


Article

# OPC-UA Agent for Legacy Programmable Logic Controllers †

Seung-Yong Lee <sup>1</sup>  and Minyoung Sung <sup>2,\*</sup><sup>1</sup> Korea Electronics Technology Institute, Seongnam-si 13509, Korea<sup>2</sup> Department of Mechanical and Information Engineering, University of Seoul, Seoul 02504, Korea

\* Correspondence: mysung@uos.ac.kr; Tel.: +82-2-6490-2394

† A preliminary version of the paper will be presented at the International Conference on Emerging Technologies and Factory Automation, Stuttgart, Germany, 6–9 September 2022.

**Abstract:** Open platform communications (OPC) unified architecture (UA) is a communication standard increasingly used in industrial automation systems to enable the exchanging of control and management data between distributed entities. This paper proposes the design of an OPC-UA agent to enable UA information service and client functionalities in legacy programmable logic controllers (PLCs). The agent runs on a separate machine connected to the PLC using a dedicated link and maintains shared memory for certain variables in the PLC. Based on the periodically synchronized variables, the agent services the OPC-UA information model and executes client function blocks on behalf of the PLC. One important design feature is the remote procedure call of IEC 61131-3-based function blocks using synchronized variables. This allows the standard OPC-UA client functions to be used in existing PLCs which only support numeric types and do not support strings or complex structures. To validate the proposed design, we implement an agent prototype and demonstrate the successful monitoring and control of an industrial robot controller via OPC-UA. Through experiments, we evaluate the performance of UA functions in terms of the latency of read services for an increasing number of items. The evaluation results are believed to provide useful insights into agent-based approaches for integrating legacy PLCs into the OPC-UA framework.



**Citation:** Lee, S.-Y.; Sung, M. OPC-UA Agent for Legacy Programmable Logic Controllers. *Appl. Sci.* **2022**, *12*, 8859. <https://doi.org/10.3390/app12178859>

Academic Editors: Luigi Fortuna and Ephraim Suhir

Received: 15 July 2022

Accepted: 29 August 2022

Published: 3 September 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

**Keywords:** open platform communications; unified architecture; programmable logic controller; remote procedure call; agent

## 1. Introduction

With the advances of industrial Internet-of-Things (IoT) in recent years, open platform communications (OPC) unified architecture (UA) is attracting attention as a communication technology for automatic monitoring and control between machines. OPC-UA is a communication standard for the exchanging of control and management data between distributed entities in industrial automation systems [1]. It is standardized by IEC 62541 [2] and specifies reliable, secure, and interoperable communication between various automation devices such as programmable logic controllers (PLCs), human–machine interfaces (HMIs), and supervisory control and data acquisition (SCADA) systems. OPC-UA offers numerous advantages [3,4]. Its functionality is platform-independent and scalable. It supports comprehensive information modeling and allows for the design of service architecture with complex hierarchical data structures [5]. Moreover, OPC-UA provides enhanced security for session encryption and authentication.

To use OPC-UA in specific applications, each application domain has an associated companion specification [6]. The OPC-UA for IEC 61131-3 specification presents an information model for PLCs and defines function blocks for using OPC-UA services in PLC programs [7,8]. The information model describes the configuration, resources, tasks, and programs of an IEC 61131-3-based software in terms of OPC-UA objects, variables, and methods, and enables the PLC to be monitored and used by OPC-UA clients. PLCs may also need the OPC-UA client functionality to perform data exchanges horizontally with

other PLCs and/or vertically with higher-level machines such as SCADA and HMIs. The specification of an OPC-UA client for IEC 61131-3 defines function blocks as those used for connecting to an external OPC-UA server, accessing data either synchronously using read services or asynchronously using a subscription, and calling methods on remote servers.

Despite the advantages of OPC-UA, there are legacy PLCs with a significant installation base that cannot benefit from the technology [3,9–12]. This is because they do not support TCP/IP or lack the built-in computational functions required by IEC 61131-3 due to limited hardware or software capabilities. Typically, programs in legacy controllers are written in instruction lists (ILs) or ladder diagrams (LDs) that only support numeric types and do not support string or complex structure types. Even if the PLC has the necessary computing resources, including the OPC-UA, functionality in the PLC can have catastrophic computational side effects. Since PLCs perform real-time control functions, interference by the additional functions must be thoroughly investigated to ensure that the timing constraints of the control functions are satisfied [13,14]. Therefore, the approach of having a dedicated agent running OPC-UA on a separate machine could be a viable solution for these legacy systems.

This paper proposes the design of an OPC-UA agent to enable information service and client functionalities in legacy PLCs. The agent runs on a separate machine connected to the PLC using a dedicated link, and maintains shared memory for certain variables in the PLC. Based on the periodically synchronized variables, the agent services the OPC-UA information model and executes client function blocks on behalf of the PLC. One important design feature is that, using synchronized variables, the agent implements simplified remote procedure calls (RPC) of IEC 61131-3-based function blocks. This allows the standard OPC-UA client API to be used in existing PLCs that do not support strings or complex user-defined structures. To validate the proposed design, we implement an agent prototype and demonstrate the successful monitoring and control of an industrial robot controller via OPC-UA. Through experiments, we evaluate the performance of UA functions in terms of the latency of the read service as the number of items increases.

The remainder of the paper is organized as follows. In Section 2, we describe the OPC-UA specification and the IEC 61131-3 companion profile. In Section 3, we explain the design of an OPC-UA agent for legacy PLCs, and in Section 4, we present the evaluation results. Section 5 provides the conclusions of the paper.

## 2. Background

### 2.1. OPC Unified Architecture

OPC-UA defines an information model and a set of abstract services and describes the mapping to implementation technologies, such as binary coding over TCP/IP or XML over HTTP [15,16]. It is therefore platform-independent and supports different stack implementations to interoperate with each other in different programming and runtime environments. The client-side OPC-UA stack provides a programming interface, and the client application requests service by using the interface. The stack then generates and transmits a request message based on the service definition [17]. When the server-side OPC-UA stack receives the message, it replies with a response message containing data provided by the server application.

An information model is defined by *nodes* and *references* [3,16,18]. Each node has a set of *attributes*. A *node class* describes the node type and specifies the node attributes. Several base node classes are defined in the standard: *objects* are used to represent real-world entities, and *variables* and *methods* typically belong to objects and represent values and services, respectively. A reference describes the relationship between nodes. In OPC-UA, the node hierarchy is called the *address space* and is application-specific.

The OPC-UA service defines several functions for exchanging data between distributed devices, including service discovery, session establishment, node management, synchronous data access, and asynchronous data update [16]. In OPC-UA, variables are usually modeled as attributes, and *read* services are primarily used to access the attributes of

nodes synchronously. This service allows the client to specify a list of nodes and attributes and perform bulk reads with a request. It is generally the preferred service for reading variables. Another sophisticated method of data collection is to use a *subscription* service to receive the asynchronous updates of variables. A client creates a subscription with the desired publishing interval and then creates *monitored items* within the subscription by specifying node and attribute IDs, sampling intervals, and filters. The sampling interval defines how often the server checks for changes to variables and the filter specifies the conditions in which to put the sampled value into the data queue. The values in the queue create notifications in the next publishing cycle. When the server receives a publish request from the client, the notification then generates a response that delivers the updates.

## 2.2. OPC-UA for IEC 61131-3

A PLC is a computing system for industrial control and automation and has been standardized by IEC 61131 [5]. A PLC replaces the hardware such as relays, timers, and counters in conventional control panels with software. It receives input from sensors, processes control logic, and transmits outputs to external devices such as motor drives. IEC 61131-3 defines the model of PLC software, and describes the software in terms of *configurations, resources, tasks, and program organization units (POUs)* [19–21]. A configuration describes the hardware such as processors and memory and includes one or more resources. A resource can be considered to be a processing facility capable of executing PLC programs. A resource may contain one or more tasks, where a task controls the execution of a *program* and/or a set of *function* and *function blocks*. A task can be executed periodically or when a specified trigger, such as a variable change, occurs. A POU is the basic building block that contains data structures and algorithms. It can be a program, a function, or a function block. Generally, a program consists of functions and function blocks that pass data arguments.

The specification of OPC-UA for IEC 61131-3 defines server and client technologies for enabling OPC-UA communication in PLC software [7,8]. It explains the information model to describe the IEC 61131-3 software architecture and defines important object types that represent tasks, configurations, resources, and POUs. The OPC-UA client specification defines IEC 61131-3-compliant function blocks that allow PLC software to act as an OPC-UA client and communicate with external OPC-UA servers. These technologies enable the platform-independent, scalable, and secure communication of PLCs in industrial automation.

Figure 1 shows the OPC-UA information model for PLCs and depicts the main object types and their relationships. The IEC 61131-3 object types derive from OPC-UA core and device integration (DI)-level object types and include *CtrlTaskType*, *CtrlConfigurationType*, *CtrlResourceType*, and *CtrlProgramOrganizationUnitType*. The *CtrlTaskType* object type defines a PLC task with an execution cycle (Interval), scheduler priority (Priority), and trigger event (Single). When a trigger event is set, the task runs on the rising edge of the event regardless of its period. *CtrlConfigurationType* represents the PLC configuration and includes global variables (GlobalVars) and resources (Resources). *CtrlResourceType* expresses PLC resource information. It includes global variables, task information (Tasks), and programs (Programs). *CtrlProgramOrganizationUnitType* defines a POU and represents a program (*CtrlProgramType*) or function block (*CtrlFunctionBlockType*). Each object can have an input variable (InputVar), an output variable (OutputVar), or an input and output variable (InOutVar). Relationships between objects are defined by the reference types *HasSubType* or *HasComponent*, which represent derivation and containment relationships, respectively. The application-level objects are defined based on the object types.

The client function blocks support essential OPC-UA services, including service discovery, attributes, methods, monitored items, and subscription services. The function blocks for IEC 61131-3 are summarized in Table 1. The function blocks differ from the original OPC-UA API in that they use handles when nodes need to be specified for read and write operations. Taking *UA\_ReadList* as an example, Figure 2 shows a diagram

representation of a function block with types and names of input (left), output (right), and in–out (both) arguments.

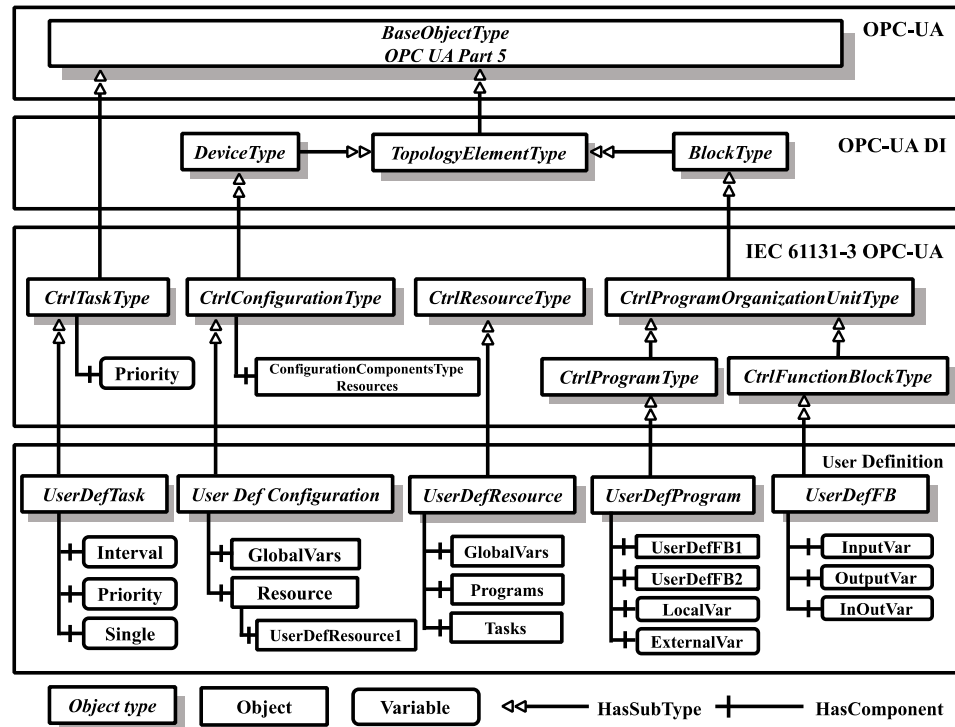


Figure 1. OPC-UA information model for IEC 61131-3.

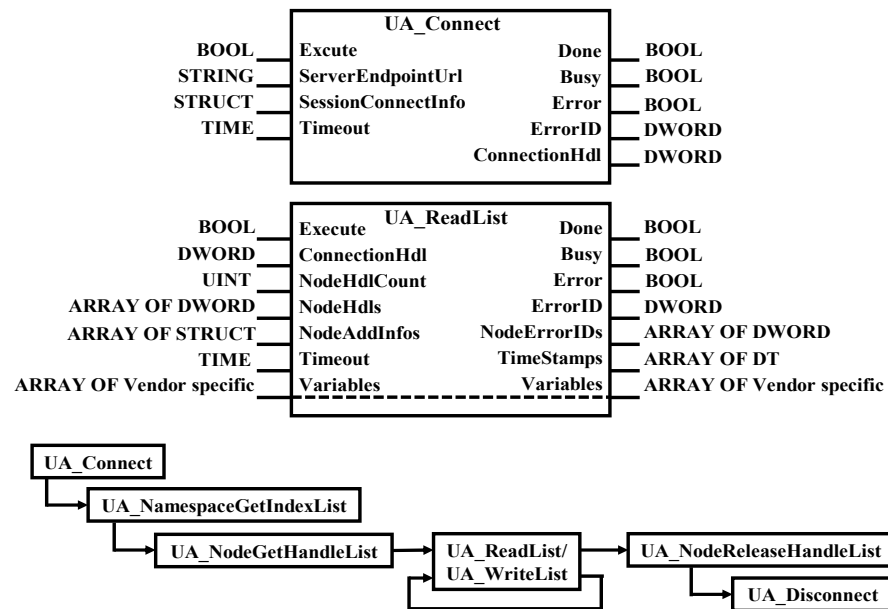


Figure 2. Use case of OPC-UA client function blocks and diagram representation of UA\_Connect and UA\_ReadList.

**Table 1.** OPC-UA client function blocks for IEC 61131-3.

Function Block Name	Description
UA_Connect	Create a connection of OPC-UA session.
UA_Disconnect	Close a connection of OPC-UA session.
UA_NamespaceGetIndexList	Acquire a list of indexes for the given namespaces.
UA_ServerGetUriByIndex	Acquire the server-URI with a given index.
UA_ServerGetIndexByUriList	Acquire a list of indexes for the server-URIs.
UA_TranslatePathList	Acquire a list of node IDs for the node paths.
UA_NodeGetHandleList	Acquire node handles for the given node IDs.
UA_NodeReleaseHandleList	Release a set of node handles.
UA_NodeGetInformation	Acquire the node information.
UA_SubscriptionCreate	Create a subscription.
UA_SubscriptionDelete	Delete a subscription.
UA_SubscriptionModify	Modify publish parameters of a subscription.
UA_SubscriptionProcessed	Check if monitored items have been published.
UA_MonitoredItemAddList	Add nodes for monitored items.
UA_MonitoredItemRemoveList	Remove monitored items from a subscription.
UA_MonitoredItemModifyList	Modify monitored items.
UA_MonitoredItemOperateList	Update the values of monitored items.
UA_ReadList	Read values of a list of nodes.
UA_WriteList	Write values of a list of nodes.
UA_MethodGetHandleList	Acquire handles for method calls.
UA_MethodReleaseHandleList	Release method handles.
UA_MethodCall	Call a method routine.
UA_Browse	Navigate through OPC-UA address space.
UA_EventItemAdd	Create events in a subscription.
UA_EventItemOperateList	Acquire a list of event information.
UA_EventItemRemoveList	Remove events from a subscription.
UA_HistoryUpdate	Insert or update data in the historical database.
UA_ConnectGetStatus	Acquire the connection status.

The use case of function blocks when using UA\_ReadList or UA\_WriteList to access data on a remote server is shown in Figure 2 [8]. The PLC software first calls UA\_Connect to connect to the server and open an OPC-UA session. If it needs the information model, it uses UA\_NamespaceGetIndexList to read the namespace, and it calls UA\_NodeGetHandleList with the node ID as an argument to acquire the handle of the node to read or write. The UA\_ReadList or UA\_WriteList function block is then used to perform a read or write operation using the node handle. When the data transfer is complete, UA\_NodeReleaseHandleList deallocates the node handle and UA\_Disconnect closes the session and terminates the connection.

### 2.3. Related Works

Thanks to its advantages over its predecessor, OPC Classic, OPC-UA has been used in a variety of industrial applications [4,22–24]. It has been used for the autoconfiguration of real-time Ethernet (RTE) systems [22]. A universal device discovery mechanism was introduced, and it was demonstrated that the parameters of an RTE system can be configured automatically. An OPC-UA-based information system was also proposed for condition monitoring [4]. Based on OPC-UA, an extensible architecture has been developed to provide device status information from heterogeneous field devices and sensors to enterprise-level services. A proof-of-concept implementation was presented with a case for the monitoring of devices in a mining environment. Device information collection using OPC-UA was also utilized for the purpose of the optimization of manufacturing processes [23]. OPC-UA was used to model device-level information, and a set of services were constructed to solve a constraint satisfaction problem for production plan computation. Recently, it has been shown that the comprehensive information-modeling capability of OPC-UA can be used to realize the plug and play of cyber–physical system (CPS) components [24]. The hardware implementation of OPC-UA has been proposed to enable OPC-UA support in

embedded field devices such as sensors and actuators [25,26]. The extensions of OPC-UA to support representational state transfer (REST), a popular architecture style for distributed applications, have also been studied [27].

In recent years, with increasing interest in time-sensitive networking (TSN), methods of utilizing OPC-UA for real-time communication have been actively studied [28]. A practical implementation of the OPC-UA TSN communication architecture was presented for a manufacturing system [29] and a static timing analysis of OPC-UA pub-sub has been carried out [14]. A simulation study has been conducted for using OPC-UA pub-sub and TSN for field-level hard real-time communication [30], and TSN traffic shaping has been proposed for OPC-UA field devices [31].

Efforts have been made to utilize OPC-UA in legacy devices and various environments [6,9–11,16,32,33]. Wrappers and proxies are an approach for migration from OPC Classic to modern OPC-UA systems [10]. An OPC-UA wrapper is a UA server with a Classic client interface which allows UA clients to access a Classic server, whereas an OPC-UA proxy is a UA client with a Classic server interface that allows a Classic client to access a UA server. Some studies have proposed a standalone OPC-UA wrapper to minimize the impact on existing servers and have analyzed the effect of parameters such as number of items, publish interval, and sampling interval on performance [5]. For the integration of legacy devices, an approach using the OPC-UA companion model for ISA95 has been proposed [9]. A new layer based on OPC-UA was designed to map field device data to an ISA95-based information model and ensure the interoperability of an industrial CPS. The implementation of an OPC-UA interface for legacy PLC-based automation systems using a cloud computing service has also been studied [11]. In practice, for application in a specific field, the creation of an OPC-UA companion specification requires a cross-domain collaborative modeling process, which often results in the creation of an inconsistent set of nodes. Studies have shown that model-based development using the Eclipse modeling framework is effective in resolving inconsistencies [33]. In an industrial CPS, model interoperability is crucial to manufacturing intelligence. A recent study proposed an OPC-UA compatible interface to ensure the interoperability of data analytics across heterogeneous devices [6]. The interface is designed to enable the exchange of predictive model markup language (PMML), a domain-independent standard for representing XML-based data analytics models via OPC-UA.

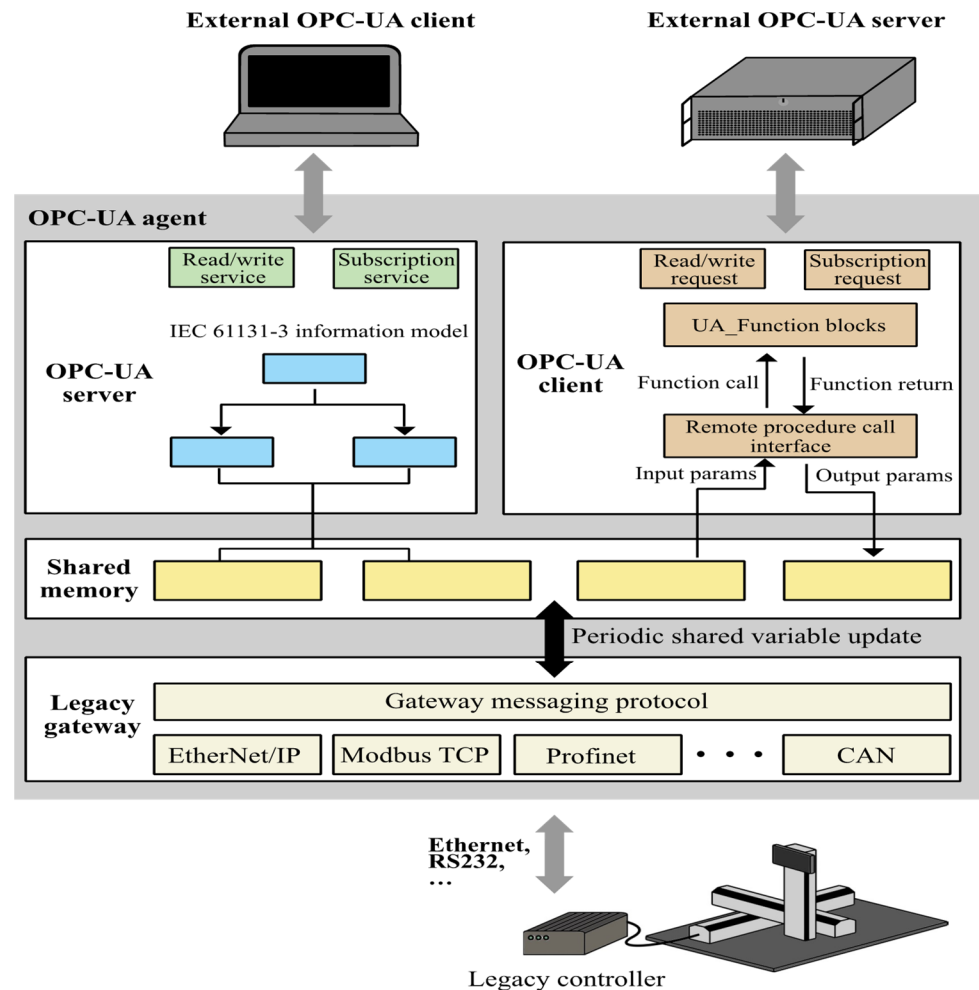
Some works carried out performance evaluation of OPC-UA. The impact of OPC-UA parameters was analyzed for read and subscription services [15], and the assessment of different OPC-UA implementations has been performed in industrial IoT-based measurement applications [32]. Recently, the performance of OPC-UA has been evaluated and compared with MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol) in a study on the performance analysis of IoT protocols [34].

There have been studies to enable OPC-UA in PLCs [11,35]. However, there have been little work done to support OPC-UA in legacy PLCs that lack the functional support or computational resources required for OPC-UA. This study is the first, in our knowledge, to propose an agent architecture that enables the use of OPC-UA client API based on the IEC 61131-3 standard in PLCs that do not support strings or complex user-defined structures [36]. In our work, we present the details of agent design and the results of experimental evaluation using prototype implementations. We believe that this work provides useful insights into agent-based approaches for integrating legacy PLCs into the OPC-UA framework.

### 3. OPC-UA Agent for Legacy Programmable Logic Controllers

Our agent design consists of three components running concurrently that share memory with each other: the legacy gateway, UA server, and UA client. Figure 3 shows the organization of an OPC-UA agent. The legacy gateway maintains images of certain PLC variables in the shared memory area. It communicates with the PLC and synchronizes the variable values. The UA server manages the IEC 61131-3 information model of the legacy

controller and handles service requests from external UA clients. The UA client contains the OPC-UA client stack and is responsible for the remote execution of function blocks [16]. When it receives an RPC request from the PLC, it identifies the requested function block, constructs arguments from the shared variables, and starts executing the corresponding API function in the UA client stack. When the UA client completes the function, it stores the output parameters and return values in the shared memory and prepares the RPC response. The response is later transferred by the gateway to the legacy controller. The client functions in the agent support both the synchronous and asynchronous collection of data from external OPC-UA servers using read and subscription services, respectively.



**Figure 3.** Organization of an OPC-UA agent with three concurrently running components: the legacy gateway, UA server, and UA client.

### 3.1. OPC-UA Server

For the UA server to access the user variables defined in the PLC program, the agent maps all application variables to the shared memory. While the PLC program is running, the legacy controller periodically sends variable values to the UA agent, and the shared memory is updated accordingly. So, when an external UA client requests synchronous data access such as read or write to the agent, the UA server can immediately access the shared memory to service the request. Similarly, when an external client requests a subscription-based asynchronous update service, the UA server can create monitored items, sample variable values, and generate publish responses, based on the shared memory.

The UA server provides access to the information model and handles various services for external clients. An external client may request runtime information about a PLC program and some variables such as, for example, actual motor positions and I/O data. To

create an information model for the PLC, users provide an XML file of the PLC software structure that describes the configuration, resource, task, and POU. Figure 4 shows an XML example for the PLC software structure and the OPC-UA information model built from it. For instance, the Proj\_OPC object instance of CtrlConfigurationType node class in the figure has Resource1 object of CtrlResourceType class, and Resource1 has Task1 (CtrlTaskType) and Main\_prog (CtrlProgramType) object instances. The Main\_Prog object contains the local variables defined in the PLC program and the Task1 object has the Interval and Priority instances of the controller task. During initialization, the UA server parses the XML file and builds a corresponding IEC 61131-3 information model.

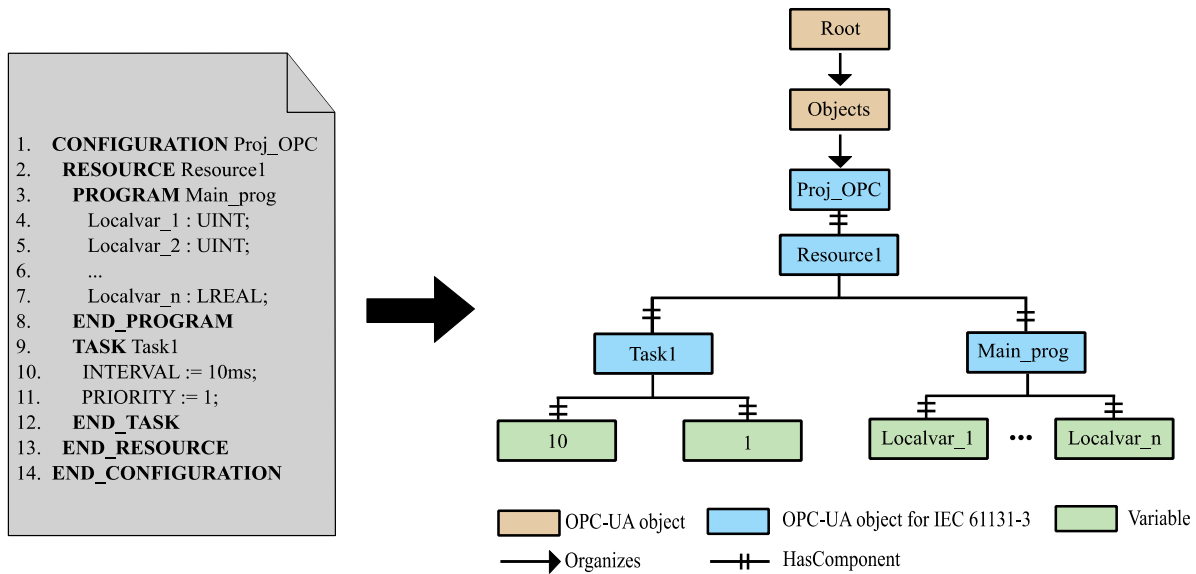


Figure 4. XML for the PLC software structure and the OPC-UA information model built from it.

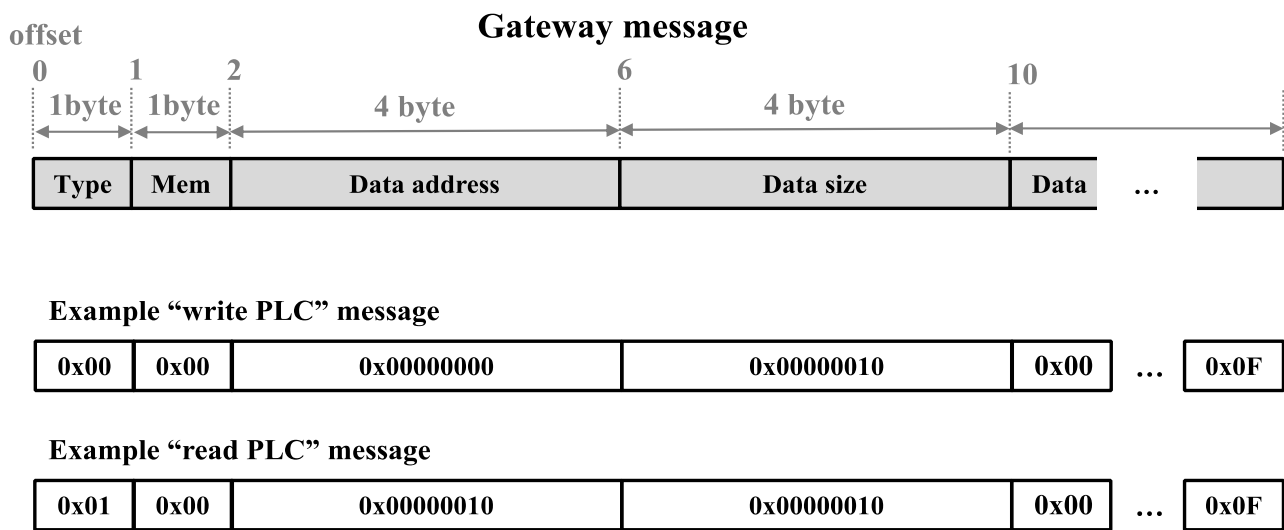
### 3.2. Legacy Gateway and Shared Memory

The gateway is responsible for maintaining images of certain PLC variables in a shared memory area. In general, PLCs provide a way for external hosts to access PLC variables for monitoring or programming purposes. The legacy gateway uses a dedicated link to communicate with the PLC and synchronizes the shared memory to keep variable values up to date. Shared memory is basically used to read and write node values in the IEC 61131-3 information model and is periodically synchronized with the PLC memory. In addition, the shared memory in our design stores the input and output parameters of UA function blocks, and is used collaboratively by the agent and PLC during the RPC process.

For external access, some PLCs support standard protocols such as EtherNet/IP, Profinet, or Modbus TCP/ UDP [24,37], while others have proprietary communication links and manufacturer-specific protocols. To address the diversity issue, we define a gateway messaging protocol that is used to communicate with the agent in a platform-independent manner.

Figure 5 shows the structure of a gateway message. The byte at offset 0 is the message type, and the values of 0 and 1 mean “write to” and “read from” the PLC, respectively. The byte at offset 1 determines the memory type: type 0 means the integer memory, while type 1 means the memory for floating-point numbers. The data address of the message indicates the address from which to start reading or writing in the shared memory. The data size at offset 6 is the number of data items, which is followed by data to be read or written in the message. Depending on the target PLC, the gateway protocol is implemented using a specific networking mechanism. It is noteworthy that the agent is designed to service more than one PLC if it has sufficient memory.



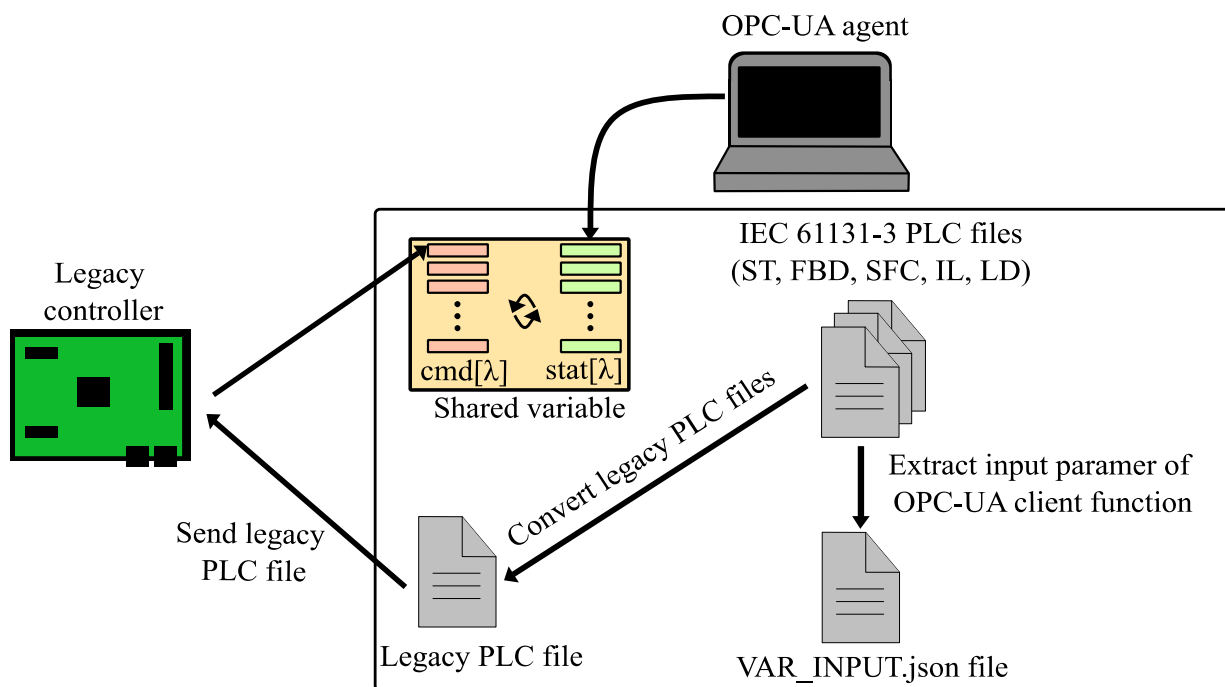


**Figure 5.** Structure of gateway messages used by the gateway messaging protocol for platform-independent communication between OPC-UA agent and PLC.

### 3.3. OPC-UA Client and RPC

In IEC 61131-3-based systems, users can write OPC-UA client codes using either textual programming languages such as structured text (ST) and instruction list (IL), or graphic languages such as the ladder diagram (LD), function block diagram (FBD), and structured flow chart (SFC) [5]. Legacy PLCs usually accept a single program file for execution and require a translator to convert the IEC 61131-3 code files into a PLC-executable program. Figure 6 shows the procedure. Many legacy controllers support only numeric variable types, i.e., bool, integer, and real types, and do not support derived or complex types such as array or structure. To support the parameters of types other than numeric types, the translator in our design generates VAR\_INPUT.json during the conversion process and stores input values for such non-numeric parameters of OPC-UA client functions. For example, the ServerEndPointUrl input in UA\_Connect (see Figure 2) specifies the URL or IP address of the OPC-UA server the program will connect to, and the VAR\_INPUT.json file stores the address value for ServerEndPointUrl.

The example in Figure 7 shows (a) an OPC-UA client application written in IEC 61131-3 ST and (b) the corresponding program converted into a PLC-executable script. The program receives the target position from an external OPC-UA server and operates a single-axis motor drive connected to the legacy controller. Lines 1-11 in Figure 7a define the variables that will be used in the rest of the PLC procedure. Next, line 14 shows the program calling UA\_Connect to create an OPC-UA session with an external server. During the conversion process, the address value for the server URL parameter is inserted into a table for the URL parameters in VAR\_INPUT.json. The index of the table entry is stored in the converted program and later used by the UA agent to look up the argument value when UA\_Connect is called. When the connection is established, UA\_NodeGetHandleList is called and a new handle (\_nHdls0) is created for the node that stores the target position (line 17). The legacy controller then reads the target position from the server by calling UA\_ReadList and stores it in the \_tarPos variable (line 20). The while loop in lines 19–24 repeats until the target position received is greater than 200. Then the UA\_ReadList function is no longer called, and the connection with the server is closed using UA\_Disconnect.



**Figure 6.** Procedure to convert IEC 61131-3 code files into a PLC-executable program.

For the RPC of OPC-UA client functions, a handshaking between the PLC and agent has been defined using shared variables. Each client function  $\lambda$  is associated with two state variables,  $cmd[\lambda]$  and  $stat[\lambda]$  (see Figure 6). The  $cmd[\lambda]$  variable is used by the PLC to command the start and end of  $\lambda$ . The  $stat[\lambda]$  shows the status of  $\lambda$  and can be modified by the agent. If  $cmd[\lambda] = 0$  and  $stat[\lambda] = 0$ , it means that  $\lambda$  is idle and not being used by the PLC. When PLC wants to call  $\lambda$ , it sets  $cmd[\lambda] = 1$ . The agent then acknowledges the call by setting  $stat[\lambda] = 1$  and starts executing  $\lambda$ . When  $\lambda$  completes, the agent sets  $stat[\lambda]$  to 2 or 3, to indicate that  $\lambda$  succeeded or failed, respectively. Then, the PLC resets  $cmd[\lambda] = 0$  to end, calling  $\lambda$ , and the agent confirms this and resets  $stat[\lambda] = 0$ . Figure 7b shows the PLC program and explains RPC handshaking. For  $\lambda_1 = UA\_Connect$ , the shared variables I100 and I128 represent  $cmd[\lambda_1]$  and  $stat[\lambda_1]$ , respectively (lines 9–10). For  $\lambda_2 = UA\_ReadList$ , I117 and I145 refer to  $cmd[\lambda_2]$  and  $stat[\lambda_2]$ , respectively. If  $cmd[\lambda_2] = 0$  and  $stat[\lambda_2] = 0$ , the input parameters are stored in I277~I279 (lines 16–23). Now, when X2, the variable mapped to an external input device, is set to 1, the agent executes  $\lambda_2$ . When  $\lambda_2$  finishes successfully ( $cmd[\lambda_2] = 1$  and  $stat[\lambda_2] = 2$ ), the target position value (I208) is copied to  $\_tarPos$  (lines 24–32).

```

1.begin procedure
2.  VAR
3.  _done:BOOL;
4.  _busy:BOOL;
5.  _error:BOOL;
6.  ...
7.  _nodeIDs:UANodeID_ARRAY;
8.  _connHdl:DWORD;
9.  _nodeHdls:DWORD;
10. _tarPos:REAL;
11. END_VAR

12. _nodeIDs[0].NamespaceIndex:=4;
13. _nodeIDs[0].Identifier:=000C01-B01949-000000-000000;
14. UA_Connect(Execute:=X0, ServerEndpointUrl:=‘192.168.10.163:4840’, Done=>_done, Busy=>_busy,
15.           Error=>_error, ConnectionHdl=>_connHdl);
16. IF _done AND !_error THEN
17.   UA_NodeGetHandleList(Execute:=X1, ConnectionHdl:=_connHdl, NodeIDCount:=1, NodeIDs:=_nodeIDs,
18.                       Done=>_done1, Busy=>_busy1, Error=>_error1, NodeHdls=>_nodeHdls0);
19.   WHILE _tarPos <= 200 DO
20.     UA_ReadList(Execute:=X2, ConnectionHdl:=_connHdl, NodeIDCount:=1, NodeHdls:=_nodeHdls,
21.               Done=>_done1, Busy=>_busy1, Error=>_error1, Variables=>_tarPos);
22.     MC_MoveAbsolute(Axis:=13, Execute:=X3, Position:=_tarPos, Velocity:=100.0, Done=>_done2, Busy=>_busy2,
23.                   Error=>_error2);
24.   END_WHILE
25. END_IF
26. UA_Disconnect(Execute:=X4, ConnectionHdl:=_connHdl, Done=>_done, Busy=>_busy, Error=>_error);
27.end procedure

```

(a) An OPC-UA client application written in ST.

```

1.begin procedure
2.  VAR
3.  _done:BOOL;
4.  _busy:BOOL;
5.  _error:BOOL;
6.  ...
7.  _tarPos:REAL;
8.  END_VAR

9.  IF I100 = 0 THEN
10.   IF I128 = 0 THEN
11.    I100 := X0;
12.   END_IF
13. END_IF
14. ...
15. WHILE _tarPos <= 200 DO
16.   IF I117 = 0 THEN
17.    IF I145 = 0 THEN
18.     I277 := 1;
19.     I278 := _nodeHdls;
20.     I279 := _connHdls;
21.     I117 := X2;
22.    END_IF
23.   END_IF
24.   IF I117 = 1 THEN
25.    IF I145 = 2 THEN
26.     I117 := 0;
27.     _tarPos := I208;
28.     _done1 := 1;
29.     _busy1 := 0;
30.     _error1 := 0;
31.    END_IF
32.   END_IF
33.   IF I117 = 1 THEN
34.    IF I145 = 1 THEN
35.     _done1 := 0;
36.     _busy1 := 1;
37.     _error1 := 0;
38.    END_IF
39.   END_IF
40.   IF I117 = 1 THEN
41.    IF I145 = 3 THEN
42.     _done1 := 0;
43.     _busy1 := 0;
44.     _error1 := 1;
45.    END_IF
46.   END_IF
47.   P1.EXE := X3;
48.   MABS P1;
49.   P1 I3 _tarPos 0.0 0.0 0.0 0.0 100.0 0.0 0.0;
50.   _done2 := P1.DONE;
51.   _busy2 := P1.BUSY;
52.   _error2 := P1.ERROR;
53. END_WHILE
54. END_IF
55. IF I101 = 0 THEN
56.   IF I129 = 0 THEN
57.    I101 := X4;
58.    I102 := _connHdls;
59.   END_IF
60. END_IF
61. IF I101 = 1 THEN
62.  ...
63.end procedure

```

(b) A PLC-executable script converted from the ST code above.

Figure 7. An OPC-UA client application written in ST and its converted PLC-executable script.

#### 4. Performance Evaluation

This section evaluates the performance of OPC-UA functions by the agent. We investigate the impact of the number of data items requested by external clients on the UA server. We also measure the response time of the UA client function blocks called from legacy PLC

programs. Response time is generally defined as the elapsed time from when a client sends a request to when it receives a response from the server. It is mainly used as a measure to evaluate the performance of OPC-UA servers with client–server architecture [5,15]. We conduct the performance evaluation by focusing on the trend of response time according to the number of items included in a request.

For performance evaluation, an experimental setup was configured using a legacy controller, an OPC-UA agent, and a user host to be used as an external OPC-UA server or client as shown in Figure 8. The OPC-UA agent and the user host were set up using open62541, a widely used open-source OPC-UA stack [17,38]. It was chosen because it supports almost all OPC-UA features and can be easily ported to any hardware or software platform [39,40].

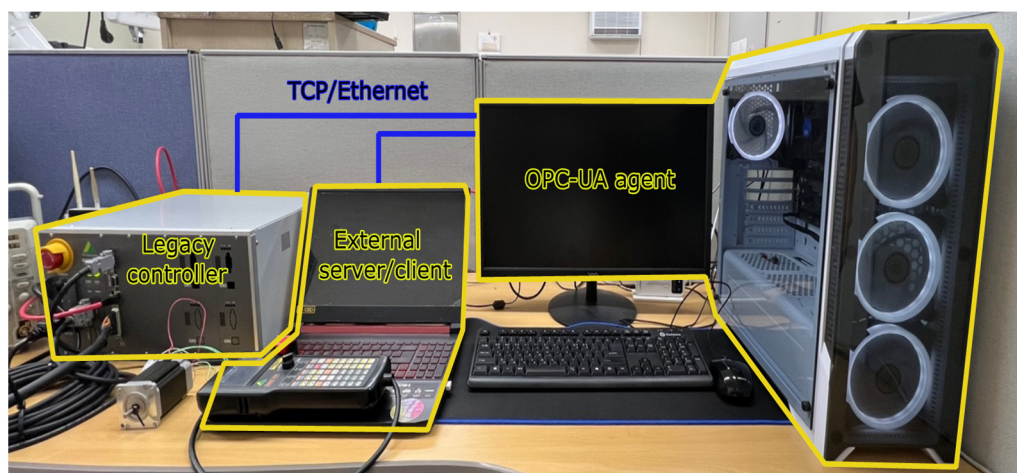


Figure 8. Experimental setup used for performance evaluation.

#### 4.1. Experimental Setup

When building the OPC-UA system, the user host was configured to have only core namespace nodes, and the agent host was set up to support the information model for IEC 61131-3 and the UA client function blocks shown in Table 1. The legacy controller is an embedded system that runs PLC programs without an operation system. It controls servo motors and handles 20-channel digital I/O. Table 2 summarizes the specifications of the hardware and software used in the experiment. The OPC-UA agent connects to the legacy controller and the external UA server or client over the TCP/Ethernet network.

Table 2. Specifications of experimental system.

System	Item	Description
External OPC-UA system	CPU	Intel(R) Core(TM) i5-9300H CPU @2.40 GHz
	Main memory	8.00 GB DDR4 DRAM
	Operating system	Windows 11 Pro
	OPC-UA software	open62541 v1.1 by open62541 community project
	Network	Realtek Gaming GbE Family Controller
OPC-UA agent	CPU	Intel(R) Core(TM) i5-8500 CPU @3.00 GHz
	Main memory	8.00 GB DDR4 DRAM
	Operating system	Windows 10 Home
	OPC-UA software Network	open62541 v1.1 by open62541 community project Intel(R) Ethernet Connection(7) I219-V
Legacy controller	CPU	Arm Cortex-R4F-based TI RM46L852 MCU
	Main memory	16 MB flash memory and 32 KB NVRAM
	Network	WIZnet W5300 Ethernet controller

Based on the OPC-UA agent design, we have defined performance metrics for the OPC-UA server and client as shown in Figure 9. The UA server agent can handle various services for accessing variables in the legacy PLC. One of those services, UA read service, can be used synchronously to access data items by an external UA client. To evaluate the read performance, we define the latency for read service ( $L_{read}$ ) and the response time of the UA server agent ( $R_{server}$ ).  $L_{read}$  means the time from when the external client sends a read request to when the server agent sends a read response.  $R_{server}$  is defined as the total time required by the OPC-UA server to process the read service. So, the terms  $L_{read}$  and  $R_{server}$  can be calculated as:

$$L_{read} = t_4 - t_1, \tag{1}$$

$$R_{server} = t_3 - t_2. \tag{2}$$

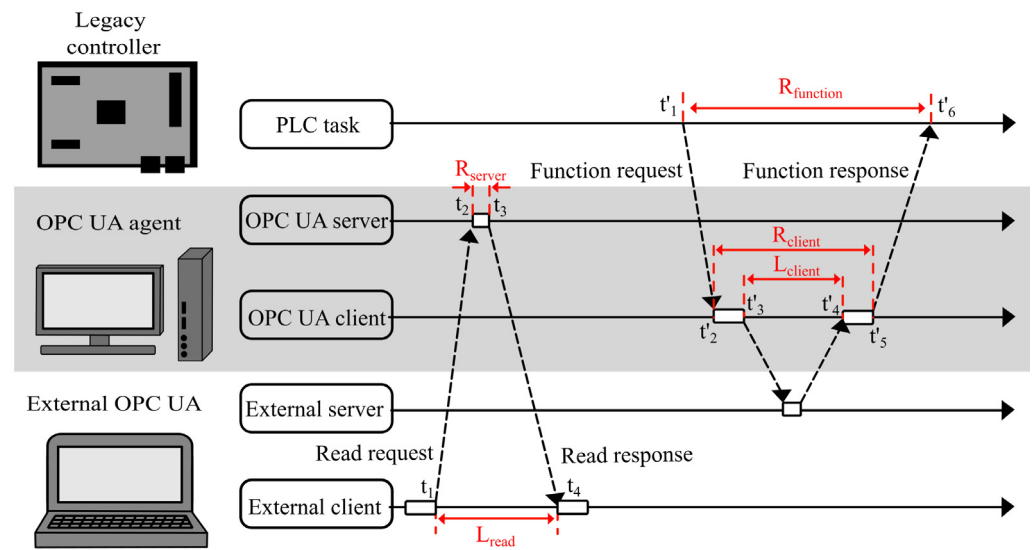


Figure 9. Sequence diagrams for the OPC-UA server and client functions by the agent.

The time  $t_1$  is when the external client sends a UA read request to the agent, and  $t_2$  and  $t_3$  represent the time when the agent receives the UA read request, and when it sends the response to the external client, respectively. The  $t_4$  represents the time the external client receives the response data sent by the UA server in the agent.

In the legacy controller, OPC-UA client function blocks can be executed along with other PLC codes. For the evaluation of client function blocks, we define performance measures considering the execution of RPC-based client functions. The response time of a client function in the legacy controller,  $R_{func}$ , is defined as the total time elapsed from when the client function is called in the PLC task until when the RPC response from the OPC-UA agent is delivered back to the PLC task. The time required to execute the client function in the agent,  $R_{client}$ , means the total time it takes to recognize the call request, process it, and update the result in the shared memory. Similarly,  $L_{client}$ , the latency for a client function, is defined as the time it takes the UA client agent to receive its response after requesting a service from the external server. So,  $R_{func}$ ,  $R_{client}$  and  $L_{client}$  are calculated as

$$R_{func} = t'_6 - t'_1, \tag{3}$$

$$R_{client} = t'_5 - t'_2, \tag{4}$$

$$L_{client} = t'_4 - t'_3. \tag{5}$$

The time  $t'_1$  is when the cmd[λ]-shared variable for the UA client function requested by the PLC task is set to 1 and  $t'_2$  is the time when the OPC-UA client confirms the change of cmd[λ]. The  $t'_3$  and  $t'_4$  represent the time when the OPC-UA client requests the necessary

service to the external server and receives the response from the server, respectively. The time  $t'_5$  is when the  $stat[\lambda]$ -shared variable is changed to indicate that the requested function has completed execution, and  $t'_6$  is the time when the legacy controller acknowledges the change of the  $stat[\lambda]$ -shared variable by the OPC-UA agent.

#### 4.2. Read Service of OPC-UA Server

To evaluate the design of the OPC-UA server supporting the IEC 61131-3 information model, we first verified the service of the OPC-UA agent using UaExpert from Unified Automation GmbH [41]. It is a full-featured OPC-UA client software that offers a free trial version. Figure 10 shows the information model for a PLC program developed for verification when viewed using the UaExpert client. It can be seen that the information model for the PLC program has been successfully created in compliance with the IEC 61131-3 specification. The information related to the PLC task has been created as the Task1 object instance, and the variables defined in the PLC program are displayed in the Main\_prog object instance.

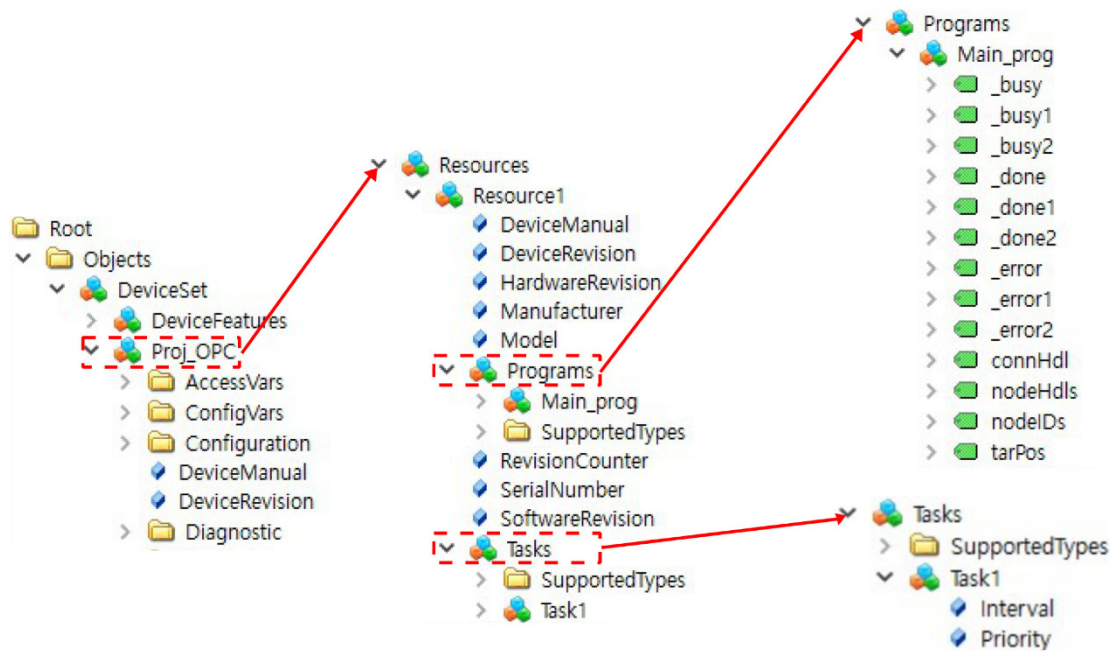
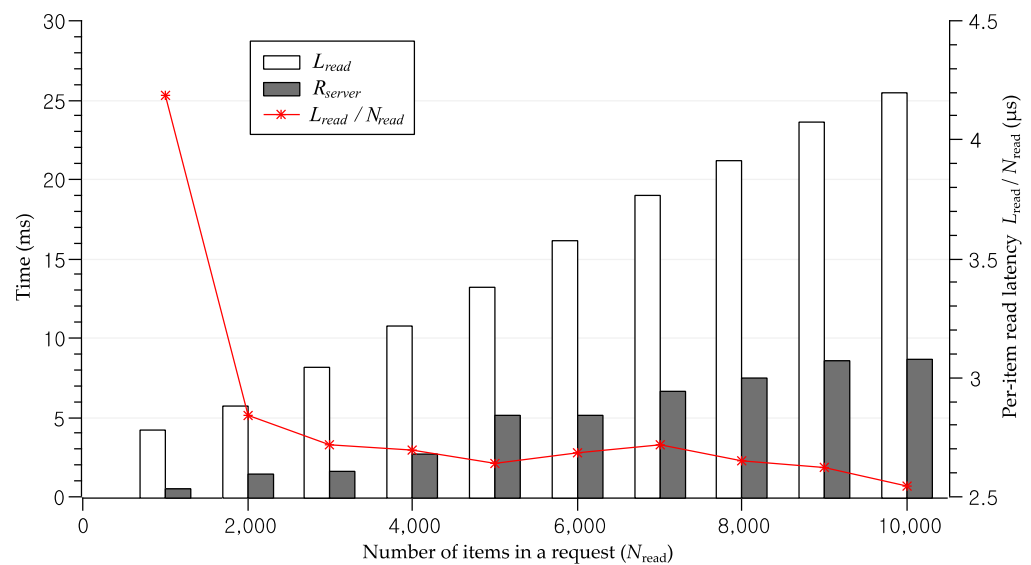


Figure 10. IEC 61131-3 Information model of a PLC program viewed using UaExpert client program.

Next, we investigate the latencies for read service and by the OPC-UA server. Let  $N_{read}$  denote the number of requested data items when the read service is called by the external client. Figure 11 shows the measurement results for  $L_{read}$  and  $R_{server}$  according to  $N_{read}$ . As  $N_{read}$  increases,  $R_{server}$  increases almost linearly overall, while the  $L_{read}$  increase is relatively small. In addition, we found that the per-item latency,  $L_{read}/N_{read}$ , decreases rapidly as  $N_{read}$  grows to 2000. Therefore, given the trend of  $L_{read}/N_{read}$ , it turns out that 3000–4000 data items are best suited when an external client requests a read service from the OPC-UA agent.



**Figure 11.** Average latencies for read service ( $L_{read}$ ) and response time of the OPC-UA server ( $R_{server}$ ) according to the number of data items ( $N_{read}$ ) in a single request.

#### 4.3. OPC-UA Client Function Blocks for IEC 61131-3

To evaluate the OPC-UA client function block, the response time of the client function called from the PLC program was measured. Among the 28 client APIs in Table 1, the function blocks using the number of node items as an input argument were set to have a value of 1000. The legacy controller was programmed to run a single PLC task with a period of 1 ms. To handle the subscription service, the external server must set both the publishing interval and sampling interval. The sampling interval is the rate at which the external server checks the values of monitored items in the subscription. If a publish request is sent from the OPC-UA client agent to the external server after the variable value is changed, the response is delivered to the client in the next publishing cycle. In the experiment, the external server set the publish interval to 500 ms and the sampling interval to 250 ms, respectively.

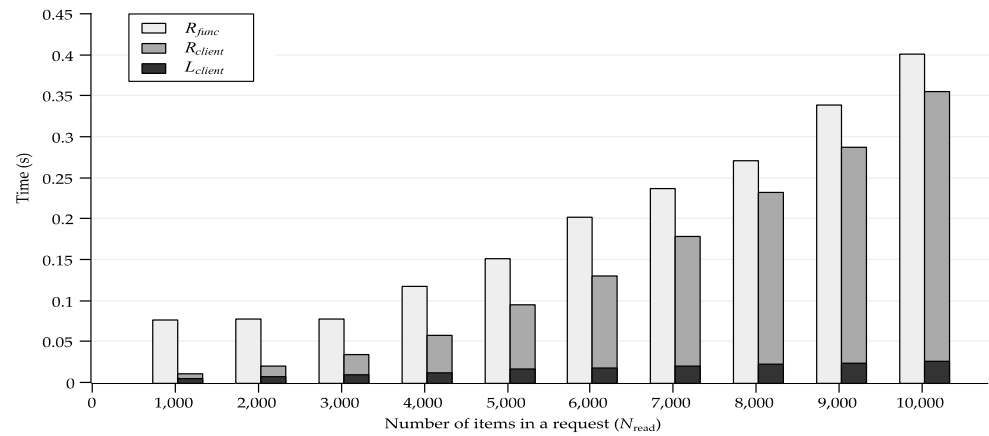
Table 3 summarizes the measurement results of  $R_{func}$  and  $R_{client}$  for each client function block. The averages of  $R_{func}$  for all function blocks were found to be very similar. However, the averages of  $R_{client}$  were measured to be significantly different depending on the client function. This is because shared memory is updated periodically and  $R_{func}$  is strongly affected by the update cycle of shared memory. Therefore, it can be seen that  $R_{func}$  can be improved significantly by reducing the update cycle of shared memory. It was also found that the  $R_{client}$  of UA\_Connect, which is used to establish a new connection of an OPC-UA session, is large compared to other function blocks. The response times for UA\_MethodGetHandleList, UA\_MethodReleaseHandleList, and UA\_EventItemOperateList are very small because the functions are executed without communication with the external server.

**Table 3.** Average response time of OPC-UA client functions.

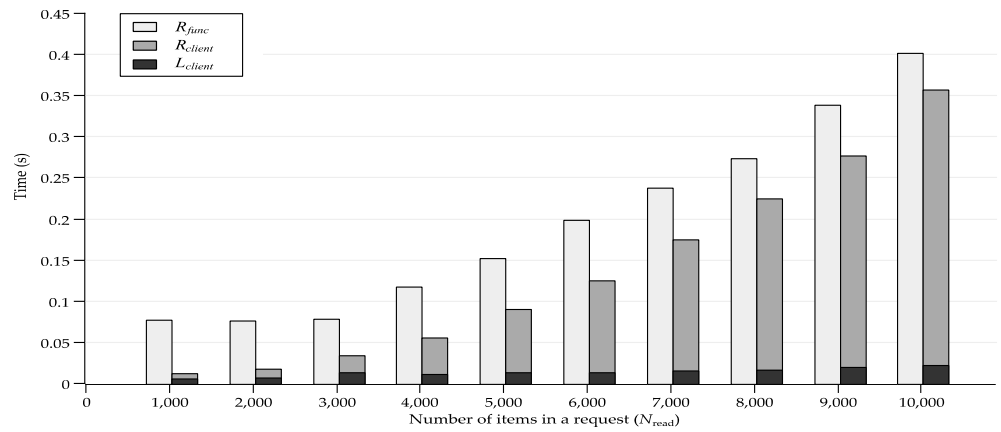
Function Block	Function Response Time (ms) (Average, Std. Dev.)		
	$R_{func}$	$R_{client}$	
Function block without the number of nodes argument	UA_Connect	79.09, 1.23	26.93, 8.09
	UA_Disconnect	79.30, 0.47	3.90, 0.63
	UA_TranslatePathList	78.25, 2.72	8.10, 1.75
	UA_NodeGetInformation	77.13, 5.44	14.57, 5.94
	UA_SubscriptionCreate	75.50, 7.97	1.70, 0.56
	UA_SubscriptionDelete	75.05, 8.74	14.15, 1.08
	UA_SubscriptionModify	77.28, 6.49	2.57, 0.46
	UA_SubscriptionProcessed	74.68, 9.14	8.69, 3.59
	UA_Browse	77.82, 6.69	5.85, 0.47
	UA_ConnectGetStatus	77.41, 6.76	3.67, 1.77
	UA_NamespaceGetIndexList	77.06, 6.41	4.20, 2.06
	UA_ServerGetUriByIndex	77.00, 6.16	2.65, 0.09
	UA_ServerGetIndexByUriList	75.70, 6.02	4.54, 0.40
	Function block having the number of nodes argument	UA_NodeGetHandleList	77.05, 6.19
UA_NodeReleaseHandleList		76.20, 7.15	2.59, 0.26
UA_MonitoredItemAddList		77.29, 6.09	19.95, 0.89
UA_MonitoredItemRemoveList		77.47, 6.35	17.26, 6.35
UA_MonitoredItemModifyList		77.50, 6.56	27.00, 2.31
UA_MonitoredItemOperateList		76.80, 6.10	7.35, 0.33
UA_ReadList		76.00, 7.33	10.51, 1.87
UA_WriteList		78.05, 4.70	12.31, 0.48
UA_MethodGetHandleList		77.95, 4.53	0.97, 0.26
UA_MethodReleaseHandleList		78.61, 4.67	0.05, 0.02
UA_MethodCall		77.31, 5.52	8.08, 2.25
UA_EventItemAdd		77.67, 5.89	1.59, 0.35
UA_EventItemOperateList		76.24, 7.92	0.06, 0.01
UA_EventItemRemoveList		76.00, 7.14	1.81, 0.16
UA_HistoryUpdate	76.00, 7.33	11.38, 2.58	

Among frequently used function blocks, the response time of UA\_ReadList, UA\_WriteList, and UA\_MonitoredItemAddList may vary depending on  $N_{read}$ , the number of node items. In particular, it is interesting to analyze the trend of response times,  $R_{client}$  and  $L_{client}$  according to the number of items, and discuss the time required for function execution and data exchange with an external server. Figure 12 shows the experiment results. It can be seen that  $R_{func}$  is highly affected by the shared memory update cycle because  $R_{client}$  and  $L_{client}$  are very small until  $N_{read}$  increases to 3000. As  $N_{read}$  increases gradually, it takes time to dynamically create the requested nodes and update the response variables before and after executing the client function, so it was observed that  $R_{client}$  increases as rapidly as  $N_{read}$  increases. The latency  $L_{client}$  of UA\_MonitoredItemAddList is always larger than the other two function blocks. This is because more processing is required for data exchange such as for the variables, status, and timestamps, as well as the node list in the subscription service.

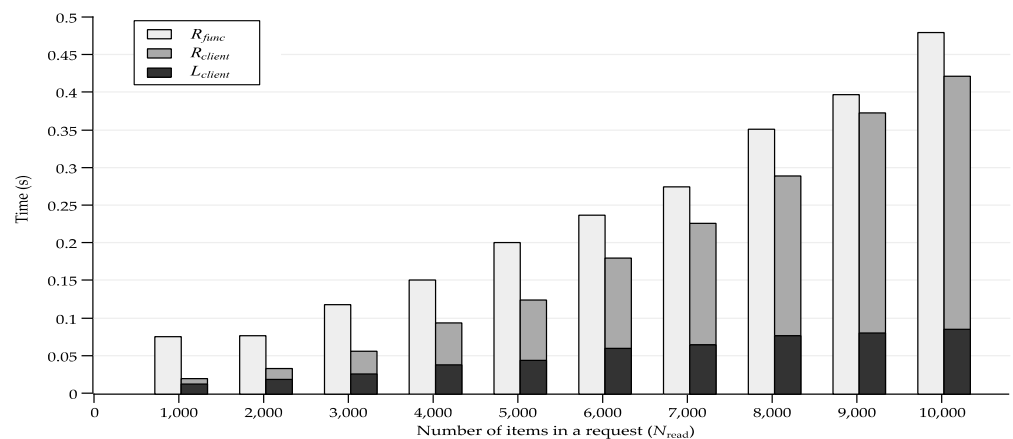




(a) UA\_ReadList.



(b) UA\_WriteList.



(c) UA\_MonitoredItemAddList.

**Figure 12.** Average response times and latencies of UA\_ReadList, UA\_WriteList, and UA\_MonitoredItemAddList according to the number of data items in a single request.

### 5. Conclusions

In this paper, we proposed the design of an OPC-UA agent. By running the OPC-UA server and client functions on a separate machine, the agent allows legacy PLCs to support an OPC-UA-based information service in a seamless and secure manner and to remotely access external OPC-UA servers. The agent is connected to the PLC using a dedicated link and maintains shared memory for specific variables in the PLC. Based on the periodically

synchronized variables, the agent serves the IEC 61131-3 information model and executes client function blocks on behalf of the PLC.

Our agent design consists of three components running concurrently that share memory with each other: the legacy gateway, UA server, and UA client. The legacy gateway communicates with the PLC and maintains an image of certain PLC variables in a shared memory area. The UA server manages the IEC 61131-3 information model of legacy controllers and handles service requests from external UA clients. The UA client contains the OPC-UA client stack and is responsible for the RPC of function blocks. The shared memory is periodically synchronized with the PLC memory and is primarily used to read and write node values in the IEC 61131-3 information model. In our design, the shared memory also stores the state of the UA function blocks and their input and output parameters. For the RPC of OPC-UA client functions, each client function is associated with two state variables, and the agent and PLC collaboratively use the variables during RPC handshaking. It is notable that the agent allows standard OPC-UA client functions to be used in PLCs that do not support strings or complex structures. The translator in our design generates a mapping table during the conversion process and stores input values for such non-numeric parameters of OPC-UA client functions in the table. The index of the table entry is stored in the converted program and later used by the UA agent to look up argument values when the client function is called.

Through experiments, we have validated the OPC-UA agent design and evaluated the performance of the OPC-UA server and client of the agent. We evaluated the read service of the OPC-UA server according to the number of nodes and found that the most appropriate number of nodes is about 3000 to 4000 in a single request. In the OPC-UA client, we measured the response time for 28 client APIs and observed the response times and latency depending on the number of nodes for the UA client functions frequently used for data exchange.

In our future research, we will study how to further improve the response time and latency in consideration of legacy controllers with different hardware and software. One of the key findings is that the update cycle of shared memory has a large impact on response time when the number of data items in a read or write is small. We will study a scheme to maximize performance by automatically tuning the configurable parameters of the OPC-UA agent, including the update cycle.

**Author Contributions:** Conceptualization, S.-Y.L. and M.S.; methodology, M.S.; software, S.-Y.L.; validation, S.-Y.L.; writing—original draft preparation, S.-Y.L. and M.S.; writing—review and editing, M.S.; visualization, S.-Y.L.; supervision, M.S.; project administration, S.-Y.L.; funding acquisition, M.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the 2021 Research Fund of the University of Seoul.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Sauter, T.; Lobashov, M. How to Access Factory Floor Information using Internet Technologies and Gateways. *IEEE Trans. Ind. Inform.* **2011**, *7*, 699–712. [[CrossRef](#)]
2. International Electrotechnical Commission. *OPC Unified Architecture—Part 1: Overview Concepts, Rev. 2.0*; International Electrotechnical Commission: Geneva, Switzerland, 2016.
3. Mahnke, W.; Leitner, S.-H.; Damm, M. *OPC Unified Architecture*; Springer: Berlin, Germany, 2009.
4. Hastbacka, D.; Barna, L.; Karaila, M.; Liang, Y.; Tuominen, P.; Kuikka, S. Device Status Information Service Architecture for Condition Monitoring Using OPC UA. In Proceedings of the IEEE Conference on Emerging Technology and Factory Automation, Barcelona, Spain, 16–19 September 2014; pp. 1–7.

5. Kim, W.; Sung, M. Standalone OPC UA Wrapper for Industrial Monitoring and Control Systems. *IEEE Access* **2018**, *6*, 36557–36570. [[CrossRef](#)]
6. Shin, S.-J. An OPC UA-Compliant Interface of Data Analytics Models for Interoperable Manufacturing Intelligence. *IEEE Trans. Ind. Inform.* **2021**, *17*, 3588–3598. [[CrossRef](#)]
7. PLCopen and OPC Foundation. *OPC UA Information Model for IEC 61131-3, Rev. 1.02*; International Electrotechnical Commission: Geneva, Switzerland, 2020.
8. PLCopen and OPC Foundation. *PLCopen OPC-UA client for IEC 61131-3, Ver. 1.1*; PLCopen: Gorinchem, The Netherlands, 2016.
9. Givehchi, O.; Landsdorf, K.; Simoens, P.; Colombo, A.W. Interoperability for Industrial Cyber-Physical Systems: An Approach for Legacy Systems. *IEEE Trans. Ind. Inform.* **2017**, *13*, 3370–3378. [[CrossRef](#)]
10. Hannelius, T.; Salmenpera, M.; Kuikka, S. Roadmap to adopting OPC UA. In Proceedings of the IEEE Conference on Industrial Informatics, Daejeon, Korea, 13–16 July 2008; pp. 756–761.
11. Hascamp, H.; Orth, F.; Wermann, J.; Colombo, A.W. Implementing an OPC UA interface for legacy PLC-based automation systems using the Azure cloud: An ICPS-architecture with a retrofitted RFID system. In Proceedings of the IEEE Industrial Cyber-Physical Systems Conference, St. Petersburg, Russia, 15–18 May 2018; pp. 115–121.
12. Chivilikhin, D.; Patil, S.; Chukharev, K.; Cordonnier, A.; Vyatkin, V. Automatic State Machine Reconstruction From Legacy Programmable Logic Controller Using Data Collection and SAT Solver. *IEEE Trans. Ind. Inform.* **2020**, *16*, 7821–7831. [[CrossRef](#)]
13. Cereia, M.; Bertolotti, I.C.; Scanzio, S. Performance of a Real-Time EtherCAT Master under Linux. *IEEE Trans. Ind. Inform.* **2011**, *7*, 678–687. [[CrossRef](#)]
14. Denzler, P.; Fruhwirth, T.; Kirchberger, A.; Schoeberl, M.; Kastner, W. Static timing analysis of OPC UA PubSub. In Proceedings of the IEEE Conference on Factory Communication Systems, Linz, Austria, 9–11 June 2021; pp. 167–174.
15. Cavalieri, S.; Chiacchio, F. Analysis of OPC UA Performances. *Comput. Stand. Interfaces* **2013**, *36*, 165–177. [[CrossRef](#)]
16. Birrell, A.D.; Nelson, B.J. Implementing Remote Procedure Calls. *ACM Trans. Comput. Syst.* **1984**, *2*, 39–59. [[CrossRef](#)]
17. Palm, F.; Gruner, S.; Pfrommer, J.; Graube, M.; Urbas, L. Open Source as Enabler for OPC UA in Industrial Automation. In Proceedings of the IEEE Conference on Emerging Technology and Factory Automation, Luxembourg, 8–11 September 2015; pp. 1–6.
18. Lee, B.; Kim, D.-K.; Yang, H.; Oh, S. Model Transformation between OPC UA and UML. *Comput. Stand. Interfaces* **2017**, *50*, 236–250. [[CrossRef](#)]
19. Canedo, A.; Ludwig, H.; Faruque, M. High Communication Throughput and Low Scan Cycle Time with Multi/Many-Core Programmable Logic Controllers. *IEEE Embed. Syst. Lett.* **2014**, *6*, 21–24. [[CrossRef](#)]
20. Canedo, A.; Al-Faruque, M. Towards Parallel Execution of IEC 61131 Industrial Cyber-Physical Systems Applications. In Proceedings of the Design, Automation & Test in Europe Conference, Dresden, Germany, 12–16 March 2012; pp. 554–557.
21. International Electrotechnical Commission. *IEC 61131-3—Part 3: Programming Languages, Ed. 3.0*; International Electrotechnical Commission: Geneva, Switzerland, 2013.
22. Durkop, L.; Imtiaz, J.; Trsek, H.; Wisniewski, L.; Jasperneite, J. Using OPC-UA for the Autoconfiguration of Real-Time Ethernet Systems. In Proceedings of the IEEE Conference on Industrial Informatics, Bochum, Germany, 29–31 July 2013; pp. 248–253.
23. Girbea, A.; Suciuc, C.; Nechifor, S.; Sisak, F. Design and Implementation of a Service-Oriented Architecture for the Optimization of Industrial Applications. *IEEE Trans. Ind. Inform.* **2014**, *10*, 185–196. [[CrossRef](#)]
24. Jirkovsky, V.; Obitko, M.; Kadera, P.; Marik, V. Toward plug&play cyber-physical system components. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2803–2811.
25. Bauer, H.; Hoppner, S.; Iatrou, C.; Charania, Z.; Hartmann, S.; Rehman, S.U.; Dixius, A.; Ellguth, G.; Walter, D.; Uhlig, J.; et al. Hardware Implementation of an OPC UA Server for Industrial Field Devices. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2021**, *29*, 1998–2002. [[CrossRef](#)]
26. Imtiaz, J.; Jasperneite, J. Scalability of OPC-UA Down to the Chip Level Enables Internet of Things. In Proceedings of the IEEE Conference on Industrial Informatics, Bochum, Germany, 29–31 July 2013; pp. 500–505.
27. Gruner, S.; Pfrommer, J.; Palm, F. RESTful Industrial Communication with OPC UA. *IEEE Trans. Ind. Inform.* **2016**, *12*, 1832–1841. [[CrossRef](#)]
28. Bruckner, D.; Stanica, M.-P.; Blair, R.; Schriegel, S.; Kehrer, S.; Seewald, M.; Sauter, T. An Introduction to OPC UA TSN for Industrial Communication Systems. *Proc. IEEE* **2019**, *107*, 1121–1131. [[CrossRef](#)]
29. Li, Y.; Jiang, J.; Lee, C.; Hong, S.H. Practical Implementation of an OPC UA TSN Communication Architecture for a Manufacturing System. *IEEE Access* **2020**, *8*, 200100–200111. [[CrossRef](#)]
30. Panda, S.K.; Majumder, M.; Wisniewski, L.; Jasperneite, J. Real-time industrial communication by using OPC UA field level communication. In Proceedings of the IEEE Conference on Emerging Technology and Factory Automation, Wien, Austria, 8–11 September 2020; pp. 1143–1146.
31. Gogolev, A.; Braun, R.; Bauer, P. TSN traffic shaping for OPC UA field devices. In Proceedings of the IEEE Conference on Industrial Informatics, Helsinki, Finland, 22–25 July 2019; pp. 951–956.
32. Morato, A.; Vitturi, S.; Tramarin, F.; Cenedese, A. Assessment of Different OPC UA Implementations for Industrial IoT-Based Measurement Applications. *IEEE Trans. Instrum. Meas.* **2020**, *70*, 1–11. [[CrossRef](#)]
33. Friedl, S.; Arnim, C.; Lechler, A.; Verl, A. Generation of OPC UA companion specification with Eclipse modeling framework. In Proceedings of the IEEE Conference on Factory Communication Systems, Porto, Portugal, 27–29 April 2020; pp. 1–7.

34. Silva, D.; Carvalho, L.I.; Soares, J.; Sofia, R.C. A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA. *Appl. Sci.* **2021**, *11*, 4879. [[CrossRef](#)]
35. Kim, W.; Sung, M. Poster Abstract: OPC-UA Communication Framework for PLC-Based Industrial IoT Applications. In Proceedings of the IEEE/ACM Conference on Internet of Things Design and Implementation, Pittsburgh, PA, USA, 18–21 April 2017; pp. 327–328.
36. Lee, S.-Y.; Sung, M. RPC-Based OPC-UA Agent for Legacy PLCs. In Proceedings of the IEEE Conference on Emerging Technology and Factory Automation, Stuttgart, Germany, 6–9 September 2022.
37. Wilamowski, B.M.; Irwin, J.D. *Industrial Communication Systems*; CRC Press: Boca Raton, FL, USA, 2017.
38. Open62541. Available online: <http://www.open62541.org> (accessed on 2 July 2022).
39. Müller, M.; Wings, E.; Bergmann, L. Developing open source cyber-physical systems for service-oriented architectures using OPC UA. In Proceedings of the International conference on Industrial Informatics (INDIN), Emden, Germany, 24–26 July 2017; pp. 83–88.
40. Pribiš, R.; Beňo, L.; Drahoš, P. Implementation of Micro embedded OPC Unified Architecture server-client. *IFAC-PapersOnLine* **2019**, *52*, 114–120. [[CrossRef](#)]
41. Unified Automation GmbH. UaExpert. Available online: <https://www.unified-automation.com/products/developmenttools/uaexpert.html> (accessed on 2 July 2022).