Service Orchestration with OPC UA in a Graphical Control Language

Alfred Theorin
Department of Automatic Control
Lund University, Box 118
SE-221 00 Lund, Sweden
Email: alfred.theorin@control.lth.se

Johan Hagsund
Department of Automatic Control
Lund University, Box 118
SE-221 00 Lund, Sweden
Email: johan.hagsund@gmail.com

Charlotta Johnsson
Department of Automatic Control
Lund University, Box 118
SE-221 00 Lund, Sweden
Email: charlotta.johnsson@control.lth.se

Abstract—Production plants need to be set up and reconfigured faster to fulfill increasing market demands. Highly flexible automation systems are needed and a promising approach is Service Oriented Architecture (SOA) which has recently received much attention in both academia and industry. OPC Unified Architecture (OPC UA), the next generation of the de facto standard for interoperability in the automation domain, has SOA capabilities.

In this paper it is presented how SOA service orchestration of OPC UA services can be done conveniently with a graphical control language. Generic support to use OPC UA servers has been integrated into the language and as an example it has been used to control a physical process which has been modeled, encapsulated, and exposed as an OPC UA server by wrapping it with an ethernet capable microcontroller.

I. INTRODUCTION

Manufacturing companies have to set up and reconfigure their production plants in shorter time frames to fulfill increasing market demands. At the same time, both manufacturing equipment (the field devices) and production processes become more sophisticated. Production processes are typically implemented with control programs in a Distributed Control System (DCS) or in Programmable Logic Controllers (PLCs). More sophisticated field devices and production processes also means that the control programs become more complicated, leading to higher efforts for development, commissioning, maintenance, and reconfiguration. Shorter set up times can be achieved by basing the control programs on the process diagrams from the planning phase [1] as well as on composable components. Field devices and control programs should be modular and easy to integrate, configure, extend, and reuse which calls for highly flexible automation systems and is essential for efficient reconfiguration.

A promising approach to compose flexible control programs is Service Oriented Architecture (SOA) which is widely used for business processes [2]. It has been recognized for use in automation in several research projects [3], [4]. The SIRENA project [5] resulted in an open source Devices Profile for Web Services (DPWS) implementation for embedded devices, thus pushing the web services technology used at business level down to device level. DPWS has been shown to be useful for industrial automation [6], [7] but the OPC Unified Architecture (OPC UA) [8], which is feature backward compatible with OPC – the current de facto standard for interoperability in the automation domain, is likely to spread faster [9].

Field device communication traditionally uses dedicated deterministic, highly reliable, and low bandwidth field buses. The traditional physical architecture of control systems is rigid, with each field device connected to one field bus which in turn is connected to one controller, for example a PLC. Industrial ethernet protocols are becoming widely accepted and used. When the ethernet infrastructure also includes the field devices, and since the bandwidth of ethernet is 2–5 orders of magnitude higher than for traditional field buses, there is now an opportunity to use ethernet not only for lower priority tasks such as field device configuration and diagnostics but also for control. As the communication with field devices no longer has to go through a specific controller, the architectural flexibility is increased by decoupling the physical and logical hierarchy of field devices. Hence loose physical coupling is attained, which simplifies horizontal and vertical integration.

In this paper it is presented how generic support to connect to and interact with OPC UA servers has been integrated into a graphical control language. As an example it has then been used to control a physical process which has been modeled, encapsulated, and exposed as an OPC UA server by wrapping it with an ethernet capable microcontroller. To the best of our knowledge, this is one of the first times that SOA service orchestration is performed with a technology that is anticipated to be widely available in the automation domain within a few years.

The aim of this paper is to present how SOA service orchestration with OPC UA can be conveniently used for control. In Section II the importance of SOA is discussed. In Section III OPC UA is described and in Section IV the graphical control language Grafchart and the Grafchart development environment JGrafchart are introduced. In Section V the new OPC UA integration in JGrafchart is described. The modeling and control of the physical process is described in Section VI and finally conclusions and future work are presented.

II. SERVICE ORIENTED ARCHITECTURE

As new functionality and systems are added they need to be integrated with existing systems. The traditional integration approach in manufacturing is to connect applications on a Point-to-Point (PtP) basis using the client/server pattern. The
pattern mandates that the client knows about the server and that the server knows about the client. The number of connections in a fully connected network increases quadratically with the number of applications. This is known as “spaghetti integration” and makes the system rigid and hard to maintain [10]. Each time an application is added, all other applications need to be updated to be able to communicate with it.

It is also common that applications are only able to communicate through proprietary or limited protocols and applications may require external message translators to communicate with each other at all. This is for example the normal case for communication between PLCs from different vendors. Another challenge is the communication between the different hierarchical levels, known as vertical integration.

The PtP approach is almost useless in supporting the expected business requirements [11]. Yet industry has been slow to migrate to new approaches, mainly due to the cost of replacing their established legacy systems based on PtP [10]. However, migration has been significantly accelerated by the advent of SOA [10].

SOA is a distributed software architecture where self-contained applications expose themselves as services, which other applications can connect to and use. To reach its full potential, SOA applications should be discoverable and self-describing as well as platform- and language-independent. This leads to loose coupling and a high degree of flexibility. SOA has recently received much attention in both academia and industry. It is widely used on the business level and has been expected to revolutionize manufacturing in a similar fashion.

The further down the physical hierarchy, the shorter the time frame. On the lowest level it is common to have real-time requirements with time frames in the order of milliseconds. The devices which execute on this level often have strictly limited memory and computational power. There is a trade-off between flexibility and real-time performance [12] and thus, the further down SOA is wanted, the more performant (and hence less flexible) it needs to be. There have been several initiatives to bring SOA to shop floor by customizing and bringing the web service technology used for business processes down to the shop floor’s resource constrained devices [13], [14], [5]. As most SOA tools are tailored for business processes, which do not have this kind of strict requirements, they are not suitable for manufacturing processes with real-time requirements. OPC UA on the other hand has SOA capabilities and has been developed specifically for the automation industry.

III. OPC UA

OPC UA is a recent standard [8] which incorporates all the features from the three classic OPC specifications OPC Data Access (OPC DA), OPC Alarm and Events (OPC AE), and OPC Historical Data Access (OPC HDA) which are all based on the Microsoft proprietary COM/DCOM technology. OPC UA on the other hand is platform and language independent and also incorporates SOA features [15]. OPC UA was developed with the goal to offer robust communication in distributed systems with high security, connecting everything from small embedded devices to large enterprise systems.

A. Information Model

OPC UA offers more powerful modeling possibilities than classic OPC in which only simple data could be represented [15]. Types and instances in the specific domain are defined in the so called address space. OPC UA data modeling uses the concepts of nodes and references. Types and instances are represented by nodes and the relationship between two nodes are represented by references. Each node has a number of node attributes, some mandatory and some optional. The data type for all attributes except the value are defined by the specification. The OPC UA specification defines a fixed number of built-in data types such as the common Double, Int32, Boolean, and String as well as the OPC UA specificNodeId and QualifiedName. Each node has a NodeId with a unique id number which is used for example to find a specific node in the address space.

Eight node classes are defined in the OPC UA specification and the three most important ones are variables, methods, and objects [15]. Variables represent values on the server. Methods are functions/operations which can be called and they may have multiple input and output parameters. Objects consist of a set of variables, methods, and objects and they are typically used to represent real world objects. Each node also has a set of references to other nodes in the server’s address space.

B. Client-Server Interaction

All information modeling is done on the server side and made securely available for clients to access; connections are set up with the desired security level [15]. The client can access a variable value in two ways: by subscribing to it (recommended) or by explicitly asking for it. The client can also specify how often values should be received in a subscription.

OPC UA has support for two transport protocols namely UA Binary and UA XML [16]. The binary TCP protocol is fastest and offers best performance and interoperability whereas SOAP is better supported by various tools [17]. Figure 1 shows an overview of the OPC UA stack. At the bottom is the transport protocol layer. Which transport protocol to use is determined by the URI scheme name (http, https, or opc.tcp) Above this is the security protocol layer which takes care of message signing and encryption. Next is the data encoding layer which determines the structure, encoding, and serialization of messages. In the application layer at the top are the user applications, implemented in the desired programming language, typically using an OPC UA toolkit for this language.

![Fig. 1. An overview of the OPC UA stack.](image-url)
C. Backward Compatibility

One goal with the new OPC UA standard was to make it platform independent and thus it was necessary to remove dependencies to proprietary technologies such as DCOM which OPC is based on. Thus it is not possible to directly connect an OPC UA client to an OPC server or an OPC client to an OPC UA server. However, OPC UA is backward compatible feature-wise, which means that all features supported by OPC are also supported by OPC UA. It is thus possible to convert calls between OPC UA and OPC [18]. In this way it is possible to connect OPC UA clients to OPC servers and OPC clients to OPC UA servers. The work presented in this paper can thus be used to connect to most commercial PLC and DCS systems.

IV. THE GRAFCHART LANGUAGE

Graphical programming languages are popular in the automation domain. Some advantages are simplicity and better application overview than for textual languages. Visualization of a graphical application is also straightforward and more intuitive than textual languages. Since there is a considerable cost to restart or have a producing machine stopped, good visualization is needed for advanced troubleshooting and online tweaking without stopping the machine.

Grafchart is a graphical programming language which extends Sequential Function Charts (SFC), one of the IEC 61131-3 [19] PLC standard languages for sequential, parallel, and general state transition oriented applications. SFC is supported by most large industrial automation systems and is widely used and accepted for industrial automation. It is a low level programming language and thus implementing sophisticated applications in SFC is inconvenient.

Grafchart has the same graphical syntax as SFC with steps and transitions, where steps represent the possible application states and transitions represent the change of application state. Associated with the steps are actions which specify what to do. Associated with each transition is a Boolean guard condition. Grafchart has, like SFC, several action types and alternative and parallel paths. Grafchart also includes the common SFC extension for hierarchical structuring (Macro Steps). Additionally Grafchart adds high level features such as reusable sub-function-charts (Procedures) and constructs for exception handling (Exception Transitions and Step Fusion Sets) which make it more convenient to implement and maintain large applications. Grafchart is described in more detail in [20].

A. JGrafchart

JGrafchart is a free Java based integrated development environment for the Grafchart programming language and can be downloaded from http://www.control.lth.se/Research/tools/grafchart.html. It is a research tool that has been used in several research projects. It supports numerous graphical elements to make it possible to create rich and interactive operator interfaces. It also supports various customizable IO to make it possible to connect JGrafchart applications to various external environments. Within the LISA research project, generic support for DPWS was added to JGrafchart [7].

V. OPC UA INTEGRATION IN JGrafchart

OPC DA is used the most in the industry today. Among products using OPC, 99% implement OPC DA while OPC AE and OPC HDA are mostly implemented in addition to OPC DA [15]. Thus, this work is mainly focused on support for OPC UA data access.

JGrafchart has been extended with integrated support to connect to and interact with OPC UA servers over the UA Binary transport protocol. All OPC UA data types needed for controlling purposes are supported. A new I/O element called OPC UA Object has been added to JGrafchart to make it possible to connect JGrafchart applications to OPC UA servers. Each OPC UA Object is bound to a variable or object node in the OPC UA server’s address space. Multiple OPC UA Objects can be added to a JGrafchart application to set up connections to multiple nodes on one or more OPC UA servers. The binding for each OPC UA Object is configured by specifying the Server URI and the full BrowsePath to the desired node.

A dialog to browse the address space of connected servers has been added to JGrafchart and is illustrated in Figure 2. The right side of the dialog shows the currently selected node’s metadata, such as its BrowsePath, data types of variables, and input and output parameters for methods, as well as the node’s device’s metadata such as Server URI and Application Name. It is also possible to connect and disconnect OPC UA servers. At the bottom of the dialog is another, more convenient, way to configure OPC UA Object bindings than specifying them manually.

![Fig. 2. The OPC UA dialog in JGrafchart.](image)

When an application with configured bindings is opened, JGrafchart tries to connect to the bound OPC UA servers automatically. OPC UA also supports monitoring the ServerState...
which for example means that it is possible to detect when a server goes down or comes back up. This has also been implemented and there is an automatic reconnect when a previously connected server comes back online.

The integrated OPC UA support has been designed so that OPC UA variable accesses and OPC UA method calls look like ordinary JGrafchart variable accesses and method calls. Variables and methods at any level in the bound node’s subtree are accessed with dot notation. It is always possible to write to variables and call methods. To receive variable value updates requires subscriptions which are managed with the new JGrafchart functions `opcSubscribe` and `opcUnsubscribe`. Figure 3 illustrates how to use OPC UA variables and methods. On the first line a subscription for the variable `MyLevel`’s value is created. On the second line the Boolean variable `MySwitch`’s value is set to true (1). On the third line the method `MyMethod` is called with two input arguments and the call’s output argument is stored in the JGrafchart variable `ret` (not shown). Finally, the variable `MyLevel`’s value is accessed in the transition condition. Method calls are currently limited to one output argument as adding support for this requires modification of the syntax of the JGrafchart actions language. Also, the current implementation is indifferent to whether a method call is synchronous or asynchronous.

```plaintext
S opcSubscribe(myDevice.MyLevel.Value);
S myDevice.MySwitch.Value = 1;
S ret = myDevice.MyMethod("sin", 45);
```

To choose which node in the address space to bind to is a trade off between the number of required OPC UA Objects and the number of levels to type for each variable and method access. The extremes are binding to the root node and binding to a variable node. Binding to the root node means that everything on the server is accessible through the same OPC UA Object but that each access requires typing the maximum number of levels. Binding to a variable node means that only that variable can be accessed through the OPC UA Object but each access only requires typing the minimum number of levels.

**VI. Example**

A device from fischertechnik has been used to evaluate the new integrated support for OPC UA in JGrafchart. The device consists of a punching machine which can punch goods, and a conveyor belt which can transport goods to and from the punching machine, see Figure 4. The device has four digital inputs and two DC motors. Two light barriers detect the position of the goods and two limit switches detect the vertical position of the punching machine. One motor drives the punching machine and the other motor drives the conveyor belt. The device is controlled by electrical signals (voltages) and to expose it using OPC UA, it is connected to an ethernet capable microcontroller, namely an Aria G25.

![Fig. 4. The fischertechnik device and the microcontroller circuit board.](image)

The task for the device is to process arriving goods. When goods arrive the conveyor should move it to the punching machine, punch it, and then move it back to the entry again.

**A. Server Side Device Modeling**

The device is wrapped by its own OPC UA server running on the microcontroller, making the functionality of the device available for OPC UA clients to access and control. The conveyor belt is modeled as an object containing two variables, `Forward` and `Backward`, which determine how to move the conveyor belt and two variables, `AtEntry` and `AtPunch`, which sense when the goods is at the entry or by the punching machine. Similarly, the punching machine is modeled as an object containing the variables `MoveUp`, `MoveDown`, `IsUp`, and `IsDown`, Figure 5.

![Fig. 5. The OPC UA server’s address space.](image)
To prevent networking issues or a poorly written client application from damaging the device, the server should contain interlocks to avoid situations which might harm the device, for example prevent the punching machine from moving further up when $IsUp$ is set.

The OPC UA server was implemented with the C++ OPC UA toolkit from Softing [21] cross compiled to the Aria G25 which runs Debian on an ARM9 processor. Implementing the server was straightforward.

The main reason to implement everything as variables is that they are considerably easier to implement on the server side than methods. An alternative would be to have a Move method for each motor which accepts an argument with the desired direction, for example an integer where positive numbers mean forward, 0 means stop, and negative numbers mean backward. It would also be possible to have both the variables and the methods so that the client can choose which to use.

B. Device Control

The device is controlled from a JGrafchart application which connects to the OPC UA server, see Figure 6. In the application cb (conveyor belt) and pm (punching machine) are OPC UA Objects bound to the corresponding object nodes on the OPC UA server running on the microcontroller connected to the fischertechnik device. In addition to what is shown in the figure there is also a parallel step which sets up subscriptions to variables and forward their values to the internal values used in the transitions.

In the initial step the application waits until goods arrives at the entry of the conveyor belt, transition GoodsAtEntry, and it then starts to move the goods toward the punching machine. When the goods arrives at the punching machine, transition GoodsAtPunch, the conveyor belt is stopped and the punching machine starts to move down. When the punching machine is in the bottom position, transition IsPunchDown, it starts to move up again until it reaches the top position, transition IsPunchUp. Then the conveyor belt starts to move the goods back to the entry position and when it arrives, transition GoodsAtEntry, the conveyor belt is stopped and the application waits for the current goods to be removed, transition !GoodsAtEntry, before going back to the initial state to wait for the next goods to arrive.

VII. CONCLUSIONS

This paper has presented a convenient way to implement SOA service orchestration with OPC UA. The support for OPC UA in JGrafchart is well integrated. OPC UA variable accesses work like ordinary JGrafchart variable accesses, OPC UA method calls like ordinary JGrafchart method calls, and OPC UA objects are traversed with dot notation like for ordinary hierarchical structures in JGrafchart. Automatic connect and reconnect to servers is also supported during both development and execution. The new OPC UA support was successfully used for control of an embedded device which had been made self-contained by wrapping it with an ethernet capable microcontroller to expose it as an OPC UA server.

As concluded in [6], both OPC UA and DPWS are required to cope with all industrial requirements on device level. As

Fig. 6. Control application for the device’s task implemented in JGrafchart using the new OPC UA capability.

JGrafchart now supports both integrated DPWS and OPC UA it can be used regardless of which is more suitable in a specific scenario.

VIII. FUTURE WORK

Data access is by far the most commonly used part of the OPC UA standard. The focus has thus so far been on integrating data access in JGrafchart. Alarm and events and historical data access are part of future work.

To evaluate the applicability of the work for legacy systems it would also be interesting to connect JGrafchart to an industrial automation system via an OPC converter.

ACKNOWLEDGMENTS

The authors are members of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University. The authors would like to thank VINNOVA for financial support through the LISA project, DFKI in Kaiserslautern Germany for all support, and Softing for the OPC UA toolkit.
REFERENCES


