Extract from the book:

OPC Unified Architecture

Mahnke, Wolfgang, ABB Corporate Research
Leitner, Stefan-Helmut, ABB Corporate Research
Damm, Matthias, ascolab GmbH

14 Conclusion and Outlook

14.1   OPC UA in a Nutshell

OPC Unified Architecture (OPC UA) is the new standard for data communication in process automation and beyond, provided by the OPC Foundation. It is expected that OPC UA will replace the very successful Microsoft-DCOM-based specifications of the OPC Foundation (DA, HDA, and A&E) over the next few years as OPC UA unifies all the functions provided by those specifications. Because of its platform-independence and use of state-of-the-art Web service technology (see Chapter 6) it is expected that OPC UA will be applied in an even wider range of industries and applications, compared to classic OPC. It can be deployed on devices, DCS, MES and ERP systems. The small set of easy-to-use services (see Chapter 5) allows accessing the unified address space in a reliable and secure manner (see Chapter 7). By using binary encoding on the wire OPC UA is a high-performance solution, significantly faster than XML data exchange (see Chapter 14).

OPC UA not only addresses data communication but also information modeling (see Chapter 2). With its rich address space model, it allows high-value metadata exposure and thus provides significantly more information than before. For this purpose, OPC UA uses object-oriented concepts and allows a full-meshed network of nodes related by multiple types of references. There is a high interest in these capabilities in many domains and there are already projects to standardize information models based on OPC UA. Examples of such activities are FDI where a common field device description is targeted and common activities with PLCopen (Industrial Control), MIMOSA (Maintenance Information – ERP and above) and S95 (Production Information – MES) (see Chapter 4).

With its profiles (see Chapter 13) OPC UA scales well from small servers to highly sophisticated systems. Small servers only providing simple functionality are able to run on limited hardware, exposing only a small set of simple data. Highly sophisticated servers are able to expose a large amount of complex information and to support complex functionality like querying the address space (see Chapter 10).

Nevertheless, some people are complaining that "Everything is so complicated" in OPC UA. Therefore, in the next section we will take a look at this objection against OPC UA. Finally, there is an outlook examining how OPC UA may be applied in the market and what is missing in it, to improve it even further.

14.2   Is OPC UA Complicated?

Over the last couple of years we had several discussions about OPC UA with people from different domains, backgrounds and, of course, different companies. Most of them were excited about the power and possibilities that OPC UA offers. However, there were a few people complaining about the complexity of OPC UA. A quick and simple answer is that OPC UA is very simple for users of base functionality. For users of advanced features, it is not complex but as simple and generic as possible, and still very powerful. Of course this answer does not really help to convince people. Thus, in the following sections, we explain the features involved in several different places in OPC UA, why it is designed in the way it is, and what this means for people actually using OPC UA. The management summary is given in Section 0.

14.2.1   Are OPC UA Services Difficult to Handle?

Looking at the OPC UA services, you can see that actually the number of services is very small. OPC UA has only 37 services, of which three services deal with discovery and six with connection handling. That leaves 28 services to actually access OPC UA data. Let us compare that with the old and very successful OPC DA specification. This specification deals only with current data, not events, history or a rich information model and thus deals only with a subset of functionality provided by the OPC UA specification. Nevertheless, the old OPC DA specification had nearly 70 methods.[1] That shows that the OPC UA service framework is designed for simplicity. The intention was not to provide two services offering the same functionality in a different manner.[2] Thus OPC UA does not offer several services with browsing functionality but one Browse service that allows the setting of filters on References, NodeClasses, etc. and specifying what information should be returned, such as the name or the type of information of the referenced node.

OPC UA services are designed in a service-oriented manner, always providing bulk operations. For example the Call service does not call a single method but allows calling a set of methods with one service call. That design principle reduces the number of roundtrips for a set of operations and is a common feature in ser-

---

[1] The OPC DA specification does not use a pure object-oriented design but supports bulk operations instead of simple methods and thus the numbers are comparable.

[2] There are some minor exceptions from that rule. For example, the Read service and the subscription mechanisms both provide access to actual data. However, the use cases are very different and thus the simple Read only reading a value once and the subscription requiring some setup first and then getting changes of the value are both supported.

vice-oriented architecture. It is also used in object-oriented APIs like the OPC DA specification and a reasonable compromise between simplicity and performance.

There are three concepts in the OPC UA service framework that can be considered to be complex: First of all the query capabilities of OPC UA, second the publish mechanism of OPC UA, and third the connection establishment.

In the first case the complexity is inherent to the provided functionality. Complex queries are complex in SQL [ISO08a] or OQL [CBB+00] as well. Queries are an optional feature in OPC UA and many servers will not support it and it does not even make sense to support it in many scenarios where the amount of data in the server is small and browsing the address space is the best way to deal with the data. However, there will be OPC UA servers having a huge address space with millions of nodes and in that scenario the querying capabilities become a requirement to efficiently find data in the server. But a complex problem cannot be solved without any complexity.

The second complex concept of OPC UA is the publish mechanism. The publish mechanism allows the logical callback to asynchronously send notification messages to a client containing data changes or event data without establishing a real backward channel from the server to the client. The main reason for designing the publish mechanism in that way was that OPC UA potentially runs in a Web Service environment, connecting clients and servers over the Internet or intranet, having firewalls between them. In that environment, it is easy to "talk" from a client to a server but often impossible to "talk" from the server to the client (that would require the server to be a client and more important the client to become a server). Introduced as a second method to do callbacks, the OPC UA working group found that the publish mechanism is always as good as a real callback mechanism, considering the additional requirement of sending keep-alive messages and sequence numbers to have a reliable communication in an unreliable environment. Thus the callback approach was discarded and only the publish mechanism is part of the OPC UA specification as it provides the same functionality as a real callback. In addition, the callback mechanism would highly increase the complexity regarding security mechanisms, as another secure connection has to be established from the OPC UA server to the client. Here again the simplicity of the OPC UA service framework can be seen: only one method is available for one purpose. However, it requires some time to truly understand the publish mechanism. The good news is that only a very small number of people really have to understand the mechanism. Most people will use a server or client SDK that deals with the mechanism and provides real callbacks internally.

The third complex concept in OPC UA is connection establishment. This step requires establishing a secure channel. On top of the secure channel a session has to be created. The secure channel provides security on the transport level, which

means that messages can be encrypted and signed.[3] OPC UA uses WS-SecureConversation as part of the WS-* standard [OASIS07] for its secure channel when SOAP messages are exchanged in the Web service world and adapts this specification to UA-SecureConversation when the high performance UA TCP is used. Details of the technologies used for secure channels are given in Chapter 7, including why no standard protocols like TLS/SSL can be used, based on requirements for OPC UA, such as having long-running connections, etc. On top of such a secure channel, multiple sessions can be created, decoupling the secure communication from the session management on the application level. The described steps are very common and a good security design in various ways. Therefore they are necessary for ensuring a secure and reliable communication between clients and servers. The good news is that again only a few people really have to deal with it, as an OPC UA SDK will provide a connect method that hides the handshaking to establish a connection.

To summarize the discussion:

1. The OPC UA service framework (measured by the number of services) is very simple.
2. OPC UA services are designed for bulk operations to avoid roundtrips. This increases the complexity of the services but greatly improves the performance. Also, classic OPC has been designed in a similar way for most methods. Handling of bulk operations is commonly used and thus it is not "too complex to be used".
3. OPC UA queries are complex; however this is part of the addressed problem. OPC UA queries are an optional feature, useful only for large address spaces.
4. The publish mechanism of OPC UA is required in environments where the OPC UA client cannot act as a server (firewall). Using only this mechanism reduces the complexity of OPC UA (compared to adding a real callback, especially if security is considered). OPC UA SDKs will hide the mechanism and provide real callbacks anyway.
5. The connection establishment in OPC UA uses proven security mechanisms and adapts them to the needs of OPC UA. Thus some messages have to be exchanged, but this complexity will be hidden by an OPC UA SDK offering only a connect method.

Thus OPC UA services are not complex but very simple with regard to the provided functionality and the addressed non-functional requirements such as security and reliability. Using an SDK will further hide the complexity in the services, for example, by hiding the connection establishment and the publish mechanism.

---

[3] Security has to be implemented in certified OPC UA products (see Chapters 7 and 13). Whether security is enabled in a concrete installation depends on the configuration based on the security requirements of the installation.

14.2.2   Is Information Modeling a Pain?

OPC UA does not only standardize the data communication but also provides a meta model allowing standardized information models built on top of it. The old OPC DA specification provided a very simple but limited way to expose data items in a hierarchy. OPC UA supports the same simple approach to build a hierarchy of data variables but it also allows exposing rich models.

There are other specifications like DSSP [MS07] or DPWS [MS06] that only standardize the data exchange without specifying a model. They may define a fixed or extendable set of operations that a server provides and a client can call. The client can ask for the meta data of the operation, i.e., what data the operation will provide. However, there is no fixed syntax[4] and semantic for the data exchanged. Obviously those specifications look less complicated at first glance as they exclude the modeling capabilities. But let us take a look at Fig. 14.1 and Fig. 14.2 to see the implications of those different approaches. With a standardized protocol for data communication, there is interoperability between applications on the communication level (right side of Fig. 14.1). Here, no point-to-point integration is needed as in the left side of Fig. 14.1.
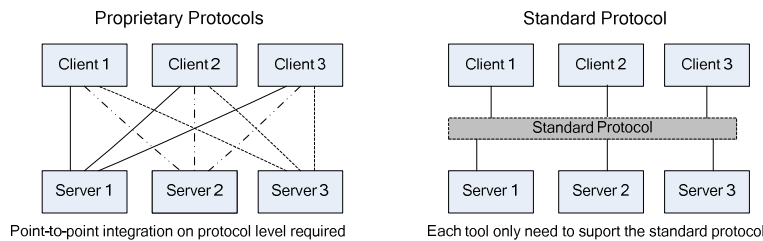
Proprietary Protocols   Standard Protocol

| Client 1 | Client 2 | Client 3 |

| Server 1 | Server 2 | Server 3 |

Point-to-point integration on protocol level required

| Client 1 | Client 2 | Client 3 |

Standard Protocol

| Server 1 | Server 2 | Server 3 |

Each tool only need to suport the standard protocol

Fig. 14.1 Interoperability on protocol level

But if no meta model is provided and the structure of the exchanged data depends on a concrete application,[5] there is no interoperability on what data are exchanged, and therefore no interoperability at the model level (left side of Fig. 14.2). Clients cannot generically interpret the data provided by the server and cannot generically provide data to the server.

---

[4] Other then XML without an XML-Schema.

[5] And of course a client also has to send data to the server in a format the server expects.

Standard Protocol and Proprietary Model   Standard Protocol and Standard Meta Model
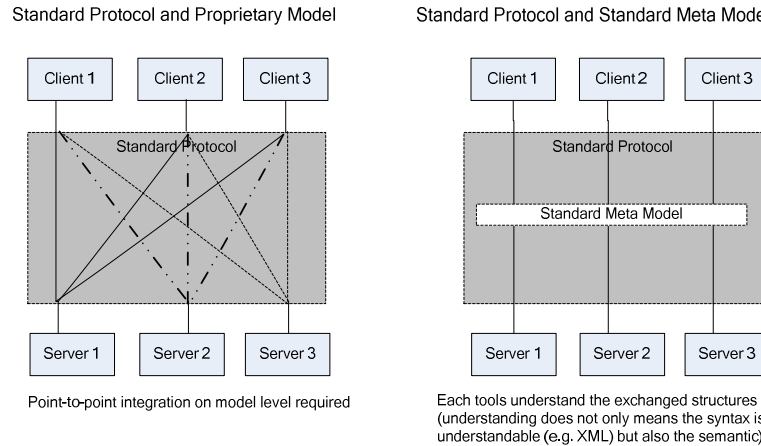

Fig. 14.2 Interoperability on model level

OPC UA offers a solution shown in the right side of Fig. 14.2. It provides the data in a way that generic OPC UA clients are able to deal with all the data (subscribe to data, browse the address space, etc.). However, OPC UA still allows servers to define their domain and vendor-specific model, based on the standardized meta model. The services are all based on the meta model and thus a generic client can deal with all the data. But not all semantic of the concrete model is directly captured in the meta model since it is an extensible model and not a concrete model tailored to one specific domain.[6] However, the meta model provides all information about the extensions and thus a generic client can easily display all semantic information, but some of them have to be interpreted by a user. Clients can be implemented with built-in knowledge of concrete OPC UA Information Model Standards and thus no additional interpretation by the user is needed.

Let us take a quick look at the implications for client and server developers, whether a meta model is provided or not. Obviously, having a common model makes life much easier for client developers as they can implement their clients with knowledge of one model supported by all servers. Otherwise they would have to deal with several servers providing different data structures. However, leaving the data structures undefined seems like a great opportunity for server developers. They have complete freedom to do whatever they want to do. But they also have all the work to do. It took the OPC UA working group several years to develop the meta model that is able to access and change real-time related data, alarms and events, and the history of both. Of course a specific server does not have to provide a generic meta model as OPC UA has to do, but there is still a lot

---

[6] The OPC UA meta model is tailored to the broad domain of exchanging real-time related data, including events and history, but not to a concrete domain like drilling or pulp and paper.

of work to do and many pitfalls to avoid. And after creating their own model they have to document it, so that clients are able to get access to the data and users understand at least a basic semantic so they are able to deal with the provided data and the ways to access them. Looking at all those tasks, mapping the data to a well-defined meta model and enriching the server to a concrete model using the extension mechanisms does not seem so complicated anymore.

After explaining the need for a meta model in OPC UA there is still the question of its complexity. The classic OPC specifications, for example, only provided one[7] concrete and very simple model. The reason for having the more powerful and more complex model in OPC UA is that it allows exposing much more semantic and thus much more information. Since OPC UA does not target one concrete domain, it allows defining information models for concrete models tailored to specific domains. Therefore we have typed objects and references, etc. To expose more information in a usable way more concepts have to be introduced and this increases the complexity. However, the complexity is found in the information provided additionally. If no additional information needs to be provided, the OPC UA model becomes very simple. For example, the OPC DA wrapper of the OPC Foundation wraps any OPC DA server. Since the OPC DA server does not provide complex information, the OPC UA model of the wrapper is very simple (see Chapter 11). It contains only one hierarchy with OPC UA Folder Objects for the OPC DA Branches and OPC UA Variables for the OPC DA Items. OPC DA properties are provided as OPC UA Properties. No detailed type information (all Objects are of the same type), no additional ReferenceTypes, no ModelChangeEvents tracking changes on the model, no other fancy stuff is to be found. It is optional to use those features and it does not make sense to use them in many cases. However, if you need to provide more information, it is very useful to have those concepts in place so you can use them appropriately. As it is the choice of the server to use features regarding the modeling, it is the choice of the client to decide what features it wants to use. Programming against types is a very powerful feature and very helpful when creating process graphics or other user interfaces specific for certain OPC UA types. However, a simple client providing only browsing capabilities will never use this feature.

Information modeling can be done very easily, keeping everything very simple. Then little effort is required and using some very basic OPC UA modeling concepts keeps everything very simple. The OPC DA wrapper is a good example of that approach. But if more information must be provided OPC UA is equipped with the necessary concepts to allow it. The complexity is based on the information to be modeled, not in OPC UA itself.

---

[7] Actually one per specification.

14.2.3   Transport Protocols and Encodings: Why So Many?

OPC UA defines an abstract set of services that is mapped to different technologies. Currently there are two protocol mappings and two encodings supported. The reason for having abstract services is that, if a new technology for data communication enters the stage, OPC UA can be adapted to that technology just by defining another mapping. But why does OPC UA support two protocols and two encodings from the beginning? The reason is that OPC UA will be applied in different application domains with different requirements. Supporting HTTP and UA TCP (see Chapter 6) allow it to run Internet applications crossing firewalls with HTTP as well as running optimized applications with limited resources via the UA TCP protocol, which is optimized for the wire (no overhead) and the needed resources (no HTTP stack needed). But the main optimization on the wire is not UA TCP versus HTTP but exchanging binary encoded data versus XML encoded data. Unlike other protocols, OPC UA does not require the data to be first converted to XML and then binary encoded; the data is directly binary encoded and is thus very efficient. However, there are applications that do not need high performance but data provided in a generic way (which nowadays means in XML) to be able to handle them easily. Those applications are typically placed not on the bottom, but the top of the automation pyramid. Therefore OPC UA supports XML encoding as well. Fig. 14.3 summarizes the use of different protocol options in a simplified form.
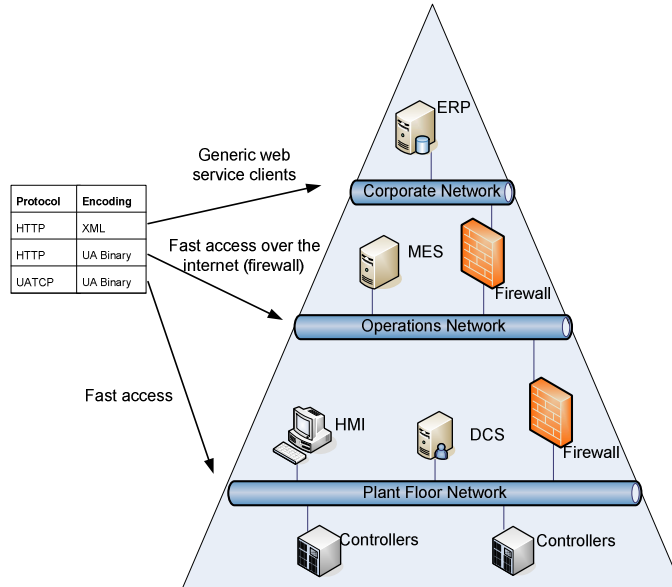

Protocol	Encoding
HTTP	XML
HTTP	UA Binary
UATCP	UA Binary

Fig. 14.3 Simplified view of OPC UA transport protocols and encodings

After understanding why OPC UA supports different technology mappings, we will discuss how this affects people dealing with OPC UA. The good news is that anybody using an OPC UA SDK or even only an OPC UA stack provided by the OPC Foundation could not care less about that. The OPC UA services stay the same and thus the server or client implementation stays the same, no matter what the stack uses to communicate with another stack (see Chapter 9 for details). Only those people developing stacks or using generic toolkits, for example, to generate clients talking to Web services (e.g., by a WSDL) are affected by supporting different protocols and encodings. It is the job of stack developers, so we will not argue about that. Generic toolkit users bound to a technology like Web services are bound to the technology provided by the toolkit. But those users choose to use that generic toolkit (which is of course reasonable in some scenarios) and therefore they are intentionally bound to that technology. This leads to potential interoperability problems between OPC UA servers and OPC UA clients supporting different technologies. But here again OPC UA profiles are a good mechanism to avoid that and new technologies are not provided so often that one must fear an explosion on OPC UA technology mappings.

14.2.4 Implementation Issues

A few people have complained about the lack of comfort of OPC UA SDKs. This is of course something we cannot answer in general but it depends on the used SDK. As a general statement, people have to be aware that OPC UA is a new specification and thus people are dealing more with developing initial products than with providing a perfect SDK. The comfort of OPC UA SDKs will increase in future.[8] However, the SDKs we are aware of already support the developer quite well by implementing all the housekeeping functions of the OPC UA service framework.

Let us take a look at server-side development. You do not have to deal with subscriptions; you only have to provide the data that are to be published. You do not have to deal with queue management, republishing, or packing of data. There is infrastructure to manage OPC UA nodes in the server. Thus you only have to configure the Address Space of your server. This can be done by an XML configuration where most of the code is generated.

On the client-side, you get real callbacks, you can connect by one call, etc. So there is already a lot of comfort in the SDKs we are aware of. However, there is still room for improvements, e.g., fancy wizards, graphical modeling tools, etc.

---

[8] This is a general statement true for all SDKs in all domains. The comfort and quality of an SDK always increases from very early versions (1.0 or even less) to higher and more stable versions.

14.2.5 Migration of Existing Code

For somebody with classic OPC servers and clients it might seem like a large amount of work migrating to OPC UA. Since the OPC Foundation provides wrappers for the classic OPC servers as well as proxies for the classic OPC clients, this is obviously not true if you do not want to change your existing code. You simply deploy the provided proxies or wrappers with your existing product and you are ready for OPC UA. However, let us take a quick look at what you need to do for native OPC UA support in existing products. As OPC DA is the most popular classic OPC specification, we will examine what to do in that case.

In the case of an OPC DA client, it is easy to replace the existing code for reading, writing, and adding groups and items to receive data changes with similar concepts from UA, such as subscriptions and monitored items. UA client SDKs will provide data change callbacks on top of a subscription hiding the publish mechanism. The biggest change is the handling of NodeIds instead of pure string based ItemIDs in old OPC DA. For the configuration part the browsing and property access methods from OPC DA can be replaced with the UA browse service calls. This is all the functionality most OPC DA clients use today.

In the case of an OPC DA server, there are only a few services that need to be implemented like Read, Write, Browse and the delivery of data changes to a UA SDK. All other services are implemented by a UA SDK or are not needed for supporting DA functionality. Providing the required data model is also very simple as exposing a pure DA address space with no type system uses only a small number of predefined types of OPC UA.

As the OPC UA design is generic and extensible, it is easy to choose an iterative development approach to add UA features over time to a product, starting with a pure DA implementation. For example, to add OPC UA Method support, a product must implement one more UA service and one more NodeClass to expose the Methods in the Address Space.

Let us assume you plan to migrate your products to a higher-level programming language like JAVA or .NET. Using classic OPC requires you to deal with the interop from COM to the modern programming language. This can become a real problem when you deal with multiple threads, the life-cycle of COM objects, etc. [Ge03]. Instead you can directly target OPC UA as communication interface. Here, your product can use UA SDKs and stacks natively developed for those programming languages. Thus, the new code is separated from any COM-based code. To connect to classic OPC products, you can use the wrappers and proxies provided by the OPC Foundation. This is exemplified in Fig. 14.4, where a C++-based OPC DA client is migrated to .NET and uses OPC UA as the new communication infrastructure.

Scenario before migrating code

OPC DA (COM/DCOM)

OPC DA Server

OPC UA
Proxy

OPC UA

OPC UA Server

C++-based OPC DA Client

Scenario after migrating code

OPC UA
Wrapper

OPC DA Server

OPC UA

In this environment
COM is not needed
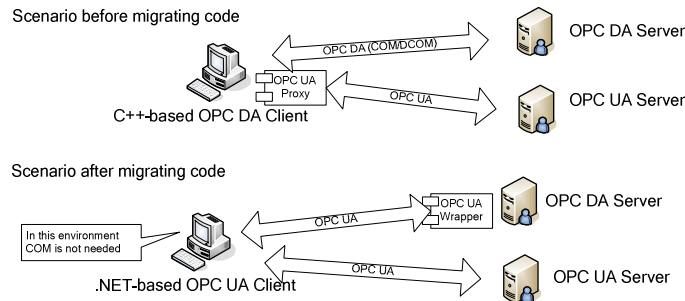
OPC UA

OPC UA Server

.NET-based OPC UA Client

Fig. 14.4 Migrating classic OPC applications to modern programming languages using OPC UA

To summarize this section, OPC UA gives you a reasonable migration strategy to move a classic OPC environment to OPC UA in different levels. The lowest level is to use the proxies and wrappers; the next level is to expose the same level of information you expose today with UA and the next level is to add additional UA features like Methods or a type system over time. If you want to migrate your code to a modern programming language, using OPC UA provides you a solution that does not require you to deal with all the COM interop problems and allows separating your new code from COM.

14.2.6 Management Summary

OPC UA is just as complex as it has to be, to fulfill the requirements of a secure and reliable communication, able to run in different environments including different networks separated by firewalls. The binary encoding provides high performance data exchange. Unlike other protocols OPC UA defines a meta model and thus not only provides interoperability regarding the protocol but also regarding the exchanged data. By defining an extensible base model with all the information necessary to know what data have to be exchanged but still allowing refinements and extensions to the model OPC UA is a well-suited compromise for a specification applied in various domains. Information model standards based on OPC UA define a more specific model tailored to the domain that is extended by vendor specific information. Generic OPC UA clients can easily access all this information. OPC UA profiles allow servers to be scaled from small servers with limited functionality able to run on limited resources to highly sophisticated servers providing a large amount of complex data with the full power of OPC UA.