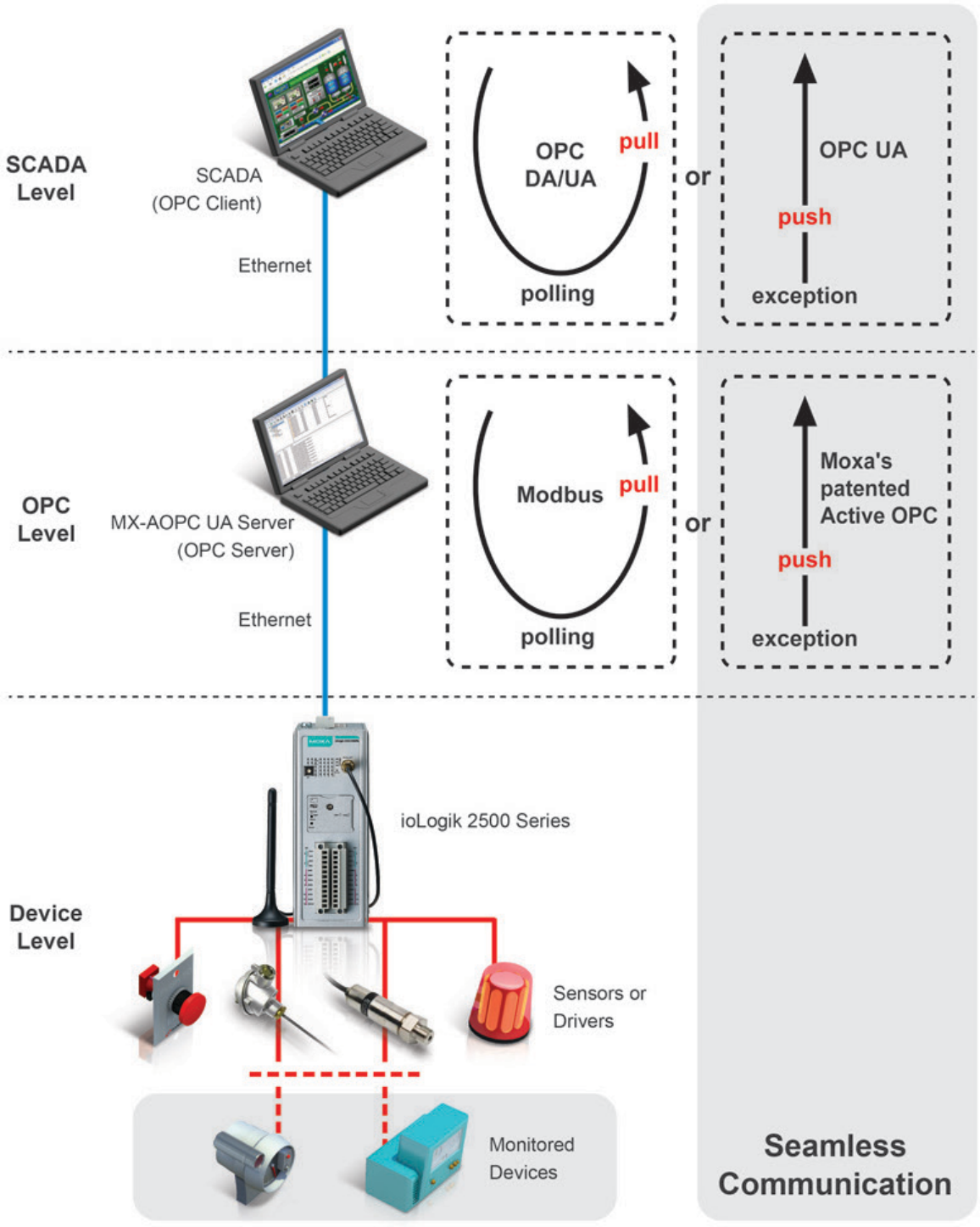

Figure 1: Structure of a modern SCADA system

SCADA software and OPC servers have traditionally been based on a client-server polling model. That is, the SCADA software polls the OPC server, which itself polls the PLC/RTU/ioLogik for current sensor readings, and then the SCADA operator issues commands in response to whatever information is provided by the SCADA software's user interface. Although some readings could be polled more or less frequently than other readings, sensors that monitor critical readings (e.g., whether or not a locked door is open or closed) may need to be polled as frequently as once per second to give operators enough time to take the necessary action (e.g., alert security personnel), and to ensure that the SCADA system is properly notified. For example, if the door's status is polled once every 5 seconds, but the door is opened and then closed within a 4-second time interval, the SCADA system won't even know that the door was opened. If you only need to monitor the status of one door, then frequent polling may not be a problem. However, for SCADA systems that monitor the status of hundreds of doors, frequent polling of so many sensors could occupy a large amount of network bandwidth, and as a result slow down other applications that are connected to the same network.

About ten years ago, Moxa introduced its patented Active OPC concept, which is implemented by Moxa's ioLogik products. Put simply, Active OPC gives dumb I/O devices the intelligence they need to initiate a connection with the OPC server. In other words, since the I/O devices are connected to the ioLogik via local serial connections, the ioLogik can poll these devices as frequently as it likes without putting any burden on the Ethernet network, and only sends readings to the OPC server (over the Ethernet network) when certain pre-configured conditions are met. As illustrated in Figure 1, the action of a traditional client-server polling model is sometimes described as a "pull" (since the OPC server "pulls" I/O readings out of the various devices), whereas the action of ioLogik's Active OPC is described as a "push" (since the ioLogik "pushes" I/O readings from the various devices to the OPC server).

In a more recent development, in 2008 the OPC Foundation standardized a "report by exception" methodology in the OPC Unified Architecture (OPC UA for short). OPC UA uses a "subscription and monitored item" model to control communication between the SCADA software and OPC server. OPC UA is completely new, in that it allows operators to work directly from their SCADA system to configure the way the OPC server interacts with the various I/O devices. In fact, since report by exception "pushes" readings from the OPC UA server to the SCADA software, using OPC UA in combination with Active OPC provides seamless communication by implementing what we could call a "push-push" strategy, which has the potential to save impressive amounts of network bandwidth.

In this white paper, we explain the difference between "updating data by polling" and "updating data by exception," give some general rules of thumb you can follow to decide which method is suitable for your various I/O devices, and introduce Moxa's new MX-AOPC UA server solution.

Updating Data by Polling or Exception

For many years now "updating data by polling" has been the industry standard for communication between the OPC server and OPC clients (i.e., SCADA software). Now, however, engineers can decide between updating data by polling and updating data by exception. Generally speaking, which option to choose depends on two factors: (1) the *frequency* with

which sensor readings change, and (2) the *urgency* with which you need to know that a reading has changed. Sensor readings that change frequently need to be sampled frequently to get a true picture of how the sensor readings change with time. For sensor readings that don't change very often, you could end up wasting quite a bit of network bandwidth if you sample too frequently. But, if you sample too infrequently, you might completely miss critical data (such as that a door has been opened and then closed). Let's look in more detail at how updating works with an OPC UA server.

Updating data by polling

All OPC UA servers still support updating data by polling, with the configuration procedure and method of operation identical to more traditional OPC DA servers.

Updating data by exception

When configured for report by exception, an OPC UA server uses a "subscription and monitored item" methodology in which a SCADA client subscribes to a set of monitored items. The OPC UA server samples the data points at regular "sampling intervals," places the item's readings in a queue, and then publishes the readings at regular "publishing intervals." A critical aspect of this operation is that if a sampled reading has not changed compared to the previous sample, the reading is not placed in the queue. What this means is that *readings that don't change aren't published*, which is the essence of the "report by exception" concept (Figure 2). Note, too, that OPC UA supports sending heartbeat signals during extended periods of inactivity so that each side of the connection will know that the other side is still alive, and consequently will not close the connection.

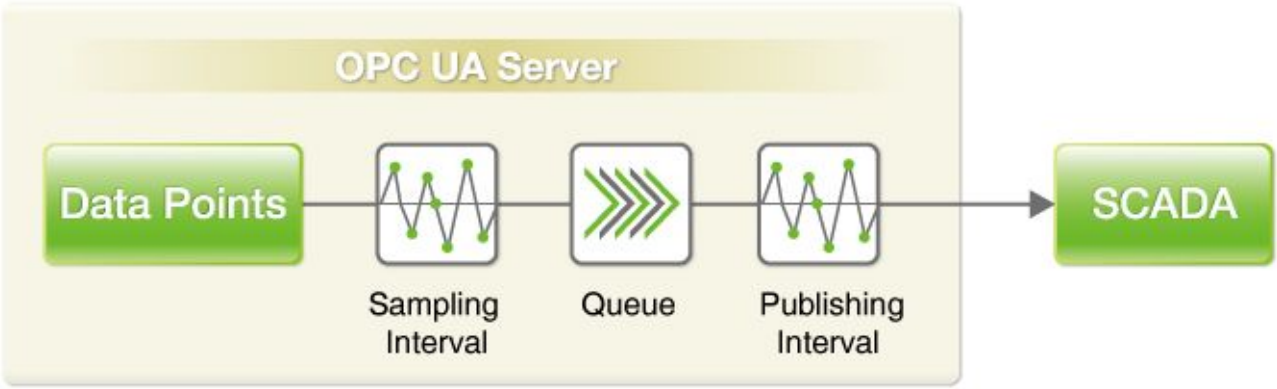

Figure 2: Report by exception using a subscription and monitored item methodology

Two settings need to be configured on the OPC client to enable updating by exception: the sampling interval and the publishing interval (Figure 3). The sampling interval defines the rate at which the server checks for changes in the monitored device readings, and the publishing interval defines the rate at which the server sends notifications to the client. The sampling interval can be shorter than the publishing interval, in which case notifications are queued in the server until the publishing interval has elapsed. At that point, the server sends all of the notifications in the queue to the client.

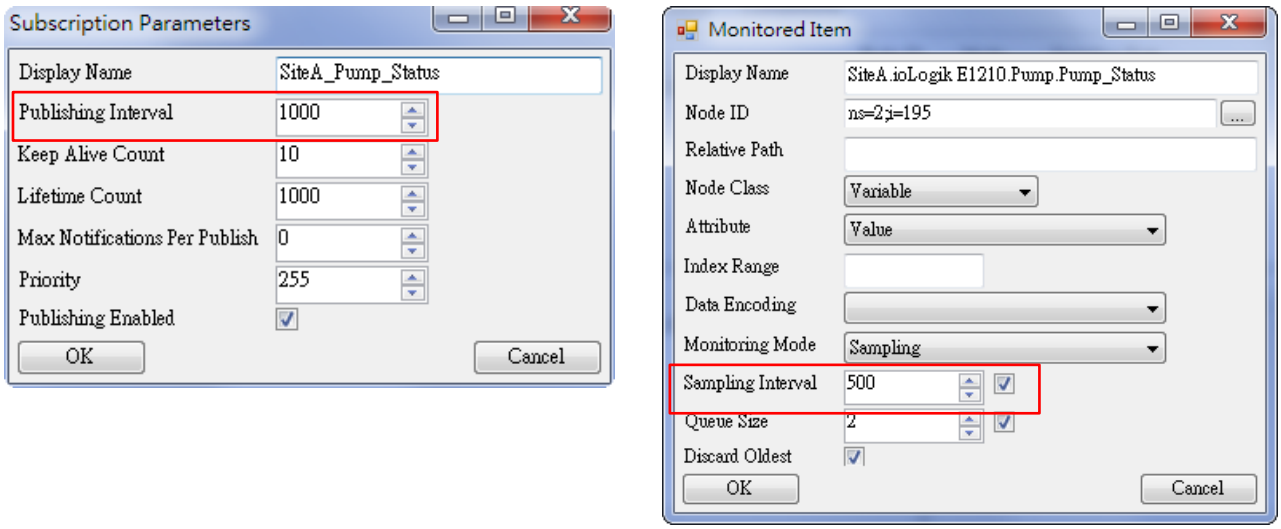

Figure 3: Subscription and monitored item settings for a sample client

With "update by exception," since I/O readings are not transmitted when the monitored system's status doesn't change, operators can greatly reduce the amount of network bandwidth that's required. This is especially true when the frequency of value changes is far less than the polling interval, such as is true when monitoring a door's open/close status. Report by exception also saves computing resources on both server and client computers for handling timeouts and retries.

If the frequency of value changes is higher than the polling interval and urgency is critical, updating data by exception is still the better way to go. However, report by exception may still cause a lot of data to be transmitted in a very short time, which could cause network congestion. The congestion can be relieved somewhat by setting an appropriate "dead band" for analog data, or by trimming down the amount of data with an appropriate numerical processing algorithm before data is sent out. On the other hand, if the frequency of value changes is higher than the polling interval and urgency is not critical (such as when monitoring the temperature of a liquid), updating by polling might be more appropriate.

Most OPC UA servers use a poll-type protocol, such as Modbus, to get data from their I/O devices. However, polling hundreds or thousands of tags is very inefficient. If both polling and exception options are available, you can determine the best approach by first categorizing device tags into one of four types (Figure 4), and then increase the efficiency of your operation by using poll-type methods on the high frequency but non-critical urgency tags to update data to the SCADA system.

Reporting Urgency / Data Changes	Critical	Non-critical
High Frequency	Update data by exception (with appropriate dead band setting)	Update data by polling (with short sampling interval)
Low Frequency	Update data by exception	Update data by exception

Figure 4: Selecting either polling or exception

Configuring Easy-to-Understand Tag Names

Most OPC servers require tag names to start with communication type, such as Ethernet or serial, followed by device name, followed by I/O point name. For example, a tag name for a pump's on/off status might be Ethernet.Device.Pump_Status. However, since the location of the sensor the tag name is associated with is not included in the tag name, and since a single SCADA system might include thousands of tags, it is difficult or impossible for operators to determine which device is being referred to just by looking at the tag name. For this reason, tags are often associated with more detailed descriptions, with the tag names and descriptions organized in an Excel worksheet.

One way to get around this problem is to include the device's location in the tag name by appending it to the device name. To illustrate, suppose the SCADA system uses the same model of I/O device to monitor two different pumps named PumpA and PumpB. If the two pumps are monitored by different I/O devices of the same model, you could write the tag names as Ethernet.Device_SiteA.Pump_Status and Ethernet.Device_SiteB.Pump_Status to differentiate between the two.

But why should tag names start with communication channel? If the tag names are based on the actual application architecture, it would be easier for users to construct the tag names. The difference in tag naming strategy is illustrated in Figure 5. The diagram on the left is less intuitive, and could get rather messy since if Site A also uses serial devices, then "Site A" would also appear under the Serial branch. The diagram on the right shows the same system organized by the application architecture. In this case, all devices at Site A will appear under the Site A branch, and the tag names could be written as SiteA.Device.Pump_Status and SiteB.Device.Pump_Status. These tag names are more readable, and make it easier to configure your SCADA system.

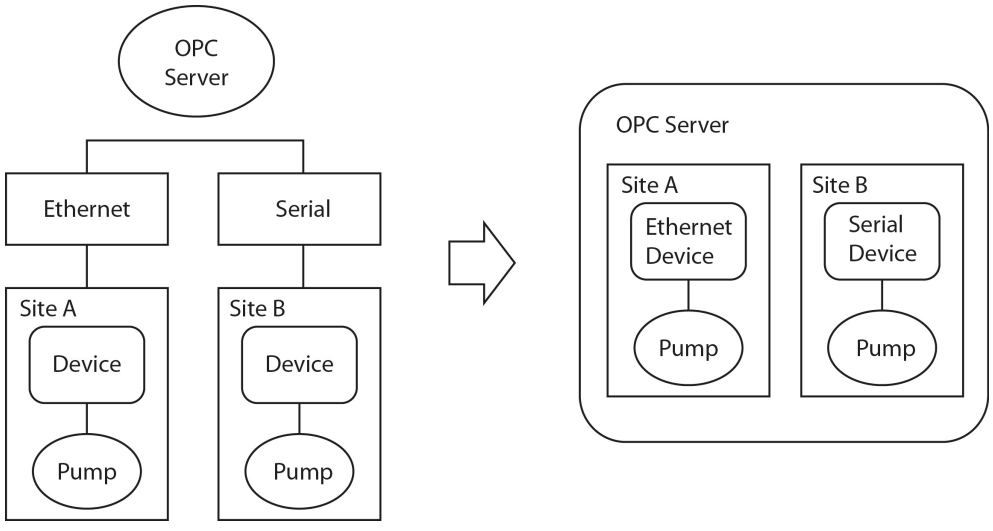

Figure 5: Changing from communication channel to application for tag naming

OPC UA Makes it Easier to Connect Servers and Clients

Configuring OPC to work between the server and the client on different computers was a real headache before the OPC unified architecture was available. For example, the user had to log in with the same account and password on both the server and client computers, which can be extremely inconvenient from a practical point of view. In addition, the user needed to follow detailed unintuitive step-by-step instructions to configure DCOM security.

In contrast, OPC UA uses an optimized TCP-based UA binary protocol for data exchange, in which communication can be activated by opening up a single user-configurable port in the firewall, as illustrated in Figure 6. Users can create many TCP URLs for OPC server endpoints, with each endpoint mapping to a unique port. OPC UA clients only need the URL of the server endpoint to connect to the OPC UA server.

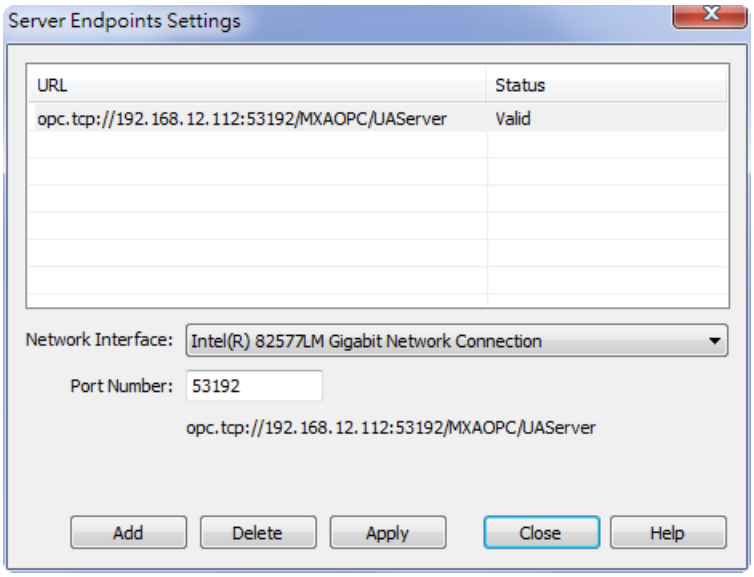

Figure 6: Server endpoints settings from Moxa MX-AOPC UA server

Integrated security mechanisms such as X509 certificates ensure secure communication on the Internet. Users can define security policies such as "Sign and Encrypt" between the OPC UA client and server (Figure 7).

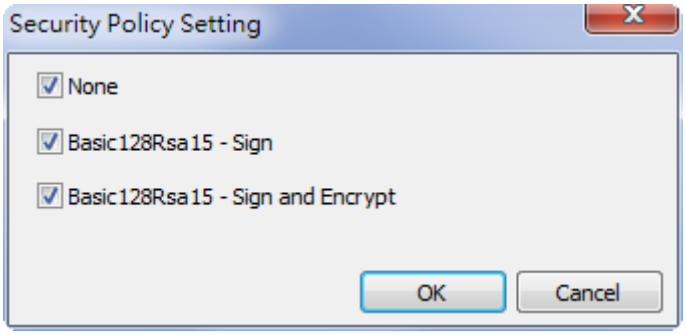

Figure 7: Security policy setting on MX-AOPC UA server

Users only need to import the client's "Certificate Authority" file from the OPC UA client and export the server's "Certificate Authority" file to OPC UA clients to establish authority between the server and client (Figure 8).

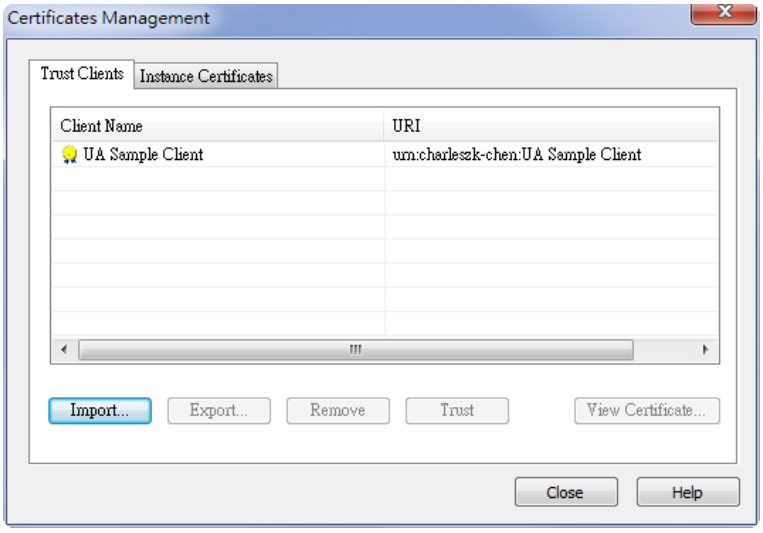

Figure 8: Certificate management

Then, the "Discover Servers" function can be used in the OPC UA client to discover OPC UA servers accessible over the network (Figure 9).

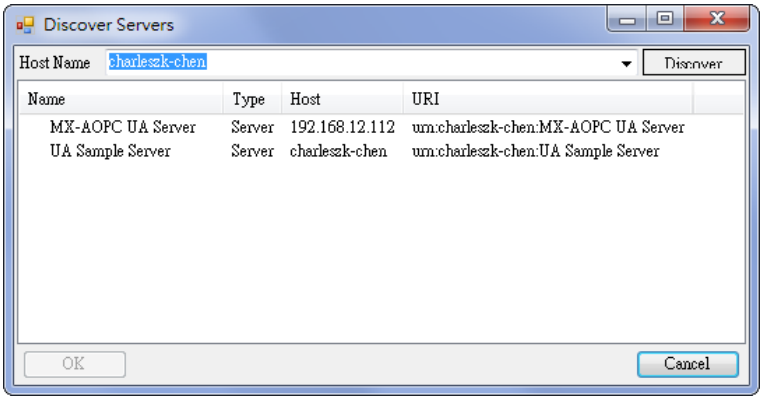

Figure 9: Discover Servers window from UA sample client

Finally, users can select the TCP URL to connect to the OPC UA server (Figure 10).

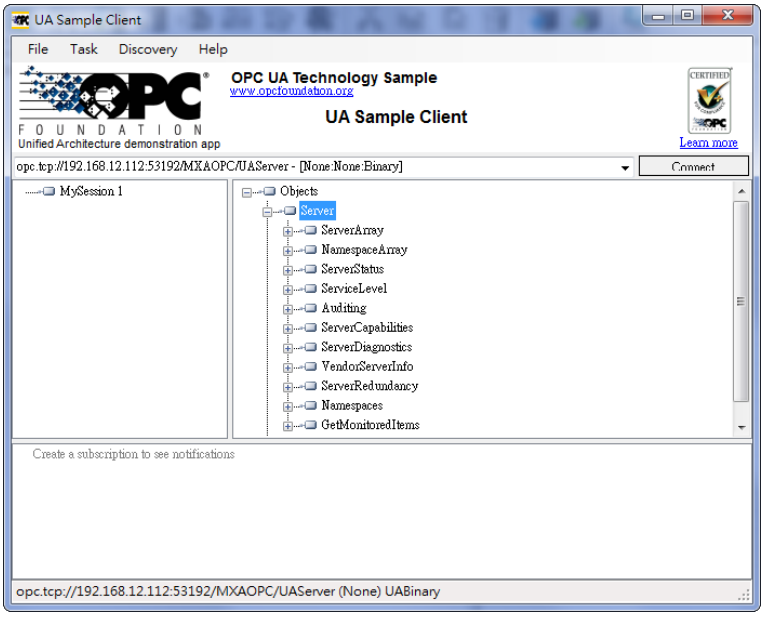

Figure 10: Connecting to the server

Moxa's MX-AOPC UA Server Solution

MX-AOPC UA Server expands on Moxa's patented "Active OPC" monitoring technology, incorporates support for Modbus protocol, and provides a secure and reliable gateway between local devices and a remote SCADA system. Moxa pioneered "push type" I/O processing (as opposed to "pull type" or simply "polling") in the automation industry with the release of its Active OPC Server. The patented MX-AOPC UA server offers both a polling and non-polling architecture alongside the standard OPC UA protocol, giving users the choice of pull or push-based communication with Moxa devices (Figure 11).

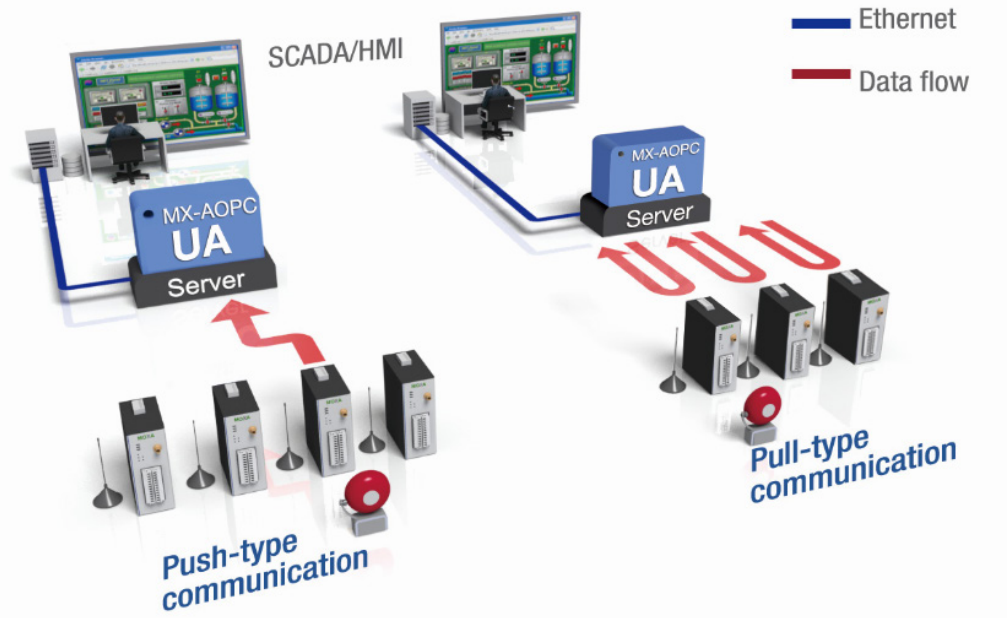

Figure 11: Choice of push or pull type communication

MX-AOPC UA Server's design logic is user-application oriented. As can be seen in Figure 12, users can create device groups, "SiteA" and "SiteB" for example, based on their application. In the example shown here, each site uses the same ioLogik E1210 unit to monitor pump status.

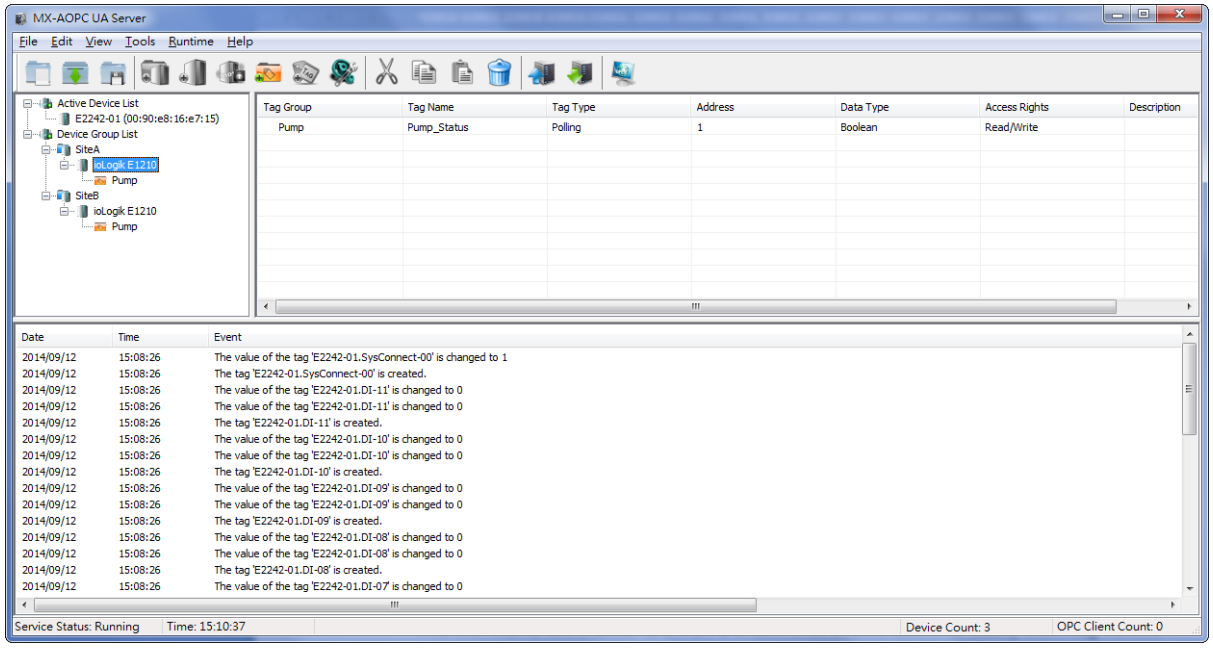

Figure 12: Application based device groups

Tag names (Figure 13) are much clearer and more readable when it comes time to configure your SCADA system.

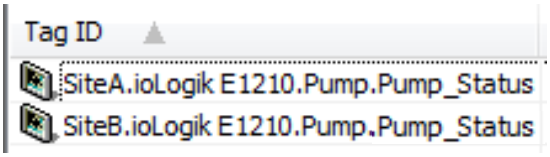

Figure 13: Clearer tag names

Moxa's Broad Selection of Data Acquisition Products

Moxa provides a wide array of reliable industrial data acquisition solutions, including easy-to-use software, for general industry use. Click here for details, and to see how Moxa's data acquisition solutions can benefit your business.

ioLogik 2500 Series
Smart remote I/O with Click&Go Plus Logic

ioLogik W5300 Series
Smart cellular remote I/O with Click&Go Plus Logic

ioLogik E2200 Series
Smart Ethernet Remote I/O with Click&Go Plus Logic

MX-AOPC UA
Automation software

References:
https://opcfoundation.org/about/opc-technologies/opc-ua/
https://opcfoundation.org/developer-tools/developer-kits-unified-architecture