An approach to bridge ROS 1 and ROS 2 devices into an OPC UA-based testbed for industry 4.0

Quang-Duy Nguyen, Saadia Dhouib, Yining Huang, Patrick Bellot

An Approach to Bridge ROS 1 and ROS 2 Devices into an OPC UA-based Testbed for Industry 4.0

Quang-Duy NGUYEN*, Saadia DHOUIB*, Yining HUANG*, and Patrick BELLOT†
*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
†LTCI, Télécom Paris, Institut Polytechnique de Paris, F-91120, Palaiseau, France
Email: quang-duy.nguyen@cea.fr, saadia.dhouib@cea.fr, yinning.huang@cea.fr, patrick.bellot@telecom-paris.fr

***Abstract*—ROS 1 and ROS 2 are two widely-used robotic middleware. One of their essential features is to enable two robots with the same middleware, ROS 1 or ROS 2, to connect and collaborate directly. However, two robots running two different middleware can only communicate by additionally using one of the bridge solutions available in the robotic community. It is even more challenging when deploying these robots as part of an OPC UA-based industrial testbed. The first challenge is to network the robots with other OPC UA devices. Second, a testbed environment sometimes requires a robot to join the system rapidly and with minimal configuration for quick experiments. While addressing the above needs, this paper presents an approach to bridge ROS 1 and ROS 2 robots to an OPC UA PubSub network. The approach derives from the actual experiences in developing an OPC UA-based robotic testbed for Industry 4.0 research.**

***Index Terms*—Industry 4.0, OPC UA, ROS 1, ROS 2, PubSub, Bridge, Testbed**

I. Introduction

The fourth industrial revolution, also known as Industry 4.0, is an inevitable technological event of the 21st century. It is about using the most advanced technologies in emerging fields to radically improve industrial systems' communication, automation, and intelligence. Robotics is one of the nine fundamental pillars forming Industry 4.0 [1]. Indeed, robots and automated machinery significantly impact today's industry: they replace human labor in complex, heavy, and dangerous activities, besides adding more value and enhancing performance in manufacturing. In the future, the Industry 4.0 expects autonomous robots to combine with the other eight technology pillars, such as Digital Twin (DT) and the Internet of Things (IoT), to put more intelligence, flexibility, and interoperability into the industry. This vision requires enormous research and innovation efforts; that also involve the need for robotic testbeds for Industry 4.0.

Regarding robotic implementation, ROS 1 and ROS 2 are two widely-used middleware. ROS stands for Robot Operating System. They are open-source and receive great support from the robotic community. Technically, ROS 1 includes a set of libraries to build robot applications and a communication layer that enables the processes generated by running the applications to connect and exchange [2]. The mentioned processes are called ROS 1 nodes. ROS 2 is not another version of ROS 1 but instead another project. Indeed, ROS 2 adopts a new architectural concept with many redesigns from the low-level core modules up to high-level application libraries [3]. On the one hand, the changes enable ROS 2 to approach the industrial level. On the other hand, they lead to an issue: ROS 1 nodes cannot directly connect to ROS 2 nodes since the network protocols of their communication stacks are different. In detail, ROS 1 uses XMLRPC combined with TCP/IP-based and UDP-based message transport, and ROS 2 relies on Data Distributed Service (DDS).

ROS 1 and ROS 2 are two excellent tools for building robotic testbeds. However, a robotic testbed for Industry 4.0 has two further requirements. First, an industrial testbed may follow some industrial standards. Consequently, robots engaging in the system must adopt these standards to collaborate with other industrial devices. Second, a testbed may be a sharing experiment base for multiple Industry 4.0 technologies, use cases, and projects. A robot may rapidly join and quit the testbed for only one quick test. While regarding the above needs, this paper aims to share our approach bridging ROS 1 and ROS 2 robots into an OPC UA-based robotic testbed for Industry 4.0. The core of the testbed is the Open Platform Communication Unified Architecture (OPC UA) standard, currently one of the most high-demanded standards in the industry. Suppose ROS 1 devices constitute a logical network called ROS 1 space, and ROS 2 devices constitute a logical network called ROS 2 space. This approach uses UA bridges to bridge the ROS 1 and ROS 2 spaces to the testbed's OPC UA PubSub network. A UA bridge works as a logical portal with two sides: one side is its ROS space, and the other side is the OPC UA PubSub network. This approach enables robots with the same middleware to collaborate in their space with their default communication method and can still have a route to exchange with other devices through a UA bridge.

The rest of this paper is organized as follows. Section II presents this research's background: the OPC UA standard and its role in our OPC UA-based robotic testbed for Industry 4.0. Section III is a literature review of some existing approaches to bridging ROS 1, ROS 2, and OPC UA devices. Then, Section IV focuses on explaining this paper's approach. Section V proves the concept by presenting a fault-tolerant product assembly line (PAL) case study. Finally, a brief conclusion sums up this paper and outlines some future works.

II. Background

This section presents first some principles of OPC UA, then the OPC UA-based testbed for Industry 4.0 at CEA LIST.

A. OPC UA

OPC UA contains a set of more than 23 specifications defined by the OPC Foundation to build an industrial system with reliability, security, and interoperability [4]. Some industrial partners also contribute to this standard by proposing companion specifications (CS). In a limited scope, this paper focuses only on the specifications related to our contribution and groups them into two principles: (1) resource representation and (2) communication and networking.

The principle of resource representation relies on the OPC UA address space and OPC UA information model. On the one hand, the OPC address space contains a list of OPC UA nodes representing all resources of an industrial system [5]. Each OPC UA node's identification (Id) comprises two information: namespace index and identifier. While a namespace index is a number, an identifier can be numeric, string, globally unique identifier, or binary data blob. The OPC UA information model is the schema modeling all the OPC UA nodes and their relations; in other words, it represents the semantics of the resources [6]. An OPC UA server manages the OPC UA address space and provides access to the resources. OPC UA clients connect to the OPC UA server and can understand the structure of the resources by learning from its OPC UA information model. Figure 1 illustrates an OPC UA-based industrial system. In this figure, an external device uses an OPC UA client application, such as UaExpert[1], to connect to the OPC UA server and access the system's resources through the OPC UA address space. For example, the OPC UA client can read data from a joint of the robotic arm.

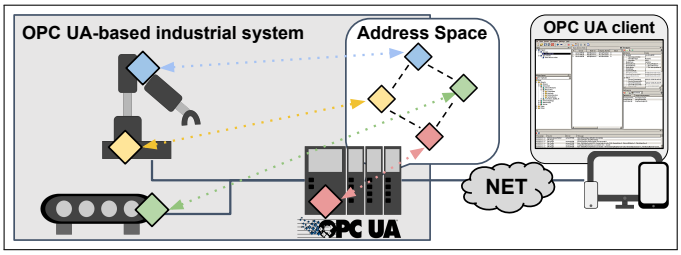

Fig. 1. Example of an OPC UA-based industrial system

Concerning the communication and networking principle, OPC UA proposes two messaging patterns: request-response and publish-subscribe. In the request-response pattern (ResReq), a client sends a request to a server, and the server answers the request with a response. Note that this pattern is also known as the client-server pattern. In the publish-subscribe pattern (PubSub), a subscriber subscribes to a data source once, but they can receive every new message from the publisher [7]. With more detail, OPC UA provides two PubSub communication modes: broker-based and broker-less. Broker-based means there is a broker in the middle that manages topics. A topic is an association between a data source and the information required to create links between the publisher and subscriber sides. Broker-less relies on the multicast mechanism of the UDP/IP stack. In detail, a publisher publishes a message to a multicast address. All subscribers who subscribe to the multicast address can receive the message. To verify if the message contains the expected data, subscribers check the three fields *PublisherId*, *WriterGroupId*, and *DatasetWriterId* in the UA Datagram Protocol (UADP) header of the message. In this paper, these three information fields are called a triple of UADP Ids. At a point, a triple of UADP Ids has the same meaning as a topic in broker-based communication mode.

[1]https://unified-automation.com/products/development-tools/uaexpert.html

B. LocalSEA: An OPC UA-based Testbed for Industry 4.0

LocalSEA is a testbed for Industry 4.0 developed and supervised by CEA List [8]. Its principal mission is to provide a local experimental base for new research, technologies, and use cases dedicated to Industry 4.0. Open Platform Communication Unified Architecture (OPC UA) standard is selected to be the core of LocalSEA since it is a potential candidate to overcome the Information Technology (IT) and Operational Technology (OT) convergence challenge [9]. The challenge reflects the need in Industry 4.0 to reduce the complexity of networking between IT and OT layers and view the resources of both layers in a unified architecture; that involves real-time monitoring, reliable control, and effective management. Figure 2 illustrates LocalSEA separated into the IT and OT layers. At the OT layer, LocalSEA has a robotic cell containing multiple robots, embedded devices, and an OPC UA server. The server follows the OPC UA standard to represent and manage the resources of the robotic cell's devices. Note that, in this paper, a device that follows the OPC UA standard will be called an OPC UA device. All devices connect to the local wireless network and the primary communication method is OPC UA PubSub. At the IT layer, applications run on distributed computers can access the robotic cell through the OPC UA server using OPC UA ResReq. Some developing applications are DT with Asset Administration Shell (AAS), DT with 3D visualization, and a framework for orchestration, choreography, and supervisor monitoring.

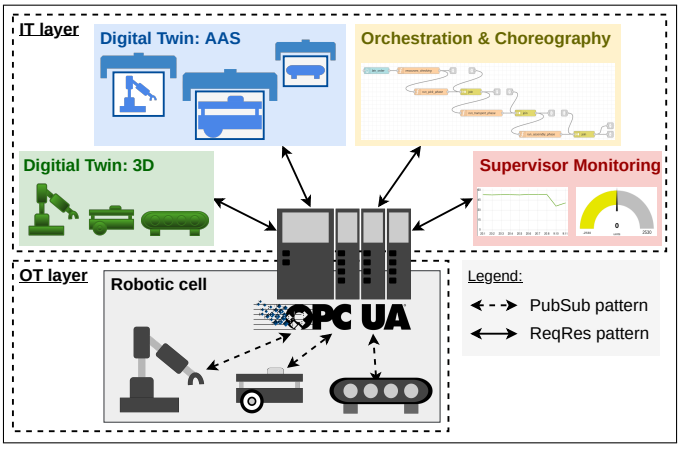

Fig. 2. The LocalSEA testbed and its IT and OT layers

OPC UA's resource representation principle improves the semantics of the whole system. Also, the complexity of networking between IT and OT layers reduces since all communication methods are from the OPC UA standard.

III. Related Work

Five possible directions for ROS 1, ROS 2, and OPC UA devices collaborating are: (1) make all devices use the ROS 1 communication method; (2) make all devices use the ROS 2 communication method; (3) make all devices use an OPC UA communication method; (4) make all devices use another communication method, none of the previous ones; (5) make a bridge for ROS 1 space to join an OPC UA network, make another bridge for ROS 2 space to join the network, and make non-ROS devices use an OPC UA communication method.

The first and second directions have two requirements. The first requirement is to install ROS middleware on the OPC UA server and other devices, to turn them into the hosts of ROS nodes. Note that ROS means ROS 1 or ROS 2, and non-ROS means neither of them. The second requirement is to install bridges that turn all ROS 2 into ROS 1 devices in the first direction, or turn all ROS 1 into ROS 2 devices in the second direction. The advantage of the two directions is that many robots run a ROS middleware by default, and there are many available documents for installing ROS 1, ROS 2, and bridges between them. However, when a new device joins the system, it must be configured to ROS 1 or ROS 2.

The third and fourth directions require all devices to have extra installation and configuration. The third is more effective than the fourth since all devices become OPC UA devices; thus, the OPC UA server can natively understand arrived messages without conversion. One example of the fourth direction is the *rosbridge* middleware that proposes a web service abstraction layer for ROS 1 [10]. As the authors claim, this middleware can be extensible for other devices, so it probably works with ROS 2 and other non-ROS devices.

The final direction is relevant to testbed environments. Indeed, ROS devices can join and work as default without particular configuration since a bridge in their ROS space can help them route data to the outside OPC UA network. The first requirement is to design the two bridges so they can convert and forward messages. The second requirement is to map data sources in ROS spaces with OPC UA nodes in the address space. Note that the second requirement is necessary to archive the OPC UA resource representation principle. Concerning the first requirement, Tripath et al. successfully bridge ROS 1 devices with OPC UA devices using the Eclipse Arrowhead framework working as a bridge [11]. Ioana et al. introduce another bridge solution that defines multiple functional devices for the exchange between DDS and OPC UA sides [12]. However, both provide no mechanism to map ROS spaces' data sources with OPC UA nodes.

IV. UA Bridges for ROS 1 and ROS 2

The approach proposed by this paper follows the fifth direction in Section III. Logically, a UA bridge is a portal with two interfaces: one interface for communication with devices in its ROS space and one OPC UA PubSub interface for communication with other devices using the OPC UA PubSub communication method. Technically, a UA bridge is a ROS 1 node when it provides a ROS 1 interface. It will be called ROS 1 UA bridge. Likewise, a UA bridge is a ROS 2 node when it provides a ROS 2 interface. It will be called ROS 2 UA bridge. Explicitly, a UA bridge is also a node in the OPC UA PubSub network. From now on, the network that OPC UA PubSub nodes communicate will be called an OPC UA PubSub space. At each interface, a UA bridge plays the role of both publisher and subscriber, and it respects the communication method of the corresponding space. Figure 3 illustrates an architecture with three different spaces, including a ROS 1 space, a ROS 2 space, and an OPC UA PubSub space.

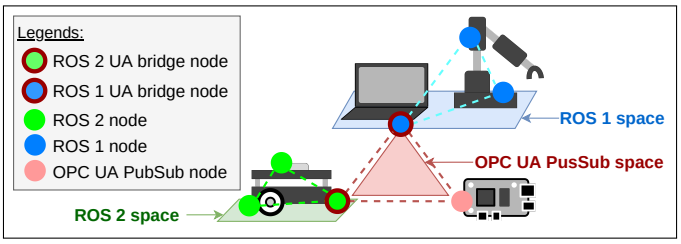

Fig. 3. The architecture of ROS and ROS 2 with UA bridges

UA bridge has two jobs. First, it receives OPC UA PubSub messages at the OPC UA PubSub interface, converts them into ROS messages, and publishes them in the ROS space through the ROS interface. Second, it reverses the processes of the first job when receiving ROS messages at the ROS interface.

Two following subsections present the two main steps to implementing UA bridges. It is worth mentioning that, since UA bridges are components of an OPC UA-based system, it is necessary to have a preamble step to prepare two fundamental elements: the OPC UA address space and the OPC UA PubSub communication method. However, these works are out of the scope of this paper.

A. Mapping OPC UA Nodes to ROS space's Data Sources

In a ROS space, a topic can represent a data source. Thus, the mapping from OPC UA nodes to ROS space's data source is similar to the mapping from OPC UA node Ids to ROS space's topics. This mapping comprises two partial mappings: (1) mapping OPC UA node Id with OPC UA PubSub topic, and (2) mapping OPC UA PubSub topic with ROS space's topic. Note that an OPC UA PubSub topic in the broker-less is a triple of UADP Ids. This paper proposes a unified format, called UaRosSpace, to design ROS space's topics: **`ua_<ROS space ID>_<Node Id name space index>_<Node Id identifier>`**. In which, *ua* is an unchangeable element identifying the format, <*ROS space Id*> provides the ROS space of the mapping data source, <*Node Id namespace index*> provides the mapping OPC UA node's namespace index, and <*Node Id identifier*> provides the mapping OPC UA node's identifier. The *ROS space Id* of a data source from ROS 1's space is always 1. The *ROS space Id* of a data source from the OPC UA PubSub's space is always 0. The *ROS space Id* of a data source from ROS 2's space can be from 2 to 232, corresponding to the ROS 2 domain ID. Note that with this configuration, a ROS 2 device must always have a domain ID superior to 1. The *Node Id*

namespace index value and the *Node Id identifier* value of a data source can be found on the OPC UA address space by querying the corresponding OPC UA node Id.

The mapping from OPC UA node Ids to OPC UA PubSub topics occurs at the OPC UA server. In the OPC UA PubSub broker-based mode, the PubSub topics also use the UaRosSpace format. In the OPC UA PubSub broker-less mode, the *PublisherId* value is the value of ROS space, the *WriterGroupId* value is designed to the mapping OPC UA node Id's namespace index, and the *DatasetWriterId* value is designed to the mapping OPC UA node Id's identifier.

The mapping from OPC UA PubSub topics to ROS space's topics occurs at a UA bridge. In the PubSub broker-based mode, since these two topics use the same format, the UA bridge can process exchange between two OPC UA PubSub and ROS interfaces without trouble. In the PubSub broker-less mode, *PublisherId* maps to *ROS space*, *WriterGroupId* maps to *Node Id namespace index*, and *DatasetWriterId* maps to *Node Id identifier*.

B. Designing a UA bridge

Since a UA bridge is a ROS node, it can profit from the publish and subscribe mechanism in the ROS space. However, developers must write codes following one OPC UA PubSub profile[2] for communication in the OPC UA PubSub space.

In this approach, a UA bridge uses the same topic for both actions of publishing and subscribing. Thus, we propose a loop-prevention mechanism implied in the algorithm of the UA bridge program as in Figure 4. It relies on the origin and destination of a message. From a UA bridge's viewpoint, when the message's origin is from its ROS space, the message's destination must be in an OPC UA PubSub space or another ROS space. In reverse, when the message's origin is from the OPC UA PubSub space or another ROS space, the message's destination must be in its ROS space. While the topic's ROS space Id can represent the message's origin information, the UA bridge's two interfaces can be used to detect the message's expected destination. In detail, when a UA bridge receives a message at its OPC UA PubSub interface, the message's expected destination should be in ROS space. Continuously, if the message's origin contains the ROS space Id that equals the UA bridge's ROS space Id, the origin and destination of the message are the same. Thus, it is a looped message, and the UA bridge drops it. We can apply the same logic to the remaining case.

The result of a UA bridge implementation is a ROS program. In good practice, a UA bridge program should run on a stable device. Thus, the UA bridge is always available for other devices in the same ROS space.

V. Case Study: Fault-Tolerant PAL System

PAL is "a manufacturing process where the bill-of-material parts and components are attached one-by-one to a unit in a sequential way by a series of workers to create a finished product" [13]. The PAL system deployed at LocalSEA imitates the box manufacturing process, in which the finished product is a box with its cover. In this PAL, there are two workers. First is Niryo Ned, a 6-axis robotic arm that picks a cover and places it into a carrier. This procedure is called a pick-and-place cycle. Second is a human that assembles a box with the cover received from the carrier to create a finished product. The primary carrier is a conveyor belt. To ensure the production line works even when the conveyor belt accidentally stops working, a TurtleBot3 Waffle Pi plays the role of a substitute carrier. Technically, Niryo Ned runs ROS 1 Melodic, and TurtleBot3 runs ROS 2 Foxy. The conveyor belt connects to the controller inside the Niryo Ned through a wired connection. Also, a Raspberry Pi 3B plus plays as an OPC UA server and a gateway at the same time. As a gateway, on the one hand, it creates a hotspot local Wi-Fi network for LocalSEA and, on the other hand, connects to the Internet. The two robots join the LocalSEA Wi-Fi. Supervisors can monitor the PAL from a distance with an Internet connection. Figure 5 illustrates the fault-tolerant PAL system.

Two robots with two different programs can choreograph due to the states of two elements: park and bucket. The park represents a position where TurtleBot3 receives a new cover. The bucket is a container that holds the cover on top of TurtleBot3. They both have two states: busy or free. When TurtleBot3 is at the park, the park is busy; when a cover is in the bucket, the bucket is busy. If the park is busy and the bucket is free, Niryo Ned can place a cover in the bucket. If the park is busy and the bucket is busy, TurtleBot3 moves to the human worker. Otherwise, the two robots wait.

Following the approach defined in Section IV, we implement one ROS 1 UA bridge and one ROS 2 UA bridge. In this case study, the ROS 1 space contains ROS 1 nodes hosted in Niryo Ned, and the ROS 2 space contains ROS 2 nodes hosted in TurtleBot3. TurtleBot3 has a domain ID equal to 2. In the preamble step, our developers reuse the OPC UA information model of another PAL monitoring case study as presented in [14], and the minimal configuration OPC UA PubSub Broker-less as presented in [15]. In the mapping step, besides mapping the variables representing the status of bucket and park to the two topics in the ROS 2 space, our developers also map other variables required for the monitoring process, to their related topics. Some variables are the status of the conveyor belt, the number of covers put on the conveyor belt, the number of covers put on the TurtleBot3, and the number of covers in the storage. Figure 6 illustrates the mapping of the case study. In the bridge designing step, our developers put the hard-coded deployment of the OPC UA PubSub UDP UADP profile into the two UA bridges for the publish and subscribe at their OPC UA PubSub interface. The final result is two UA bridge programs for two ROS spaces.

Running the case study includes two steps. The first step is to run the two UA bridge programs. The second step is to run the business programs of Niryo Ned and TurtleBot3.

Our developers propose a testing scenario for the case study with two phases. The default phase is when Niryo Ned works

[2]https://profiles.opcfoundation.org/profilefolder/320

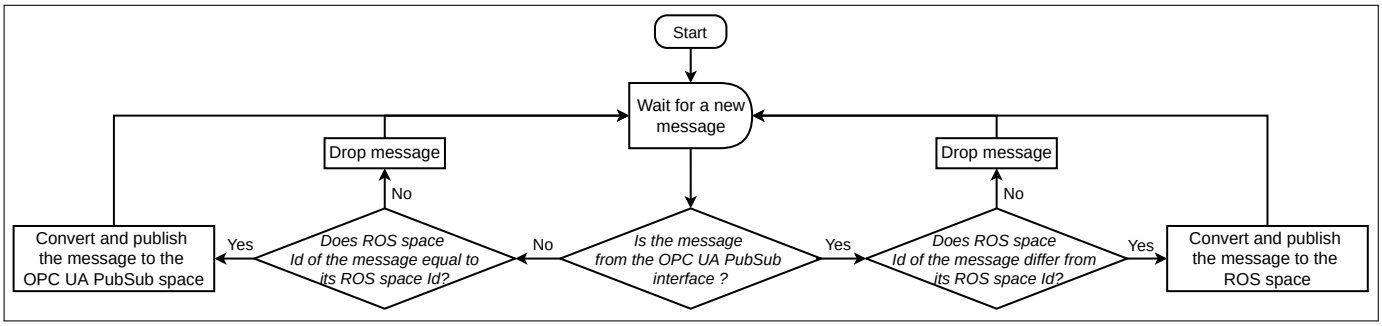

Fig. 4. Flow chart of the algorithm of the UA bridge's program

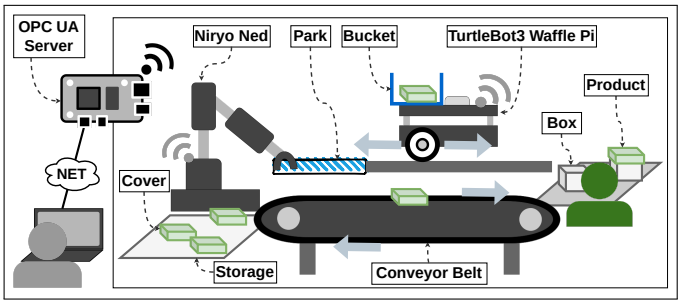

Fig. 5. The architecture of the fault-tolerant PAL case study of LocalSEA

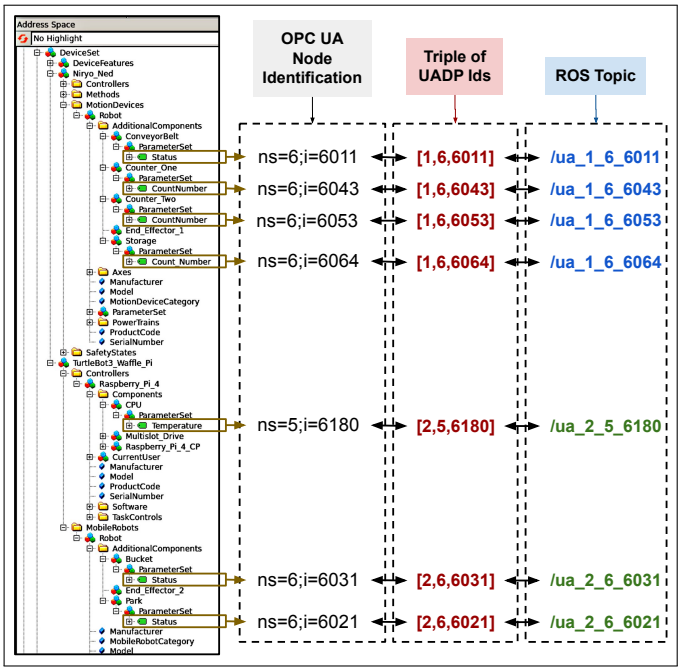

Fig. 6. Mapping from OPC UA node Ids to ROS 1 and ROS 2 topics

with the conveyor belt. The fault-tolerant phase is when the conveyor belt is off, then Niryo Ned works with TurtleBot3. The testing scenario requires two other computers. The first computer opens our supervisor monitoring application to observe the variables of the OPC UA address space. The second computer joins the ROS 1 space to turn off the conveyor belt when necessary. Figure 7 shows a testing sample's events. In this testing sample, we put four covers on the storage, so there are four pick-and-place cycles. The conveyor belt is turned off at the 25th second.

There are four remarks from the recording events of the testing scenario. First, the duration when Niryo Ned does a pick-and-place cycle is around 5 or 7 seconds, depending on the position to pick and one to place. However, the last pick-and-place cycle in the Figure is about 16 seconds. The reason is that TurtleBot3 leaves the park before Niryo Ned can finish the cycle. Thus Niryo Ned needs to wait until the 76th second. Second, both Niryo Ned and the OPC UA server can detect the event that the conveyor belt is off. However, Niryo Ned recognizes the situation about two seconds before the OPC UA server does. Since the conveyor belt connects directly to Niryo Ned by cable, Niryo Ned can recognize the situation instantly. The OPC UA server is in the OPC UA PubSub space, so it has an extra delay of two seconds. This extra delay is a sum of three sub-durations: the data transport time in ROS 1 space (tTR1), the processing time at the ROS 1 UA bridge (tPB1), and (3) the data transport time in the OPC UA PubSub network (tTR0). Third, another method to detect the delay between the ROS 1 space and the OPC UA PubSub space is to calculate the duration starting when Niryo Ned places a cover on a carrier and ending when the OPC UA server detects a counter's value change. On average, this duration is about four seconds. It is a combination of four sub-durations: the time for Niryo Ned to complete its action before publishing a notification (tAR1), tTR1, tPB1, and tTR0. Fourth, the duration starting when Niryo Ned places a cover on TurtleBot3 and ending when this mobile robot moves forwards is about ten seconds on average. This duration represents the delay in sending data from the ROS 1 to the ROS 2 space. It is the combination of seven sub-durations: tAR1, tTR1, tPB1, tTR0, the processing time at the ROS 2 UA bridge (tPB2), the data transport time in the ROS 2 space (tTR2), and the time for TurtleBot3 to load the action (tAR2). Other details, such as the delays between the storage counter and the two carrier counters, relate to the Niryo Ned business program.

From the above observation, we can conclude that the two UA bridges work. The delays are quite high since the codes of the programs are non-optimized. For example, the tAR1 is about two seconds, and the tPB1 is another two seconds.

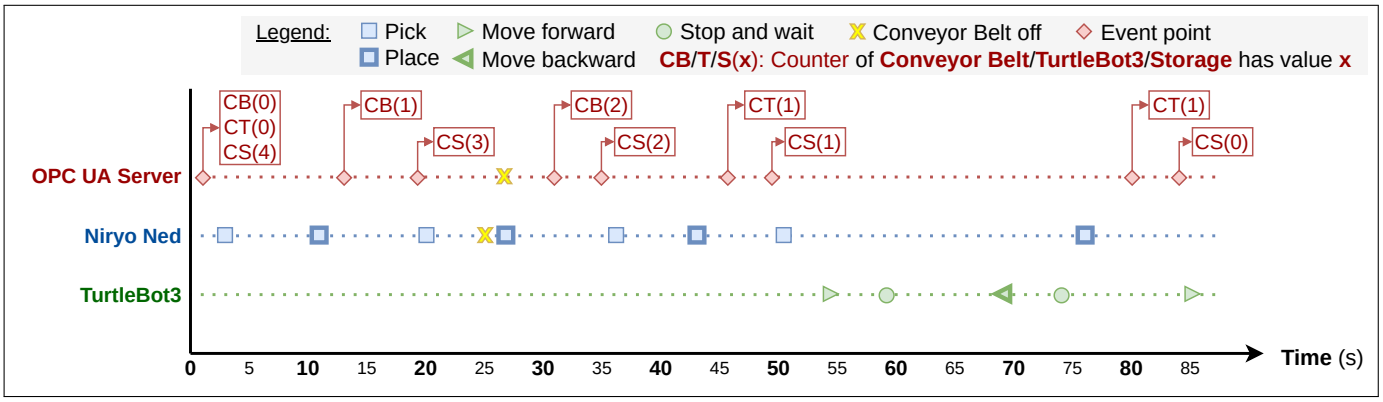

Fig. 7. Recording events of a testing sample of the fault-tolerant PAL case study

VI. Conclusion and Future Works

This paper presents an approach to making UA bridges that enable ROS devices with the same middleware and configuration to join OPC UA-based robotic testbeds for Industry 4.0 through one logical portal. Compared to other existing approaches, this approach is better in three points. First, it can profit the most from the advances of the OPC UA standard. Second, it supports ROS 1 and ROS 2 in the same scenario. Third, new ROS devices can join the system without configuration when a relevant UA bridge is running. The fault-tolerant PAL case study, also presented in this paper, proves that this approach is realizable. However, the case study is still simple, and the implementation is not optimized enough to show all difficulties and potential of the approach.

This paper has three points to discuss further. First, in our testbed, even though the OPC UA standard already shortens the gap between IT and OT layers: the resource representation principle to improve semantic interoperability and the communication and networking principle to reduce the networking complexity, there are not yet direct connections between devices of the two layers. Considering this, we plan to deploy the OPC UA PubSub MQTT JSON profile that enables the OT layers' devices to serve data to IoT applications without passing the OPC UA server.

Second, our UA bridge approach is for general purpose in developing testbeds for Industry 4.0. It is worth noting that the path from a data source to a UA bridge, then to the OPC UA server, is sometimes longer than the direct path from the data source to the server. For some specific applications that require the strict constraint on delay time, such as the synchronization between a physical robot and its DT, developers may need to deploy more than one UA bridge in a ROS space to create a shorter path for data transferring.

Third, manual mapping can be a heavy job, especially when there are too many OPC UA nodes to map. Our future goal is to develop a plugin in Papyrus[3] that facilitates the mapping design with SysML. The tool should support generating UA bridges automatically from the SysML models.

[3]https://www.eclipse.org/papyrus/

Acknowledgment

This work is partially funded by DIMOFAC, an EU Horizon 2020 research and innovation program under grant agreement No 870092.

References

[1] F. Yang and S. Gu, "Industry 4.0, a revolution that requires technology and national strategies," *Complex & Intelligent Systems*, vol. 7, no. 3, pp. 1311–1325, Jun. 2021.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, vol. 3, 2009, p. 6.

[3] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, May 2022.

[4] OPC Foundation, "OPC Unified Architecture - Part 1: Overview and Concepts," Industry Standard Specification OPC 10000-1, 2017.

[5] OPC Foundation, "OPC Unified Architecture - Part 3: Address Space Model," Industry Standard Specification OPC 10000-3, 2017.

[6] OPC Foundation, "OPC Unified Architecture - Part 5: Information Model," Industry Standard Specification OPC 10000-5, 2017.

[7] OPC Foundation, "OPC Unified Architecture - Part 14: PubSub," Industry Standard Specification OPC 10000-14, 2018.

[8] Q.-D. Nguyen, F. Rekik, Y. Huang, and S. Dhouib, "Early lessons learned from the development of a local OPC UA-based robotic testbed for research," in *2022 IEEE 31st International Symposium on Industrial Electronics*, Anchorage, United States, Jun. 2022, pp. 1–4.

[9] H. M. Park and J. Wook Jeon, "OPC UA based Universal Edge Gateway for Legacy Equipment," in *2019 IEEE 17th International Conference on Industrial Informatics*, vol. 1, Jul. 2019, pp. 1002–1007.

[10] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: ROS for Non-ROS Users," in *Robotics Research*. Cham: Springer International Publishing, 2017, vol. 100, pp. 493–504.

[11] A. Tripathy, J. van Deventer, C. Paniagua, and J. Delsing, "Interoperability Between ROS and OPC UA: A Local Cloud-Based Approach," in *2022 5th International Conference on Industrial Cyber-Physical Systems*, Coventry, United Kingdom, May 2022, pp. 1–5.

[12] A. Ioana and A. Korodi, "DDS and OPC UA Protocol Coexistence Solution in Real-Time and Industry 4.0 Context Using Non-Ideal Infrastructure," *Sensors*, vol. 21, no. 22, p. 7760, Nov. 2021.

[13] N. T. Thomopoulos, *Assembly Line Planning and Control*. Cham: Springer International Publishing, 2014.

[14] Q.-D. Nguyen, S. Dhouib, K. Suri, and F. Rekik, "From requirement specification to OPC UA information model design: A product assembly line monitoring case study," in *2022 IEEE 20th International Conference on Industrial Informatics*, Perth, Australia, Jul. 2022, pp. 1–6.

[15] Q.-D. Nguyen, P. Bellot, and P.-Y. Petton, "An OPC UA PubSub Implementation Approach for Memory-Constrained Sensor Devices," in *2022 IEEE 31st International Symposium on Industrial Electronics*, Anchorage, United States, Jun. 2022, pp. 1–7.