General OPC UA considerations

Address space considerations

One of the main selling points with OPC UA is its information modelling capabilities through which it is possible to present the data a system is providing so that other systems are able not only to access the data but also to get information on how to interpret the data. In practice each device that provides data or services through OPC UA exposes one or several address spaces which contains both the data/services and meta-data describing them. The address spaces consist of different classes of nodes that are connected by different types of references. Using nodes and references it is possible to build very different looking address spaces. It is possible to build a flat address space listing a number of plain values, very similar to what a set of modbus registers. On the other hand it is also possible to create an address space containing multiple hierarchical levels where the every piece of data is described from several different aspects. OPC UA also recognizes so called "Companion Specifications" which extends the standard and controls the way how the address spaces are built in a certain domain area. For the interoperability between systems and possible shift towards plug and play the information model is of great importance and therefore also the address space structure needs to be planned well. Due to resource limitations we did not have the possibility to study this in detail during the PoC, but nonetheless we gathered some loose observations which we think should be considered when planning the information model for our use.

Namespaces and NodeIDs

In OPC UA every node of a server can be uniquely identified by the combination of its NodeID. A NodeID consists of three parts; the NamespaceIndex, the IdentifierType and the Identifier. The NamespaceIndex is the index that a server use for representing a certain NamespaceURI, which is the unique identifier for a namespace. The NamespaceIndex itself is not persistent, so when configuring a client the URI should be favored. The identifierType describes the type of the following identifier. There are four types of identifiers, of which the most relevant are numerical and string. Regardless of which type it is strongly recommended to keep the identifiers persistent. In this way the combination of NamespaceURI and identifier can be configured to a client as the key for a specific piece of data. The big advantage with numerical identifiers is that they are lightweight both to store and to handle in an embedded system. The advantage with string identifiers is that they provide a simple way to ensure unique identifiers that can be persisted even if nodes are configured dynamically. For example in the UNIC system, the ISO codes are already unique and could be directly used as identifiers. But as said, handling string identifiers causes more CPU load both on the client and on the server side. In practice we saw this as a distinct peak in the CPU load of our COM-10 based OPC UA server whenever connecting a client and reading many nodes in the address space. Most PLC based OPC UA servers we tested used long string identifiers which were constructed from the browse paths of the nodes.

Address space creation

One of the tasks when starting a OPC UA server is to create the address space. This can be done statically using a configuration prepared on beforehand, dynamically using some logic that exposes selected internal data of the system or using a combination of these methods. The standard specifies an XML format that is used to facilitate the configuration type of address space creation. There are also software tools that can be used to model address spaces and generate standard format XML files. Some servers are able to parse these XML-files during startup while others rely on that the modeling tools generate compilable source code that creates the runtime model of the address space. Of the three SDKs we tested in our PoC, only the Matrikon stack had inbuilt support for generating the address space directly from an XML file. Unified Automation provides and own address space modeler that generates source code that is compatible with the SDK while the open62541 SDK includes a tool that generate source code out of an XML. All three SDKs support dynamic creation of the address space.

In a workflow like the one we use with UNIC, where the system is customized by configuring rather than by compiling a tailor made solution it is in practice not possible to use generated source code to create the address space. With the current workflow, where the interface needs to be provided long before the configuration work is started, the fully dynamic approach is also problematic. Most probably we will need a combination of both methods, where some parts of the interface are static and can be generated even using generated source code while other parts are dynamically created.

Configuring clients

The simplest way of configuring clients is to provide them with NamespaceURIs and identifiers, either manually or using some kind of lists. There are even commercial clients where this is the only option. Another way is that no NodeIDs are stored in the client configuration but instead the BrowsePaths to the variables of interest. Using these BrowsePaths the client is then able to navigate the server address space and find the relevant NodeIDs. Our PoC implementation of an OPC UA client used this method, in the configuration we listed BrowsePaths that the client then browsed when connecting to a server.

As noted above, the NamespaceURIs and identifiers of a certain node should remain unchanged for a certain system. Hence, browsing through the namespace of a server every time connecting to it is an unnecessary step. From what we saw in the PoC, most clients use NodeIDs directly to query for information when connecting to a server, and thus they have that information stored in their configuration. More sophisticated configuration tools for clients are generally able both to browse an online server and import standard XML files. From information gathered using one of these methods they are able to construct a graphical representation of the address space. A user of the configuration tool is then able to navigate this representation and select nodes to add to the configuration.

Unit systems

Providing information about the units of the values provided over a communication link is one of the key parts on the path towards plug and play communication. OPC UA provides a standard way of providing this information as properties of analog type of variables. The actual units are not specified in any OPC UA specification, but OPC UA recommends using the “Codes for Units of Measurement” (see UN/CEFACT: UNECE Recommendation N° 20, http://www.unece.org/tradewelcome/un-centre-for-trade-facilitation-and-e-business-uncefact/outputs/cefactrecommendationsrec-index/code-list-recommendations.html).

The use of units to describe variables is strongly advised in OPC UA, but still it is not mandatory. Our experience from the PoC is that units are generally not used in the PLC-world.

Some of the units used internally in UNIC are not found on the lists of units provided by CEFACT (such as millidegree, decicelsius). Below a table with comparison:

UNIC	UNECECode	OPC UA UnitId	DisplayName	Description
bar	BAR	4342098	bar	bar [unit of pressure]
C	CEL	4408652	°C	degree Celsius
rpm	RPM	5394509	r/min	revolutions per minute
kW	KWT	4937556	kW	kilowatt
s	SEC	5457219	s	second [unit of time]
%	P1	20529	% or pct	percent
-	1	-1		
V	VLT	5655636	V	volt
g/kg	Not found			
mbar	MBR	5063250	mbar	millibar
kW/s	Not found			
m/s	MTS	5067859	m/s	metre per second
ppt	NX	20056	‰	part per thousand
MJ/kg	JK	19019	MJ/kg	megajoule per kilogram
mHz	Not found			
Hz	HTZ	4740186	Hz	hertz
pptt/s	Not found			
deg	DD	17476	°	degree [unit of angle]
ddeg	Not found			
mm	MMT	5066068	mm	millimetre
cSt	Not found			
g	GRM	4674125	g	gram
mg/kWh	Not found			
mg/s	F34	4600628	mg/s	milligram per second
h	HUR	4740434	h	hour
min	MIN	5065038	min	minute [unit of time]
us	B98	4340024	µs	microsecond
kNm	B48	4338744	kN·m	kilonewton metre
kh	Not found			
kV	KVT	4937300	kV	kilovolt
ms	C26	4403766	ms	millisecond
pptt	Not found			

OPC UA communication alternatives

OPC UA is a large standard and there are many options on how to set up the communication. Some aspects of this matter are discussed here.

Read versus Subscribe

An OPC UA client is able to read any node exposed to it by an OPC UA server. This is similar to how modbus TCP works and is the simplest way to exchange data. For periodical updates this is though inefficient as it adds unnecessary request messages from the client to the servers. Therefore most OPC UA servers support subscriptions of monitored items. When the communication is set up using these services, the server sends periodic updates of the requested data to the client. Things like deadbands and buffering a few measurements before sending are also possible.

All OPC UA servers we came across during the PoC supported subscription type of communication. In turn, most PLC based OPC UA clients seemed not to support subscribing to values - only plain read. Only PLCs from B&R had support for subscriptions currently. Both OPC UA client implementations made by us during the PoC supported subscribing to variable updates.

It is possible to restrict many factors around subscriptions. It is for example possible to put restrictions on the amount of clients, the amount of subscriptions, the amount of monitored items per subscription and the fastest update rate. This way it is possible to control the amount of CPU load the OPC UA server puts on the system where it is running.

Secure versus plain communication

When it comes to securing the communication OPC UA comes with built in support and many different levels. On the server side, all tested SDKs had good support for secure communication - we also tested this in our server implementations. Of the tested SDKs, only Matrikon comes with client side support for secure communication. We did not have enough time and resources to test client side secure communication during the PoC. All tested SDKs rely on external libraries for the actual encryption, namely OpenSSL and mbedTLS.

One thing to note is that the servers and clients build using the tested SDKs are able to use different type of certificates needed for securing the communication. A larger question is though how to create, distribute and possibly revoke these certificates - especially when it comes to embedded systems. This was not in scope for the PoC.

Using certificates and building the information model in a suitable way it is possible to control which users or systems has what type of access to what data. A trusted Wärtsilä system may for example be give access to a larger set of variables to monitor. It is also possible to restrict writing so that only applications that are supposed to write to a certain input are able to do so.

Network layout

One of the tasks of the PoC was to study the feasibility of communicating both high-volume vertical data and latency-critical horizontal data through the same network and devices. It was found that the expected typical throughput of the vertical communication is still so low that it does not introduce risks of delaying horizontal traffic. Instead the ability of the endpoint devices to simultaneously handle vertical and horizontal communication is the limiting factor. Typically the latencies are not significant, but there is a big gap between typical and worst case latencies. See more in the performance testing section.