SUITABILITY ASSESSMENT OF OPC UA AS THE BACKBONE OF GROUND-BASED OBSERVATORY CONTROL SYSTEMS

W. Pessemier*, G. Raskin, H. Van Winckel, Institute of Astronomy, K.U.Leuven, Belgium
G. Deconinck, P. Saey, ESAT-ELECTA, K.U.Leuven, Belgium

Abstract

A common requirement of modern observatory control systems is to allow interaction between various heterogeneous subsystems in a transparent way. However, the integration of off-the-shelf (OTS) industrial products - such as Programmable Logic Controllers (PLCs) and Supervisory Control And Data Acquisition (SCADA) software - has long been hampered by the lack of an adequate interfacing method. With the advent of the Unified Architecture (UA) version of OPC (Object Linking and Embedding for Process Control), the limitations of the original industry-accepted interface are now lifted, and also much more functionality has been defined. In this paper the most important features of OPC UA are matched against the requirements of ground-based observatory control systems in general and in particular of the 1.2m Mercator Telescope. We investigate the opportunities of the "information modelling" idea behind OPC UA, which could allow an extensive standardization in the field of astronomical instrumentation, similar to the efforts emerging in several industry domains. Because OPC UA is designed for both horizontal and vertical integration of heterogeneous subsystems, we explore its capabilities to serve as the backbone of a dependable and scalable observatory control system, treating industrial components like PLCs no differently than custom software components. Performance measurements and tests with a sample of OTS OPC UA products are presented.

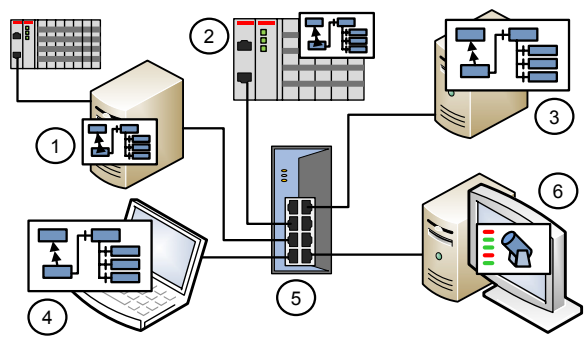
INTRODUCTION

Context and Motivation

Similar to the efforts in industry, control system engineers for astronomical observatories are continuously seeking to reduce development and maintenance costs and to increase system dependability. One of the most prominent trends is to foster software reusability, by creating frameworks which separate infrastructure code (i.e. the common *technical* aspects) from application logic (i.e. project-specific *functional* aspects) [1]. Another trend is the continued effort to integrate OTS (either commercial or public domain) software and hardware into these control systems [2]. Today, generic frameworks exist which meet both requirements to a large extent, even though some issues persist. Firstly, deeply involving frameworks may impose too strong constraints on the freedom of developers to choose the best fitting or best known technology for a given problem [3]. A "one size fits all" framework may exhibit code bloat and a lack of scalability, especially towards smaller and less demanding (embedded) applications. Secondly, as most generic frameworks to date rely on CORBA or DDS[1] as middleware to "glue" all subsystems together, integration of mainstream OTS industrial products remains inconvenient. As will be elaborated in this paper, OPC UA may offer a solution to these issues.

Methods

For this assessment we have compiled a list of qualitative and quantitative properties that are commonly required by observatory software frameworks, and in particular for the Belgian Mercator Telescope (a 1.2m optical telescope based at La Palma and currently being refurbished). The approach is twofold: not only do we want to verify if the OPC UA specification can satisfy these requirements, but we also want to test whether OTS implementations are complete and mature enough to build a functional infrastructure. For this purpose we have created a test set-up as depicted in Fig. 1, consisting of a small sample of commercial products with OPC UA connectivity. Naturally, many more products[2] of many more vendors are available on the market. Particularly important in the set-up is the SDK that we used to develop code for an experimental framework based on OPC UA. For this purpose we evaluated the C++ SDK by Unified Automation.



1. Kepware KEPServerEX UA server (www.kepware.com)
2. Beckhoff CX5020 PLC + UA server/client (www.beckhoff.com)
3. National Instruments LabVIEW UA server/client (www.ni.com)
4. Unified Automation C++ SDK server/client (www.unified-automation.com)
5. Moxa EDS-G308 Gigabit switch (www.moxa.com)
6. Siemens WinCC + Allmendinger UA client (www.siemens.com, www.allmendinger.de)

Figure 1: Test set-up.

* wim.pessemier@ster.kuleuven.be

[1] Object Management Group specifications: http://www.omg.org/spec.
[2] See OPC Foundation website: http://www.opcfoundation.org.

OPC UA BASICS

OPC UA is the successor of the suite of "classic" OPC standards that were developed by the OPC Foundation to interface industrial PLCs with SCADA and HMI (Human Machine Interface) systems. Besides the specification [4], a collection of code deliverables has been released to speed up application development and to facilitate compliance to the standard. Among these code deliverables are communication stacks in ANSI C/C++, JAVA, and C# .NET, which implement common lower level functionality. Two transport mechanisms are defined: the simple but high performance *UA TCP* and the Web Services standard *SOAP/HTTP*. Optional message signing and encryption is based on the *WS-SecureConversation* specification, and authentication is controlled by X.509 certificates. The stacks also implement data encoding via an efficient binary scheme or via XML. Built on top of the stack, an SDK is responsible for higher level functionality such as service handling and address space management. SDKs are offered by the OPC Foundation and by third party companies.

One of the most characteristic aspects of OPC UA is the extensive support for *information modelling*. Information entities (*Nodes*) and the relationships between them (*References*) can be defined, and exposed to the network in the *address space* of a server. Not only data itself but also meta-data can be exposed and transported over the wire. The semantics of the information model can be described formally in a *namespace*. A Node has several attributes and can represent an object, a variable, a method, an object/variable/data/reference type or a view (see [5]). A full mesh network of Nodes and References, representing a fictitious information model of METIS (the proposed Mid-Infrared E-ELT Imager and Spectrograph, see [6]) is shown in Fig. 2. The example is not meant to be optimal or complete, but attempts to illustrate some features of OPC UA. Object-oriented concepts such as data abstraction, encapsulation, polymorphism and inheritance are supported, as well as dynamic model changes, extensible references types, and cross-server referencing.

REQUIREMENTS

Table 2 lists a number of essential requirements [7], and whether they are met by the OPC UA specification and the tested SDK (second and third columns respectively). Only general comments are given, even though many requirements have more in-depth technical aspects which were verified by developing code snippets on top of the Unified Automation SDK. A recurring observation is the level of detail that is present in the OPC UA specifications and the derived stacks and SDKs. A substantial amount of these built-in features are distinctive for heterogeneous distributed control systems. For instance, data changes can be monitored asynchronously via the *MonitoredItems* services, which include standard arguments so that clients can request (and negotiate) the interval to sample the underlying data source, the publishing interval to minimize bandwidth overhead, an optional filter to deal with noisy data sources, and even a flag to request resampled data from the server. Dependability mechanisms are also strongly built into the OPC UA specification. For instance, OPC UA clients need to send "publish requests" to a server in order to keep the latter informed about the readiness of the client to receive events.

Quantitative requirements are more difficult to assess since they may vary strongly between projects and depend largely on the used hardware. We therefore use the set-up from Fig. 1 to measure the performance of a few typical service calls (*read* in this case) between a client and a server built with the Unified Automation SDK, see Table 1. The server is running on a Linux laptop and the client on a Windows PC. The UA Binary protocol is used to transmit the unencrypted and unsigned data via the gigabit network.

Table 1: Performance Measurements

Variables per call	Single UInt32 variable	1K*1K UInt32 matrix
1	0.37 ms	181 ms (∼22.1 MB/s)
10	0.68 ms	1784 ms (∼22.4 MB/s)
100	3.18 ms	
1000	23.26 ms	

The results indicate that the binary protocol of OPC UA is capable of high performance transmission of data (or meta-data) between components running on computer platforms. The performance in a more heterogeneous set-up (including embedded and real-time devices) is less useful to assess since it depends even more on the particular hardware and software configuration. Besides a priority mechanism for event handling on application level, no Quality of Service (QoS) features are specified by OPC UA. And, even though throughputs of around 22 MByte/s can be achieved when transmitting messages of several MByte (see Table 1), no special services are defined to exchange large data volumes. While other middleware technologies such as CORBA and DDS can meet strict real-time and data streaming requirements, OPC UA is clearly tailored for system architectures with complementary, special-purpose bus systems. These could include real-time fieldbuses, a bus for large data volume transport, an IEEE1588 timing bus, a safety bus, ... on a dedicated or shared medium (with managed switches to preserve QoS if needed).

FEASIBILITY

Besides accommodating special purpose frameworks (such as LabVIEW or PLC systems), an OPC UA-based infrastructure should provide a comfortable framework to develop applications in a popular language, on common platforms. Although SDKs provide useful high level functionality that is normally not part of a general middleware like CORBA, an extra software layer and tools are needed to simplify application development and deployment.

Table 2: Qualitative Analysis

Requirement	Spec.	SDK	Comments
Platform independence	Yes	Yes	No reliance on platform specific technology; SDKs are available for most platforms (non-real-time, real-time, and embedded).
Scalability	Yes	Yes	OPC UA is designed for vertical integration from sensor level up to mainframe level. Applications expose well defined *Profiles* to describe which services they support. In our test set-up, UA applications could be developed for embedded devices with few supported profiles (such as PLCs) to rich-featured servers on Linux PCs.
Reusability	Yes	Yes	OPC UA facilitates and encourages the definition and standardization of information models that extend the standard namespace. This is exemplified in Fig. 2: standardization can be achieved on a low level (e.g. by representing all sensor values according to the standard *AnalogItemType*), on a high level (e.g. by deriving a *DetectorControllerType*) or even on a system-wide level (by referencing Nodes in other servers in order to facilitate the organization of complex systems and avoid unnecessary aggregation).
Communication paradigms	Yes	Yes	Services are defined to read and write data, to invoke methods on remote objects, and to subscribe to events and data changes (monitored items). Multiple operations (e.g. method calls) can be invoked in one request. The tested SDK provides both a synchronous and asynchronous API for the service calls. Processes requiring a longer execution time should be modelled as *Programs* (standardized state machines) instead of methods.
Complex data	Yes	No	Servers need to expose the structured variable instances, their type definition and their encoding details (so that clients can discover how to interpret them). The SDK we tested required manual encoding of these user-defined types, but both stub/skeleton code generators (for types known at compile time) and generic helper classes (for types known at runtime) were due to be released with the next minor update.
Lifecycle management	No	No	Due to scalability and platform independence, OPC UA systems may be very heterogeneous. Therefore transparent starting, stopping and killing of client and server processes is complicated or (in case of embedded devices) may be impossible. Lifecycle management of objects similar to the container-component model can be accommodated well by OPC UA, but the central "manager" (or a hierarchy of managers) needs to be developed on top of the SDK. When a manager references Nodes in other servers, clients can easily resolve these Nodes by using standard OPC UA services.
Alarms	Yes	Yes	*Alarms* and *Conditions* (representing the state of a system) are standardized in OPC UA by extending the event mechanism. Advanced features such as "shelving" (to prevent operators from being flooded by alarms) are built-in.
Logging	Yes	Yes	Straight-forward to implement by extending the *BaseEventType* (containing fields for the timestamp, priority, description, ...) or one of its subtypes (e.g. for auditing purposes).
Location transparency	Yes	Yes	OPC UA foresees *Discovery Servers* at well known locations, to which other servers can register. Clients can query them to find the location of the server they need. More advanced discovery of applications is feasible via a *Global Directory Service*.
Historical archive	Yes	Yes	OPC UA does not define how historical information is stored, but does specify how it can be accessed. An attribute is assigned to variables and event notifiers to specify whether historical data or events are available. Clients can invoke standard services to read or modify raw (or even resampled) historical data at given timestamps and intervals.
Dependability	Yes	Yes	A wide range of mechanisms to increase reliability and availability are offered on a low level (e.g. sessions handlers can deal with network interruptions) and on a high level (e.g. compliance tools are available to test servers and clients). Security measures (confidentiality, integrity, authentication and authorization) are part of the stack and are therefore implemented by most OPC UA products. Due to the extensible design of OPC UA (e.g. new encodings and transport protocols can be added) and the wide support from industry, OPC UA is likely to be future proof. However, due to the lack of a standardized API, a system infrastructure may strongly depend on a particular stack or SDK.

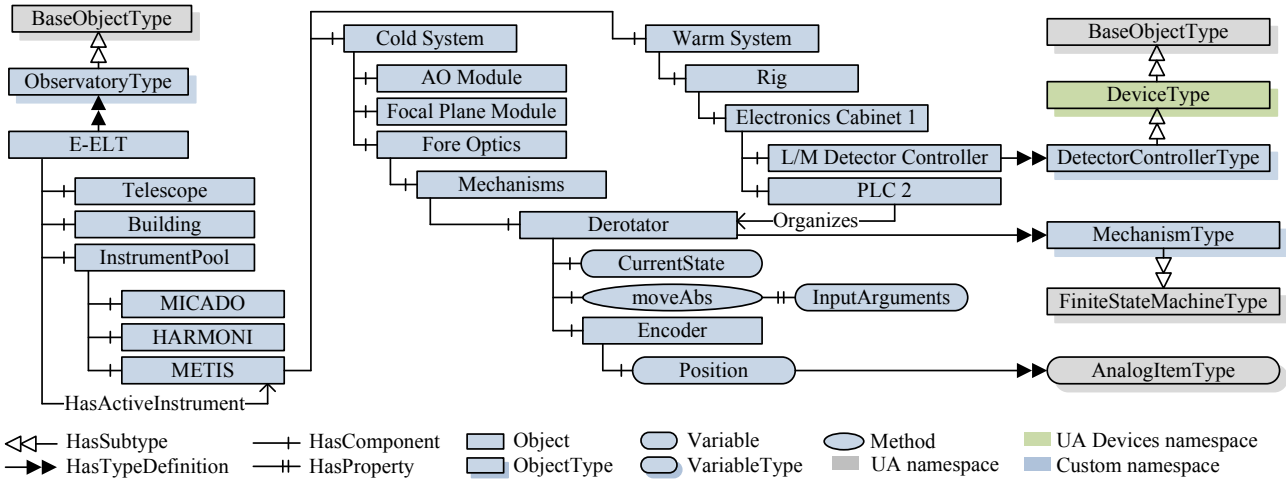


Figure 2: Example of a full mesh information model in the context of METIS.

An important difference with CORBA is the poor availability of stub and skeleton code generators. Similar to CORBA Interface Definition Language (IDL) compilers, OPC UA code generators can be used to convert XML namespace definitions into stub and skeleton code. Unfortunately, only a few OTS code generators existed at the time of writing this article. Those we did find were either inaccessible or too inflexible for us. Therefore we decided to estimate the expense of a homemade solution. For this purpose we wrote a few Python scripts which were able to convert XML namespace definitions (designed in UML [1] by one of the available graphical tools) into C++ server skeletons with rudimentary introspection and dispatching functionality. Application developers can derive classes from these skeletons and include business logic. When the corresponding objects are instantiated and registered to our server implementation, both the object instance and type definition are automatically exposed. On the client side we opted for a design with a narrow interface, in order to accommodate multiple actions in one service call. While it's clear that our experimental code is unsuitable to be used outside our test environment, we expect that it will take less than a man-year of work to prepare and commission a basic framework for the Mercator Telescope.

CONCLUSIONS AND OUTLOOK

In conclusion, we consider OPC UA suitable to serve as the "backbone" technology of ground-based observatory control systems. We estimate that the specification and available implementations are sufficiently mature and complete to cover the requirements of many projects. Even so, at this moment an infrastructure built around OPC UA cannot take full advantage from all opportunities offered by the specification, since the adoption by industrial products is still limited. For instance, UA Binary connectivity is generally supported, but SOAP/HTTP is not.

With respect to a "general purpose" middleware such as CORBA, OPC UA is naturally more tailored to facilitate interaction between the components of a heterogeneous and distributed control system. In effect, much of the basic functionality (alarms, logging, monitored items, ...) required by infrastructures for astronomical observatories is already standardized by the OPC UA specification. As a result, development efforts are reduced (because this functionality is offered by SDKs), and more importantly, industrial software and hardware can be integrated much more seamlessly since they comply to the same standard.

More effort and a more rigorous approach are certainly needed to create a framework which streamlines and facilitates application development for PCs, but we consider the investment worthwhile even within the constraints of a project on the scale of the 1.2m Mercator Telescope. For the refurbishment of this control system we aim to host most software on industrial PLCs and Linux PCs. We will thereby try to benefit as much as possible from technology advancements such as powerful and dependable soft-PLCs, flexible (object-oriented) PLC programming environments, and OPC UA as the universal control interface.

REFERENCES

[1] G. Chiozzi et al., "Enabling technologies and constraints for software sharing in large astronomy projects", Proc. SPIE, Vol. 7019, 2008.

[2] G. Chiozzi et al., "Trends in software for large astronomy projects", Proc. ICALEPCS, 2007.

[3] P. J. Young et al., "Instrument control software requirement specifications for Extremely Large Telescopes", Proc. SPIE, Vol. 7740, 2010.

[4] OPC Foundation, OPC UA specification Parts 1-13.

[5] W. Mahnke, Stefan-Helmut Leitner and Matthias Damm, "OPC Unified Architecture", Springer-Verlag, 2009.

[6] B. R. Brandl et al., METIS: The Mid-Infrared E-ELT IMager and Spectrograph, Proc. SPIE, Vol. 7014, 2008.

[7] G. Raffi, B. Glendenning, "ALMA Common Software Technical Requirements", ESO, Issue 1.0, 2000-06-06.