Real-Time Data Acquisition Support for IEC 61499 based Industrial Cyber-Physical Systems

Wanqi Huang, Shanghai Jiao Tong University, China, hwqisme92816@sjtu.edu.cn,
Wenbin Dai, *IEEE Senior Memb*er, Shanghai Jiao Tong University, China, w.dai@ieee.org
Peng Wang, Shenyang Institute of Automation, Chinese Academy of Science, wangpeng@sia.cn
Valeriy Vyatkin, Luleå University of Technology, Sweden and Aalto University, Finland, *IEEE Senior Member*, vyatkin@ieee.org

***Abstract* – Industrial Cyber-Physical Systems enhance physical devices with control, computing and communication abilities to provide smart solutions for industrial applications. In industrial applications, data acquisition is an important topic. Based on feedback data from sensors and equipment, industrial processes could be optimized either by experienced engineers manually or smart agents automatically. The IEC 61499 standard is designed for modeling distributed automation systems that perfectly fits the scope of industrial CPS. However, real-time data acquisition is not supported by the IEC 61499 standard currently. In this paper, an extension to IEC 61499 compliance profile is proposed to support real-time data acquisition. Both client/server and peer-to-peer model are supported. The proposed extension is demonstrated using a demo car manufacturing line.**

Index Terms *— Industrial Cyber-Physical Systems; IEC 61499 Function Blocks; Real-Time Data Acquisition; Management Commands; XML; Publish-Subscribe; Client-Server.*

I. Introduction

The cyber-physical system (CPS) is considered as the enabling technology for achieving industry 4.0 [1]. CPS provides rapid cooperation and optimization based on feedback received from various interconnected devices. In industrial automation, the industrial cyber-physical system (iCPS) bridges various isolated subsystems by creating peer-to-peer information channels from the enterprise level all the way down to the field level [2].

In the iCPS, one challenge is to achieve data transparency through system hierarchies. Legacy automation system architecture is guided by the ISA-95 standard [3]. The ISA-95 standard defines a 5-layered system hierarchy for industrial automation systems. Data exchange is limited to two direct connected layers in the ISA-95 reference architecture. To read a sensor value from upper layers, request and response messages must be transmitted via middle layers which cause long response time and increase network traffic load. Interoperability in iCPS cannot be achieved by using existing ISA-95 architecture.

Service-oriented architecture (SOA) is considered as an effective way to improve flexibility and enable interoperability for iCPS [4]. The SOA-enabled iCPS provides direct connections between all devices and subsystems. On the controller layer, the IEC 61499 standard offers component-based design and modeling methodology for distributed automation systems [5]. However, a complete solution for real-time data monitoring of interoperable iCPS is yet covered by the IEC 61499 standard and will be discussed in this paper.

The rest of the paper is organized as follows: In Section II, related works regarding service-oriented architecture (SOA) in industrial automation and existing approaches for data acquisition in the IEC 61499 are reviewed. In Section III, an SOA-based data acquisition architecture is proposed for the IEC 61499. In Section IV, extensions to management commands defined in the IEC 61499 compliance profile are proposed. In Section V, a case study of car manufacturing demonstration line is used to illustrate the proposed method. Finally, this paper is concluded and recommendations for future works are discussed.

II. Related Works

Service-oriented architecture is commonly adopted for flexible and interoperable iCPS. Karnouskos et. al. proposed a service-based architecture for cloud-based iCPS in the IMC-AESOP project [7]. Multi-system interactions and cross-layer collaborations are experienced by integrating cloud computing, Internet-of-Things and web services. The proposed architecture can handle requests for monitoring, management, data handling and integration for future industrial automation systems.

Jammes et. al. [6] explored all technologies deployed in SOA-based control and monitoring systems for the large-scale process industry. Critical research topics such as real-time SOA, management of large scale industrial distributed control systems, synchronization of distributed event-based systems and semantics defined in service specifications are discussed. Technologies including DPWS, OPC UA, CoAP, EXI, Service Bus and CEP are analysed for each research topic.

The OPC UA is also linked with the SOA for providing data channels between industrial applications [8]. A generic synchronized solution for data acquisition in distributed automation systems is proposed by Pethig et al. [9]. The Ethernet-based solution for data acquisition uses IEEE 1588 Precision Time Protocol (PTP) for time synchronization. On the data layer, the OPC Unified Architecture is used for collecting data. From the experiment, the real-time requirement for data acquisition is achieved by the proposed synchronized solution.

The OPC UA is widely adopted for IEC 61131-3 based PLCs [19]. Durkop et. al. [10] proposed a real-time SOA solution for industrial automation systems. The Web Service is used over the none real-time IP network for integrating enterprise systems, engineering systems, gateways, and devices. Real-time Ethernet is used between PLCs and IO devices for reading and writing data. Reconfiguration between PLC and I/O devices is achieved by ad-hoc channels. The OPC

UA model is used for mapping services between modules. Miyazawa et. al. [11] proposed a secure software model for data exchange between IEC 61131-3 PLC and OPC UA server. The security of PLC programming is ensured by using OPC UA.

The OPC UA has also experimented with IEC 61499 based applications by many researchers. Melik-Merkumians et al. [12] proposed a SOA-based middleware for industrial control software by using OPC UA. The IEC 61499 application model has converted into an OPC UA address space model automatically. It is proved that the OPC UA is suitable to be used as a middleware for industrial control systems.

Andren et al. [13] proposed a reconfigurable communication gateway for distributed industrial automation systems. The gateway is designed based on the IEC 61499 service interface function blocks. The high-level communication pattern is introduced for hardware-independent access. The proposed gateway supports multiple communication models such as client/server and publish/subscribe and able to reconfigure communication links dynamically.

This work starts from the extension to IEC 61499 proposed by Wenger et al [14]. Behavioural types are extended to existing IEC 61499 components for runtime monitoring of behavioural specifications. In this paper, more real-time data monitoring features will be proposed as the extension for IEC 61499-based iCPS.

III. Data Acquisition for IEC 61499 based iCPS

Data acquisition in legacy industrial automation systems is fulfilled by using OPC standard architecture. The OPC using Client/Server communication model to poll data from programmable logic controllers (PLC) to supervisory control and data acquisition (SCADA) systems. In iCPS point of view, data shall be retrieved from SCADA, MES or even ERP systems directly. With existing data acquisition architecture, it is impossible to achieve the goal.

In SOA-based IEC 61499 compliant iCPS, data acquisition is offered in two communication models: the Client/Server model [15] and the Publish/Subscribe model [16]. The client/server model refers to a N-to-1 architecture where single or multiple clients are communicating with a server for message exchange. A client/server model is a bi-directional communication that both client and server side can send messages. The OPC architecture is using the client/server model for polling data from controllers to SCADA systems. The OPC server side will cycle through all defined variables and update their values from PLCs where OPC client sides are deployed.

The Publish/Subscribe model is another message pattern that provides better scalability and loose coupling. The Publish/Subscribe model refers to an N-to-N architecture where a publisher can push messages to multiple subscribers and a subscriber can also receive messages from various publishers. In the Publish/Subscribe model, PLCs are responsible for pushing data to a high-level SCADA system.

In service-oriented iCPS, horizontal and vertical integration between all layers require a heterogeneous data acquisition mechanism that any node can send or receive messages from another node. In addition, information from a particular node may be required by other nodes that located in different layers of the ISA-95 hierarchy. To adopt flexible requirements in iCPS, both the client/server model and the publish/subscribe model must be considered in the IEC 61499 compliant controllers.

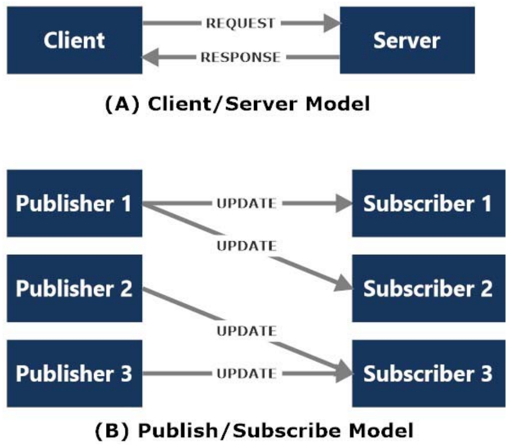

Fig. 1: Client/Server Model vs Pub/Sub Model

IEC 61499 devices are managed by management commands defined in the compliance profile of the IEC 61499 standard [17]. Existing management commands provides modifying program functions such as create and delete function block types, instances, event and data connections; management features including start, stop, kill and reset a function block; data functions are also supported for example, read and write parameters of function block instances or query for list of FB instances. However, data acquisition and monitoring are not currently supported in management commands as defined in the IEC 61499 compliance profile.

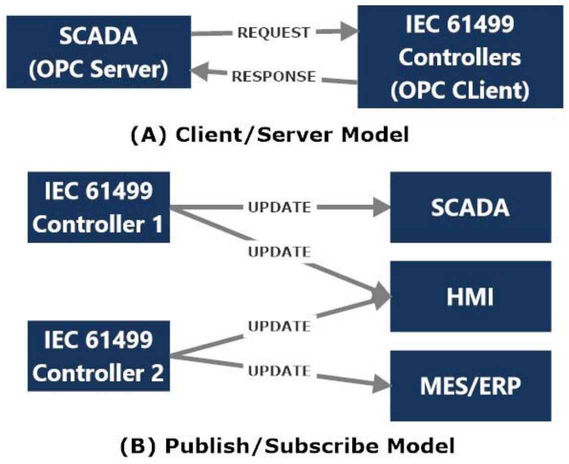

Fig. 2: Client/Server Model and Pub/Sub Model in IEC 61499

IV. IEC 61499 Management Commands Extensions For Data Acquisition

To support both client/server and publish/subscribe model for data acquisition, extensions to existing management commands are proposed in this section.

First, three elements *Event*, *Var* and *Watch* are added to management command definitions for monitoring purposes. The *Event* is defined as:

```
<!ELEMENT Event (Var*)>
<!ATTLIST Event
  Name CDATA #REQUIRED
  Type (Input | Output) #IMPLIED
  With CDATA #IMPLIED
  Comment CDATA #IMPLIED
  Lastts CDATA #IMPLIED >
```

Where *Type* indicates event input or output; *With* refers to associated data variables with this event; *Lastts* indicates last triggered time.

The element "Var" is defined as:

```
<!ELEMENT Var EMPTY>
<!ATTLIST Var
  Name CDATA #REQUIRED
  Type (Input | Output | Internal) #IMPLIED
  Value CDATA #IMPLIED
  Forced (True | False) #IMPLIED >
```

Where *Type* refers to input, output or internal variables; *Forced* indicates whether this variable is overridden and forced to a constant value.

The element "Watch" is defined as:

```
<!ELEMENT Watch (Var | Event)>
<!ATTLIST Watch
Type CDATA #IMPLIED
Source CDATA #IMPLIED >
```

Where *Type* refers to the monitoring model (CS - client/server or PS - publish/subscribe); *Source* indicates the address of the server or the subscribing node.

A. Add Watch to an element of a FB instance

To add watch to an event of a FB instance, the following request message is sent:

```
<Request ID="{0}" Action="CREATE">
   <Watch Type="{1}" Source="{2}">
    <Event Name="{3}" Type="{4}" With="{5}" />
</Watch>
</Request>
```

Where {0} refers to positive integer number that links between a request and response message.

To register monitoring associated data variables with this event, the attribute field *With* can be used. When the *With* attribute is set to "*", all data variables associated with this event are registered; to register one or more data variables associated with this event, the *With* attribute can be set to "*Var1*, *Var2*…".

Similarly, to monitor a variable of a FB instance, the following management command can be used:

```
<Request ID="{0}" Action="CREATE">
   <Watch Type="{1}" Source="{2}">
    <Var Name="{3}" Type="{4}" />
   </Watch>
</Request>
```

The variable name is formed as *<DeviceName>.<ResourceName>.<ApplicationName>.<FBInstanceName>.<VariableName>.*

B. Remove Watch from an element of a FB instance

To remove an event or a variable from the watch list, the *Action* field of the request message shall be set to "*DELETE*" and the only name of this event or variable is required. All data variables associated this event will also be removed from watch list simultaneously.

```
<Request ID="{0}" Action="DELETE">
   <Watch>
    <Event/Var Name="{1}" />
   </Watch>
</Request>
```

C. Start/Stop Monitoring

After events and variables are registered, monitoring process could be started or stopped at any time using the following command:

```
<Request ID="{0}" Action="START/STOP">
   <Watch Type="{1}" Source="{2}" />
</Request>
```

D. Query All Watching Elements

To list all currently being watched elements, the following command can be used:

```
<Request ID="{0}" Action="QUERY">
   <Watch Type="{1}" Source="{2}" />
</Request>
```

The response message is constructed as:

```
<Response ID="{0}" Reason="{1}">
   <Event Name="{2}" Type={3}" >
      <Var Name="{4}" Forced="{5}" />
   </Event>
  <Var Name="{6}" Forced="{7}" />
</Request>
```

The response message contains a list of all elements being monitored including both events and variables. All variables associated with an event will be listed as a subitem of the *Event* element. In addition, override status of variables is also returned in the *Forced* field.

E. Read Values of Watching Elements

To trace triggered events, the following command is used:

```
<Request ID="{0}" Action="READ">
```

```
  <Event Name="{1}" Type="{2}" With="{3}" />
</Request>
```

The response message is formatted as:

```
<Response ID="{0}" Reason="{1}">
  <Event Name="{2}" Lastts="{3}" >
   <Var Name="{4}" Forced="{5}" />
  </Event>
</Request>
```

Where {1} indicates result of the request message; {3} refers to the last triggered time of this event; {4} refers to associated variable name; {5} refers to associated variable value; {6} refers to If this variable is overridden by a forced value.

If an event is never triggered, the *Lastts* field will be returned as "0". To read all associated variables of this event at the same time, the *With* field should be filled.

Similarly, to read data from an individual variable,

```
<Request ID="{0}" Action="READ">
  <Var Name="{1}" Type="{2}" />
  <Var Name="{1}" Type="{2}" />
  ……
</Request>
```

And corresponding response message is:

```
<Response ID="{0}" Reason="{1}">
  <Var Name="{2}" Value="{3}" Forced="{4}" />
  <Var Name="{2}" Value="{3}" Forced="{4}" />
  ……
</Request>
```

Multiple requests of reading variables could be composed as single request to improve efficiency.

F. Write Values to Watching Elements

To write values to watching elements, the following command shall be used:

```
<Request ID="{0}" Action="WRITE">
   <Event Name="{1}" Type="{2}">
     <Var Name="{3}" Type="{4}" Value="{5}" />
  </Event>
  <Var Name="{6}" Type="{7}" Value="{8}" />
  ……
</Request>
```

The write requests can be a combination of events, associated data variables and individual variables. This request will only write once to target elements. When the target is an event, it will trigger the event and attached variables will be updated with new values.

G. Force Values to Watching Data Variables

Finally, data variables could be overridden to a constant value by using the following command:

```
<Request ID="{0}" Action="WRITE">
  <Var Name="{1}" Type="{2}" Value="{3}"
Forced="{4}" />
  ……
</Request>
```

Where {4} must be set to "True" to indicate adding force value to this variable. Multiple variables could be overridden in single request.

Forced values must be removed by using the following command:

```
<Request ID="{0}" Action="WRITE">
  <Var Name="{1}" Type="{2}" Forced="{3}" />
  ……
</Request>
```

Where {3} must be set to "False" to remove force value from this variable. Also, multiple forced variables could be cleared in single request.

V. Implementation and Case Study

The proposed extensions to management commands are implemented in the function block service runtime (FBSRT) [18] for testing purpose. The FBSRT is an IEC 61499 execution environment based on service-oriented architecture. Function blocks are created as software services with unique addresses that can be accessed directly using Web Services. In this section, the data acquisition process will be demonstrated based on a demo car manufacturing line.

The demo car manufacturing line is shown in Fig. 3. There are two assembly stations, one inspection station and two quality control stations in the assembly line. Once an order is received from the customer, two assembly stations are responsible for pick up the selected chassis and the body part for the assembly process. The completed model car is then sent to the inspection station for confirming whether the order is manufactured correctly. If the completed model car does not match the order, it will stop at the quality control station for human assistances. Once the order is completed, it will return via recycle line.

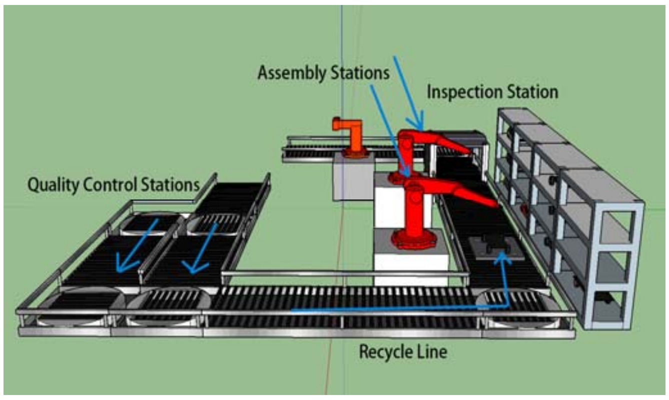

Fig. 3: Demo Car Manufacturing Line

Firstly, the client/server model is demonstrated. To monitoring all device status from SCADA systems, a sequence of management commands must be used. For example, to monitor the status of a conveyor *Conv1*, the following sequence of management commands shall be used.

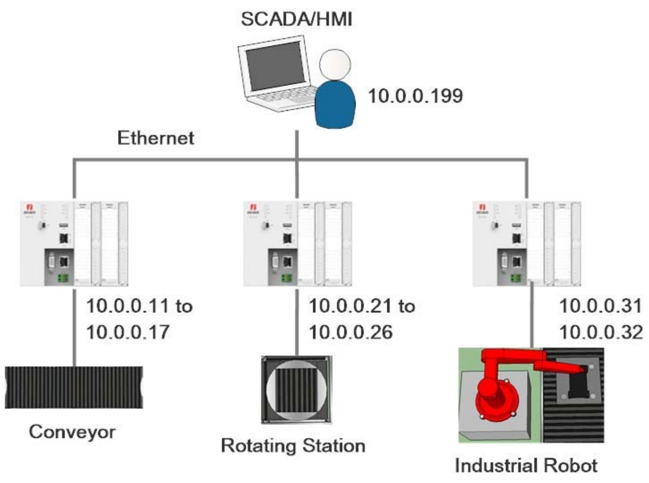

Fig. 4: Control Network Diagram

1) Assume the IP address for SCADA server is 10.0.0.199, to register conveyor status variable *Status* from FB instance FB_*Conv1* with Controller *RES1.DEV1*, the following request message is used.

```
<Request ID="1" Action="CREATE">
  <Watch Type="CS" Source="10.0.0.199">
    <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Type="Output" />
  </Watch>
</Request>
```

2) To confirm if the conveyor status variable is registered with the Controller *RES1.DEV1,* a query message shall be used.

```
<Request ID="2" Action="QUERY">
  <Watch Type="CS" Source="10.0.0.199" />
</Request>
```

The following response message is received for confirmation:

```
<Request ID="2" Reason="RDY">
     <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Forced="False" />
</Request>
```

3) Start Monitoring Process by sending the following management command to the Controller *RES1.DEV1* for authorization approval.

```
<Request ID="3" Action="START">
  <Watch Type="CS" Source="10.0.0.199" />
</Request>
```

4) Polling conveyor status variable from the Controller *RES1.DEV1* by:

```
<Request ID="4" Action="READ">
  <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Type="Output" />
</Request>
```

The controller will return the latest status of the variable by

```
<Response ID="4" Reason="RDY">
  <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Value="1" Forced="False" /> //Running Status
</Request>
```

The Step 4 shall be repeated periodically to update status of this conveyor. This sequence must be repeated for all conveyors to collect up-to-date device status. Once the conveyor status variable is no longer required by the SCADA system, the following sequence of management commands shall be used. Firstly, the monitoring process must be stopped.

```
<Request ID="5" Action="STOP">
  <Watch Type="CS" Source="10.0.0.199" />
</Request>
```

Then the conveyor status variable could be safely deregistered from the controller by:

```
<Request ID="6" Action="DELETE">
  <Watch>
    <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Type="Output" />
  </Watch>
</Request>
```

The IEC 61499 standard is designed based on event-trigger function blocks. Function block instances are only executed when there is an input event raised. This could be beneficial for publish/subscribe data model in this case. All monitored variables are only updated to the SCADA system if there is a change of their values will reduce network traffic significantly. The publish/subscribe model is used to monitor all conveyor status:

```
<Request ID="1" Action="CREATE">
  <Watch Type="PS" Source="10.0.0.199">
    <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Type="Output" />
     ...... //Other Conveyor Status
  </Watch>
</Request>
```

The confirmation and monitoring start commands are identical to Step 2 and 3 mentioned above. However, the watch type must be set to "PS" to indicate publish/subscribe. Once the monitoring process starts, there is no polling request message required. The controller will publish updated status if any. For example, when an order is received, conveyor *CONV1* and *CONV2* for assembly stations will start running. The following response message will be published:

```
<Response ID="4" Reason="RDY">
  <Var Name="RES1.DEV1.App1.FB_Conv1.Status"
Value="1" Forced="False" /> //Running Status
```

```
  <Var Name="RES1.DEV1.App1.FB_Conv2.Status"
Value="1" Forced="False" /> //Running Status
</Request>
```

As there is no change to other conveyors, the response message only contains the new status of two assembly stations. A quick performance comparison is measured for both data acquisition models. In the client/server model, the total time of reading a variable is approximately 250ms for a pair of a request and a response message. In the publish/subscribe model, the updated value will be pushed back automatically, the total time will reduce to about 30ms.

VI. Conclusions And Future Work

Industrial Cyber-Physical Systems coordinate various devices and equipment to provide intelligent control to existing industrial automation systems. Legacy industrial controllers are only providing data back to SCADA systems. To achieve cooperation in iCPS, real-time feedback data must be extracted from all devices directly. Existing IEC 61499 based industrial automation systems are not able to provide data interface due to lack of real-time data monitoring support. In this paper, several new elements and a set of new management commands are introduced in the compliance profile of the IEC 61499 to support real-time data monitoring. Two data acquisition models are implemented, one for OPC-like client/server model and the other for peer-to-peer publish/subscribe model. With these models, data could be retrieved anytime from any device or system directly.

Continuing from this work, different real-time requirements for industrial applications will be investigated. For high real-time constraint applications such as motion control, further performance analysis and improvement need to proceed and management commands need to be further optimized. Finally, integrating with OPC UA servers is also a compulsory requirement to be compatible with IEC 61131-3 based controllers and other high-level IT systems.

Acknowledgements

This research work is sponsored by Shanghai Pu Jiang Scholarship 2015 (No. 15PJ1403400) and National Science Foundation, China (No. 61503247).

References

[1] H. Kagermann, J. Helbig, A. Hellinger and W. Wahlster, "Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry", *Final Report of the Industrie 4.0 Working Group*, Forschungsunion, 2013

[2] A. Colombo, S. Karnouskos, and T. Bangemann. "Towards the next generation of industrial cyber-physical systems." *Industrial cloud-based cyber-physical systems*. Springer International Publishing, 2014. 1-22.

[3] B. Scholten, "The road to integration: A guide to applying the ISA-95 standard in manufacturing", *ISA*, 2007.

[4] T. Erl, "Service-oriented architecture: concepts, technology and design", Prentice Hall Professional Technical Reference, 760 pages, 2005.

[5] IEC 61499, Function Blocks, *International Standard*, Second Edition, 2012

[6] F. Jammes, B. Bony, P. Nappey, A. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka and M. Tilly, "Technologies for SOA-based Distributed Large Scale Process Monitoring and Control Systems", *38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2012.

[7] A. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes and J. Lastra, "Industrial cloud-based cyber-physical systems. *The IMC-AESOP Approach",* ISBN: 978-3-319-05623-4, 245pp, 2014.

[8] S. Leitner, and M. Wolfgang, "OPC UA–service-oriented architecture for industrial applications." *ABB Corporate Research Center*, 2006.

[9] F. Pethig, B. Kroll, O. Niggemann, A. Maier, T. Tack and M. Maag. "A generic synchronized data acquisition solution for distributed automation systems." *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2012.

[10] L. Durkop, H. Trsek, J. Otto and J. Jasperneite, "A field level architecture for reconfigurable real-time automation systems". *10th IEEE Workshop on Factory Communication Systems (WFCS)*, pp. 1-10, May 2014.

[11] I. Miyazawa, M. Murakami, T. Matsukuma, K. Fukushima, Y. Maruyama, M. Matsumoto, J. Kawamoto and E. Yamashita, "OPC UA information model, data exchange, safety and security for IEC 61131–3". *Proceedings of SICE Annual Conference (SICE)*, pp. 1556-1559, September, 2011.

[12] M. Melik-Merkumians, T. Baier, M. Steinegger, W. Lepuschitz, I. Hegny, and A. Zoitl, "Towards OPC UA as portable SOA middleware between control software and external added value applications.", *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1-8, September, 2012.

[13] F. Andrén, T. Strasser, A. Zoitl, and I. Hegny, "A reconfigurable communication gateway for distributed embedded control systems", *38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, pp. 3720-3726, October 2012.

[14] M. Wenger, A. Zoitl, and J. O. Blech, "Behavioral type-based monitoring for IEC 61499", *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1-8, September 2015.

[15] B. Matteo, F. Ferraris, C. Offelli, and M. Parvis, "A client-server architecture for distributed measurement systems.", *IEEE Transactions on Instrumentation and Measurement,* Vol. 47, no. 5, pp 1143-1148, 1998.

[16] A. Demers, G. Johannes, H. Mingsheng, R. Mirek, and W. Walker, "Towards expressive publish/subscribe systems.", *International Conference on Extending Database Technology*, pp. 627-644. Springer Berlin Heidelberg, 2006.

[17] A. Zoitl and R. Lewis, "IEC 61499 Compliance Profile for Feasibility Demonstrations", *Modelling Control Systems Using IEC 61499*, DOI: 10.1049/PBCE095E, 2014

[18] W. Dai, V. Vyatkin, J. Christensen, V. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Dynamic Reconfigurability", *IEEE Transactions on Industrial Informatics*, Vol. 11, No. 3, pp 771 - 781, 2015.

[19] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Third Edition, 2013