Bridging Automation and Robotics: an Interprocess Communication between IEC 61131-3 and ROS

Tiago Pinto, Rafael Arrais, and Germano Veiga
INESC TEC - INESC Technology and Science, FEUP campus, Rua Dr. Roberto Frias, Portugal
Email: {tiago.f.pinto, rafael.l.arrais, germano.veiga}@inesctec.pt

***Abstract*—The contemporary adoption of Cyber-Physical Systems and improvements in robotic applications in industrial scenarios demands for horizontal integration mechanisms with already existing automation equipment, controlled by PLCs. This paper aims to shorten the gap between the automation and robotics domain, by proposing an Interprocess Communication method to establish interoperability between robotic systems and automation equipment in a reliable and straightforward manner. In particular, this paper introduces a novel approach for linking ROS and IEC 61131-3 by way of shared memory interfaces, enabling and promoting their interactions. Moreover, this paper addresses the applied synchronization mechanism for handling concurrent accesses to the shared memory location, explores data type mapping between ROS and IEC 61131-3, and identifies some practical industrial applications.**

I. Introduction

In recent years, an ever-increasing interest for robotic applications in industrial contexts accelerated the development of more flexible and adaptive Cyber-Physical Systems (CPS) [1]. With the industrial emergence of such systems, great challenges and opportunities appear in terms of their vertical integration with Manufacturing Execution System (MES) and Enterprise Information Systems (EIS) [2], and horizontal integration with a vast array of automation equipment.

The current manufacturing paradigm enhances the obligation for companies to respond quickly to customer needs and market changes, increasing demand diversity, products with shorter life cycles, and low volumes per order, while still managing costs and maintaining, if not improving, quality and reliability. In order to cope with these requirements, companies need responsive robotics and automation solutions, which must not only have effective communication methods, but also be quickly re-programmable and re-allocable to different manufacturing tasks. Moreover, the advent of flexible industrial robotic solutions, that vastly diverge from the classical fenced approach, increase the necessity of methods for expediting the intercommunication with automation equipment, as to empower the Plug'n'Produce concept [3].

In contrast with current needs, the interoperability between robotics and automation equipment is still an open-ended problem. In fact, even on the automation scope, there are still many challenges on integrating industrial hardware from different vendors, due to proprietary protocols, communication interfaces, and buses. Most frequently than desired, in order to not expose the system to the risk of incompatibility, integrators opt for equipment from the same vendor, due to the usual inconvenience of combining hardware from different suppliers.

Initiatives such as OPC-Unified Architecture (OPC-UA) [4] are being proposed with the goal of improving interoperability amongst automation equipment and support connectivity with MES, EIS and Human-Machine Interfaces (HMIs). Although promising, their reach has not yet fully expanded to the robotics community and to practical horizontal integration efforts.

As a consequence, connectivity between robots and industrial hardware is often limited to communications through fieldbuses. Indeed, the Robot Operating System (ROS) community and initiatives such as ROS-Industrial currently provide support for interoperability with some fieldbuses, such as Modbus, CANopen, EtherNET/IP, and EtherCAT. Although possible, the usage of fieldbus-based communication in ROS can still be a cumbersome task, on account of the inherent development overhead imposed by the aforementioned protocols. Furthermore, this support represents a significant development effort for the robotics community, and practical results can easily become outdated or originate unstable interfaces.

To overcome these limitations, this paper proposes a communication method between automation equipment and robotic systems that facilitates the establishment of an effective, bidirectional, reliable, and structured interoperability between robotic systems and software-based Programmable Logic Controllers (PLCs). Specifically, this paper explores a method for establishing bidirectional communication between ROS and CODESYS, a softPLC based on the IEC 61131-3 standard running on an embedded system. In the proposed approach, an Interprocess Communication (IPC) method, based on a publisher-subscriber messaging pattern is employed. The achieved implementation explores the existing style for transmitting information between ROS nodes and extend it to support IPC through a shared memory methodology. As such, the proposed application allows a ROS node to publish a message to a topic and have its content written on a shared memory location, that can then be read by CODESYS. Inversely, it also makes possible for CODESYS to write data on a shared memory location, that is then propagated through a ROS topic for the ROS system. This paper also addresses the measures adopted to handle process synchronization in the perspective of concurrent access to a shared memory location, using a semaphores-based approach.

Thus, the proposed approach is able to establish interoperability between robotic systems and automation equipment. Furthermore, it allows ROS systems to use CODESYS as a communication bridge, taking advantage of its natural ability to establish effortless communication with fieldbuses and even to protocols not yet supported by ROS, such as OPC-UA. Also, it enables and promotes interaction between the robotics and the automation domains, making the best use of both worlds. Examples of possible applications include the possibility for a ROS system to take advantage of tools such as motor drivers that are known for being stable in CODESYS, or the possibility to re-program robot tasks using IEC 61131-3, which might be appealing for automation technicians who do not possess the required skills to delve into complicated robotic systems and architectures.

The remainder of this paper is organized as follows: Section II describes related work, Section III details the developed approach, Section IV explores potential applications of the proposed approach, and, finally, Section V draws the conclusions.

II. Related Work

Developing a completely new robotic application can be discouraging if the developer has to start programming from driver-level up to the high-level tasking. Besides being time consuming, this whole spectrum of knowledge is usually beyond the capabilities of researchers and robotic enthusiasts. Given these problems, there is a need for a unified software architecture that simplifies the development of new robotic applications. Due to its modularity, helpful tools and active community, in the last years, ROS has become a de facto framework for robot development, and is expanding its industrial presence over time.

A typical ROS architecture is modular, in the sense that each component of the system can be decomposed in an arrangement of nodes, each responsible for a particular feature in the global robotic system. Therefore, ROS systems are typically composed by multiple nodes that are interconnected by communication mechanisms in a distributed fashion [5]. Even though ROS supports other means of internal communication, usually it is accomplished through a publisher-subscriber pattern [6]. By using this pattern, messages are published on topics which have a specific name in the ROS network. Other nodes can access data published on these topics, by subscribing them by their names. Each topic has a message type, which determines the type of data to be transmitted. Some of these types are already predefined as standard messages, nonetheless it allows users to create their own custom messages.

In what concerns to industrial automation, one of the most important standards is the IEC 61131 which defines all the aspects of PLC development. The third part of this standard, IEC 61131-3, deals with the programming aspect of PLCs and defines the programming model, composed of three Program Vaporisation Units (POUs) and five programming languages [7]. This standard has brought unification among PLC manufacturers, allowing more compatibility and code portability between devices, taking advantage of the modern concepts of software technology. Even though PLC vendors have generally accepted this standard, they are adopting it at their own pace and with some modifications, which results in programs created for different PLC brands not being fully compatible. A broader acceptance came from softPLC vendors, since the recentness of the technology allowed for an outright disregard of maintaining backward compatibility with existing PLC programs [8].

Currently sofPLCs are replacing traditional PLCs in many fields. Its hardware agnostic framework allows costumers to choose the most suitable device for its application. SoftPLCs can be implemented in a vast number of devices, ranging from smaller embedded devices to robust industrial computers. Usually, devices running softPLCs are provided with powerful and flexible communication systems and its instructions can be easily updated [9].

One of the most known names in sofPLCs market is CODESYS, from the German company 3S-Smart Software Solutions GmbH. Besides its softPLC that runs on Windows and Linux, CODESYS also offers an Integrated Development Environment (IDE) for PLC programming that implements to a great extent the IEC 61131-3 standard and allows users to program not only the CODESYS softPLC, but also PLCs from multiple vendors. CODESYS is hardware-independent, nonetheless, there are more than 250 hardware manufacturers who have chosen to use CODESYS as a development tool for their equipment [10]. Besides being IEC 61131-3 compliant, CODESYS supports a vast variety of fieldbuses such as PROFIBUS, CANopen, EtherCAT, PROFINET, EtherNet/IP and MODBUS.

Traditionally, these fieldbuses are the main way of bridging PLCs or softPLC applications to the non-automation domain. Nonetheless, softPLCs are applications running on PC-based devices which can support other applications running in parallel. Therefore, the communication with non-automation software by softPLCs can also be achieved through IPC methods. The advantage of using IPC instead of fieldbuses is that by using the later, data must be routed through all the network layers inherent to network-based communication, introducing unnecessary overheads. These overheads can be avoided by implementing IPC methods which usually outperform network-based ones.

The fieldbus approach can be observed in a multitude of robotic applications, where there is an implicit need to interconnect the automation and robotic disciplines. One example is the *Rollin'Justin* robot, a mobile platform coupled with a humanoid upper body [11]. In this robot, a TwinCAT sofPLC from BECKHOFF was installed and used as a data logger, communicating with the mobile platform and a real-time control computer using EtherCAT and communicating with the robot arms and torso using SERCOS. Although not frequent, similar approaches as the one proposed by this paper were already pursued. An example is the control of a parallel cable robot used for large scale assembly of solar power plants [12]. In this case the control of the robot was implemented on a PC-based real-time operating system, where the softPLC communicates with a numeric control kernel through shared memory.

Both of the aforementioned practical approaches were effective in solving their own problems, but the developed solutions could not be easily applied in different systems with different configurations, since they were developed in an ad hoc fashion, i.e., programmed to interchange problem-specific variables. Contrary to the problem-oriented approach described in the previous examples, this paper proposes to maximize the re-usability of the bridge between ROS and CODESYS by sharing variables using an IPC method in a dynamic way.

Additionally, a communication system between ROS and an open source softPLC, BEREMIZ, was proposed in [13]. In this work, the authors achieved successful interoperability by mapping variables between a ROS system and a BEREMIZ softPLC application. Adapting this approach to CODESYS is not straightforward, since, unlike BEREMIZ, CODESYS is not open source. As such, one is bounded to implement this communication interface using libraries provided by CODESYS.

An IPC method that CODESYS supports by default is the POSIX shared memory. This library provides functions for accessing a memory area that can be shared by multiple processes. In order to avoid data corruption, it is necessary to assure that only a single process accesses the shared memory at a given time [14]. This can be achieved by implementing a synchronizing mechanism. In the CODESYS side of the equation, there is a library that implements POSIX semaphores, which can guarantee the shared memory access synchronization. Both POSIX shared memory and POSIX semaphores can also be implemented on ROS, making this communication method a viable solution to accomplish data transmission between a ROS system and a CODESYS softPLC running on the same device.

III. IPC BETWEEN ROS AND CODESYS

In order to achieve the proposed IPC communication between ROS and CODESYS, two independent interfaces must be implemented: one on ROS side and other on CODESYS side. These interfaces are mutually responsible for capturing and writing data to the shared memory location, and consequently, reading and transmitting data located on the shared memory location to processes that will use it. Thus, this method is bidirectional, in the sense that both CODESYS and ROS must be able to transfer and access data located on the shared memory location.

The most primitive approach to implement this methodology would be individually mapping each variable to share on both sides, i.e., on ROS nodes and on CODESYS programs. This approach would grant read and write access to a common variable mapped on the ROS and CODESYS systems. Although the complexity of this implementation is not daunting, for each variable to be shared by both processes there is an implicit need to repeat the shared memory implementation

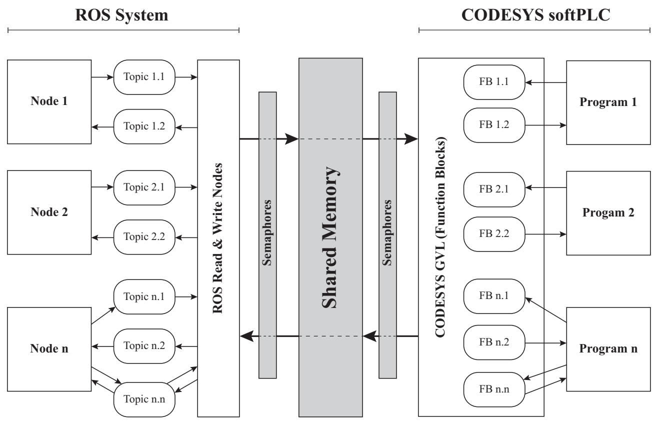

Fig. 1. High-level overview of the proposed ROS-CODESYS interprocess communication interface architecture. In the proposed architecture, a synchronization mechanism based on semaphores handles concurrent accesses to the shared memory location. Figures 2 and 4 will present in more detail the proposed ROS and CODESYS interfaces, respectively.

code. Besides not being feasible and scalable in large systems with multiple variables, this basic approach could be error-prone and problematic when applied to complex ROS and CODESYS systems.

To tackle these limitations, this paper proposes a more systematic approach to allow an intuitive and fast implementation of a shared memory interface between ROS and CODESYS systems. The proposed approach aims to be easily adaptable to fit already existing robotic and industrial automation systems. Moreover, unlike the aforementioned procedure, the proposed approach allows data transferring through shared memory in a dynamic way, taking advantage of ROS and CODESYS features. The proposed ROS and CODESYS interfaces will be described in Sections III-A and III-B, respectively. Moreover, Section III-C dissects how the proposed approach implicitly addresses the synchronization mechanism that handle concurrent accesses to the shared memory location, using a semaphore-based approach. Finally, Section III-D will portray how data types between ROS and IEC61131-3 were mapped and will delve into user-defined custom message types.

A. ROS Interface

One of the main features of ROS is its implementation of the publisher-subscriber pattern that allows communication between nodes. As such, there are obvious advantages in connecting ROS topics to the proposed shared memory methodology: on one hand, developers could create new ROS packages with capabilities to connect with CODESYS using the topic-based interfaces that they are already used to work with. On the other hand, adapting existing ROS packages to work with CODESYS would be an easy task, since existing packages are already compliant with the publisher-subscriber ideology. An additional benefit would be the fact that ROS nodes could communicate with CODESYS programs without being aware that it is out of the ROS system. Furthermore, since the publisher-subscriber pattern has a many-to-many data model, the data shared with CODESYS could be read and modified by multiple ROS nodes.

Therefore, a shared memory interface that is capable of publishing and subscribing the appropriate topics was conceptualized. The proposed interface is able to read messages published by the ROS system and write its content to the shared memory location, that is then accessible by the CODESYS interface. Similarly, on the other way around, it is also able to read what the CODESYS system had written in the shared memory location and publish it to a topic, making the data available for being subscribed by any ROS node. A representation of the adopted architecture is depicted in Figure 1. In it, it is possible to observe that by using an interface interconnected dynamically with ROS topics it is possible to have multiple nodes sharing data with CODESYS programs.

In practical terms, the implementation on the ROS side was developed around a C++ template class. Each instance of this class corresponds to a subclass associated with a ROS message type and its relative CODESYS compatible structure. This data type mapping will be further addressed in Section III-D.

As such, whenever a developer needs to transmit data between a ROS topic and the shared memory, an instance of a subclass of the template needs to be created, matching the message type of the corresponding topic. This subclass takes the name of the topic as an input to its constructor method, and uses this information not only to identify the topic, but also to identify the named shared memory location and its corresponding semaphore. Internally, within this subclass, in order to read and publish the content of the named shared memory location, a method responsible for triggering all the pipeline of operations needed to route the data from the shared memory to the desired topic is called. This pipeline includes operations such as reading data from the shared memory, mapping it to the structure which is equivalent to the respective ROS message, convert the structure into a real ROS message, and, finally, publish it on the topic. Similarly, to write data to the shared memory, an internal method triggers the pipeline to route the data from the ROS topic to the named shared memory location. This pipeline encompasses all the operations that are necessary to route the data published on a given ROS topic, transform it into a CODESYS compatible data structure, and then map this structure to the shared memory named after the subscribed topic.

On these operations it is crucial to synchronize the access to the shared memory in order to maintain the integrity of the data. Therefore, all interactions with the shared memory are preceded by a semaphore lock and followed by a semaphore release, in order to avoid data corruption. When the semaphore is locked, if the CODESYS application tries to access the shared memory, it will have to wait until the ROS node finishes the operation and releases the semaphore. Section III-C will present further details on this synchronization mechanism.

Even though the proposed mechanism can be instantiated in any ROS node, it was idealized to be used on isolated nodes that are only responsible for handling this IPC method. This way, other nodes on the system can be completely unaware that the information that is subscribed or published to a topic is being shared with CODESYS. A representation of the idealized communication node is depicted on Figure 2, where on a single node there are multiple instances of the interface objects, corresponding to various topics. Each object has a *read* or *write* method that can be called to execute the desired goal.

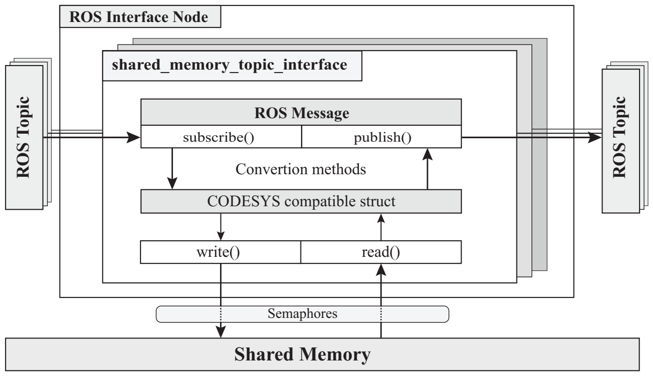

Fig. 2. Conceptual overview on the proposed ROS shared memory interface.

B. CODESYS Interface

On the industrial automation side, since the IEC 61131-3 standard does not define any publisher-subscriber messaging pattern, it was necessary to implement a system that mimics this pattern, using tools available on CODESYS. In order to make the interchanged data available to all programs on the CODESYS project, it was decided to use Global Variable Lists (GVLs). GVLs are components of the IEC 61131-3 standard that allow variables to be declared and made available to all POUs of a project.

Following the pattern of the ROS implementation, two Function Blocks (FBs) were assigned to handle each ROS message type. On this idealization, one FB is responsible for reading data from the shared memory (the *subscriber*), while the other is responsible for writing data to the shared memory (the *publisher*). Having the *publisher* and the *subscriber* in separate FBs simplifies the implementation of ROS communication using the graphical programming languages of the IEC 61131-3 standard. In Figure 3, there is a representation of the FBs used for reading and writing an equivalent data structure of a ROS *Pose* message.

In Figure 4, the interaction between CODESYS elements can be analyzed. The FBs responsible for reading and writing data to ROS topics are declared on a GVL. The data read by the reading method of the FB will be mapped to a data structure that will be available to be used by CODESYS programs and FBs under the same project. Similarly, data from programs and FBs in the CODESYS project can be used as input to the analogous writing FBs.

In what regards implementation, since the CODESYS interface is going to read and write data to the shared memory which will, in turn, be related to the corresponding ROS topics,

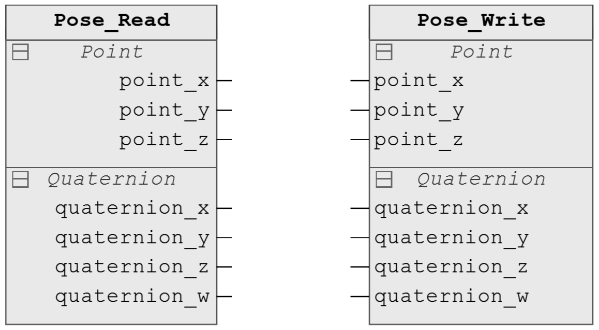

Fig. 3. ROS *Pose* message represented as graphical FBs on IEC 61131-3. On the left side, the FB used for reading the ROS *Pose* message from the shared memory is shown, while on the right side, the FB employed for writing the referred data to the shared memory is depicted.

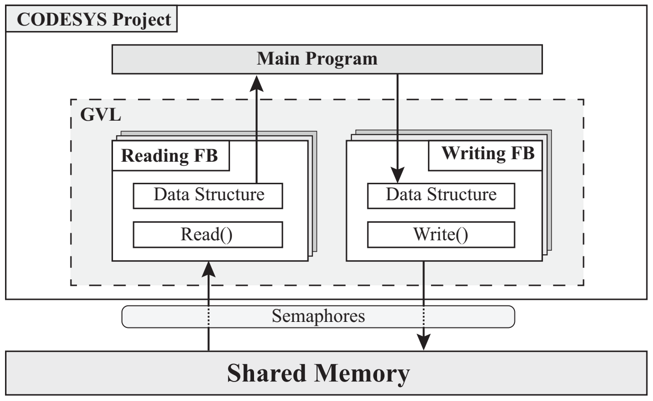

Fig. 4. Conceptual overview on the proposed CODESYS shared memory interface.

a data structure similar to the one used on the ROS interface must be also declared in the CODESYS domain, using the IEC 61131-3 nomenclature. Due to limitations related with object-oriented programming in IEC 61131-3, creating a template class, as was done in the ROS implementation, was not feasible. Therefore, attending also to the usability of the graphical languages of the IEC 61131-3 standard, two FBs were created for each type of message, one responsible for reading data from the shared memory location, and the other responsible for writing data to it. The instantiation of both types of FBs is done on a GVL, using the name corresponding to the associated ROS topic. Besides being the name of the topic, this denomination will also identify the respective shared memory and semaphore.

Similarly to the implementation on ROS, these FBs contains variables and methods. Variables are used to implement the semaphores, the shared memory mechanisms, and the structure related with the ROS messages. Methods are responsible for initializing the shared memory and the semaphore with the appropriate topic name, and to trigger the operations needed to access the shared memory and read its content or to write data to it.

In order to use the proposed CODESYS interface, a developer must add to its project three kinds of objects: GVLs, where the FBs used as interface with the shared memory are declared, a program to call the methods responsible to access and modify the shared memory, and a task to trigger the preceding program periodically.

C. Synchronizing the Access to the Shared Memory

The POSIX shared memory library by itself does not have any synchronizing mechanism. To avoid data corruption, a synchronizing mechanism must be implemented. For this purpose, CODESYS offers the `SysSem` and the `SysSemProcess` libraries. Both libraries implement semaphores described on POSIX.1b standard. The main difference between them is that `SysSem` library creates semaphores associating them to a pointer and the `SysSemProcess` creates semaphores associating them to a string, therefore it is commonly known as named semaphore.

Both functionalities are also available to implement on the ROS side through the API for C/C++ of the Realtime Extension library of the POSIX standards. Among other features, this library implements shared memory and semaphore functions compatible with the CODESYS ones.

The usage of the same string to identify semaphores and shared memory locations is a simplification to the proposed implementation, allowing efficient and reliable synchronization between ROS and CODESYS without a previous agreement of a specific memory location. The role of the synchronization mechanism in the overall architecture of the system can be seen in Figure 1.

D. Mapping Data Types

In order to ensure the exchange of information through the proposed IPC method, it was necessary to check the compatibility of the data types between ROS and CODESYS. As will be analyzed, in some cases where the data types were not compatible, a conversion mechanism was developed in order to establish a reliable communication.

The conducted analysis mapped which IEC 61131-3 data types corresponded to ROS message primitive data types in C++. Table I presents the relation between the ROS message primitive data types and the data types of IEC 61131-3.

TABLE I
CORRESPONDENT DATA TYPES BETWEEN ROS, C++ AND IEC61131-3

Description	ROS	C++	IEC 61131-3
Unsigned 8-bit Integer	bool	uint8_t	USINT
Signed 8-bit Integer	int8	int8_t	SINT
Unsigned 8-bit Integer	uint8	uint8_t	USINT
Signed 16-bit Integer	int16	int16_t	INT
Unsigned 16-bit Integer	uint16	uint16_t	UDINT
Signed 32-bit Integer	int32	int32_t	DINT
Unsigned 32-bit Integer	uint32	uint32_t	UINT
Signed 64-bit Integer	int64	int64_t	LINT
Unsigned 64-bit Integer	uint64	uint64_t	ULINT
32-bit IEEE Float	float32	float	REAL
64-bit IEEE Float	float64	double	LREAL
ASCII String	string	std::string	STRING
Time (secs/nsecs)	time	ros::Time	TIME
Time (secs/nsecs)	duration	ros::Duration	TIME

Generally, ROS message primitive data types are mapped to C++ data types by the ROS C++ API, and, therefore, could be directly mapped to the corresponding IEC 61131-3 data type. Time related variables and strings are exceptions, in the sense that they are mapped differently. Consequently, conversion mechanisms were developed to handle these exceptions.

ROS uses strictly typed data structures as messages, which can be composed not only of all standard primitive types, but also its arrays and nested messages. IEC 61131-3 also supports data aggregation by similar means, therefore the implementation of the interface described on this paper was developed in order to confine all messages into C structures and map them to different shared memory locations. POSIX shared memory objects are created in a virtual filesystem. In Figure 5 there is a representation of the virtual filesystem as a strip, where the data to be shared is disposed. Each shared memory location is related to a message that is enclosed to a structure, carries its own data fields and is identified by a string. Each message can have single fields (`struct 3`), multiple fields (`struct 1`) or even nested messages between other fields (`struct 2`).

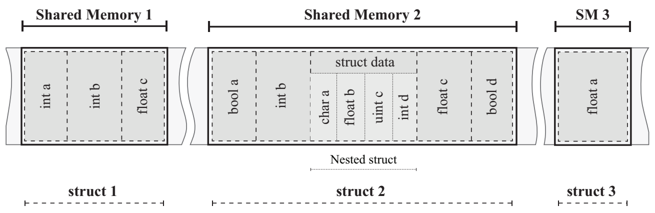

Fig. 5. Schematic representation of a shared memory virtual filesystem: `struct 1` represents a message with a single field, while `struct 2` and `struct 3` represent messages with multiple fields and a nested message, respectively.

ROS provides predefined messages to be used promptly: `std_msgs` that contains wrappers for ROS primitive types and `common_msgs` created for specific robotic tasks, which include messages for ROS actions `actionlib_msgs`, diagnostics `diagnostic_msgs`, geometric primitives `geometry_msgs`, robot navigation `nav_msgs`, and common sensors `sensor_msgs`. All predefined messages were mapped from ROS to CODESYS except the ones that uses multidimensional arrays with a non-defined length, due to limitations imposed by IEC 61131-3 when dealing with arrays of variable size.

Even though ROS common messages are frequently used in robotic applications, in a complex system it is a recurring practice to define new types of messages, composed by ROS primitive data types that are more appropriate to a certain task [6]. The proposed approach allows for an easy introduction of user-defined custom messages in the system. On ROS, to add a custom type of messages on the interface, the developer needs to create a data structure compatible with CODESYS, create a subclass of the template class derived by the type of the new message, and, finally, overload the methods used for reading and writing to the shared memory. On the CODESYS side, the developer needs to create the FBs used to access and modify the shared memory, based on generic shared memory FBs already implemented.

The introduction of user-defined custom messages is a systematic process that can be even automated with a custom script that generates code based on the message type information.

IV. Practical Application Examples

The IPC method proposed in this paper was idealized for a BeagleBone Black running the CODESYS softPLC runtime v3.5.10.30 and ROS Kinetic on Ubuntu 16.04. Since both CODESYS and ROS are supported by multiple devices running Linux, the proposed interfaces could be easily achieved using other embedded devices. Therefore, the reach of practical applications for the proposed IPC method is indubitably broad, as demonstrated in Figure 6. In this section, three examples of potential practical applications are portrayed.

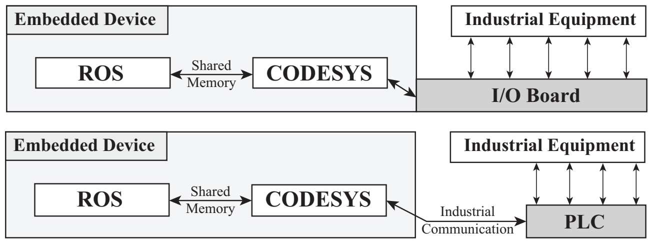

Fig. 6. Potential application examples of the proposed ROS-CODESYS bridge: on the top, ROS can directly actuate on an industrial equipment through the proposed interfaces; on the bottom, a ROS system running in parallel with CODESYS on an embedded device, such as a BeagleBone Black, can interact via industrial communication with an external PLC.

A. Using CODESYS as Communication Bridge

CODESYS framework has tools that make the implementation of fieldbuses and other industrial network protocols effortless. Even though there are already some packages implementing these protocols on ROS, the development processes can be difficult and slow.

In order to facilitate the communication between a ROS system and a device using fieldbuses, a CODESYS project could be developed just to handle the external communications in an uncomplicated way. The information to be transmitted to the device could be published on a ROS topic, subscribed and mapped to a shared memory, read by CODESYS and then handled by the CODESYS fieldbus implementation. It could also work on the other way, where the device transmits data via a fieldbus to CODESYS, that is then responsible for mapping that information through the shared memory mechanism to be read and published on a ROS topic.

This approach could be specially handy if it is necessary to communicate with a device trough a protocol not yet implemented on ROS, such as OPC-UA. OCP-UA is currently supported on CODESYS and it would be easy to share information between ROS and an OPC-UA device, even maintaining the structural composition of ROS messages.

B. Horizontal Integration between ROS and CODESYS

CODESYS is not only helpful in dealing with network connections, a robot developed in ROS can also take advantage of other tools and plug-ins provided by CODESYS, such as advanced motion control, which implementation is simpler than developing controllers for motor drivers on ROS. CODESYS also offers a simple way to create visualizations for HMIs that can be accessed through the web that could facilitate the integration of an user interface for a robot.

An industrial machine can also be programmed using CODESYS as softPLC. Therefore, using the proposed bridge, effective horizontal integration between machines and robots can be achieved. For example, a mobile manipulator can have the task of machine tending on multiple industrial machines controlled by CODESYS. When each machine finishes its task, it can communicate with the mobile manipulator to signalize that the finished product is ready for pick up using the proposed ROS-CODESYS bridge.

C. Programing robotic tasks in IEC 61131-3 programming languages

The parametrization of robotic applications can also be done using CODESYS. To achieve this configuration, robotic applications developed on ROS must be expecting data coming from the CODESYS system. In this way, robotic tasks can be activated and parameterized using CODESYS.

This can be helpful in industrial contexts due to the lack of robotic specialists in the shop floor of an industrial plant. In contrast, it is common to have operators familiar with the IEC 61131-3 programming languages. Therefore, these operators can use their knowledge to reprogram robots using IEC 61131-3, without the necessity of delving into complex robotic systems or programming tools.

V. Conclusion

This paper proposes an IPC method to establish interoperability between robotic systems and automation machinery, in the form of a shared memory interface implemented on ROS and CODESYS. Besides specifying the development approach followed in the implementation of both interfaces, this paper also delves into mapping data types between ROS and IEC 61131-3, and the conceived synchronization mechanisms to handle concurrent access to the shared memory location.

It is expected that the proposed approach can play an important role in linking PLCs, programmed in conformity with the IEC 61131-3 standard, and robotic systems, developed with the ROS framework. This can not only promote the horizontal communication between robots and industrial machines, answering the contemporary industrial needs, but also make robot re-programming easier to automation technicians, contributing to a better acceptance of robotics in industry.

Future work will include automated scripts to handle the integration of user-defined custom messages autonomously, assessments with diverse industrial equipment and robotic systems in practical industrial scenarios, and integration with service-oriented communications, for a broader scope of utilization.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 723658.

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e Tecnologia, within project SAICTPAC/0034/2015- POCI-01-0145-FEDER-016418

References

[1] P. Leitão, V. Mařík, and P. Vrba, "Past, present, and future of industrial agent applications," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2360–2372, 2013.

[2] W. He and L. Da Xu, "Integration of distributed enterprise applications: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 35–42, 2014.

[3] G. Reinhart, S. Krug, S. Hüttner, Z. Mari, F. Riedelbauch, and M. Schlögel, "Automatic configuration (plug & produce) of industrial ethernet networks," in *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*. IEEE, 2010, pp. 1–6.

[4] T. Hannelius, M. Salmenpera, and S. Kuikka, "Roadmap to adopting opc ua," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. IEEE, 2008, pp. 756–761.

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[6] A. Santos, A. Cunha, N. Macedo, R. Arrais, and F. N. Dos Santos, "Mining the usage patterns of ros primitives," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 3855–3860.

[7] M. Tiegelkamp and K.-H. John, "Iec 61131-3: programming industrial automation systems," 1995.

[8] M. De Sousa, "Proposed corrections to the iec 61131-3 standard," *Computer Standards & Interfaces*, vol. 32, no. 5-6, pp. 312–320, 2010.

[9] M. Zhang, Y. Lu, and T. Xia, "The design and implementation of virtual machine system in embedded softplc system," in *Computer Sciences and Applications (CSA), 2013 International Conference on*. IEEE, 2013, pp. 775–778.

[10] D. H. Hanssen, *Programmable Logic Controllers: A Practical Approach to IEC 61131-3 Using CODESYS*. John Wiley & Sons, 2015.

[11] M. Fuchs, C. Borst, P. R. Giordano, A. Baumann, E. Kraemer, J. Langwald, R. Gruber, N. Seitz, G. Plank, K. Kunze *et al.*, "Rollin'justin-design considerations and realization of a mobile platform for a humanoid upper body," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 4131–4137.

[12] A. Pott, C. Meyer, and A. Verl, "Large-scale assembly of solar power plants with parallel cable robots," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE, 2010, pp. 1–6.

[13] M. de Sousa and H. Sobreira, "On adding iec61131-3 support to ros based robots," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–4.

[14] R. Benosman, K. Barkaoui, and Y. Albrieux, "A new dynamic ipc-memory allocator based on a paging approach," in *High Performance Computing and Simulation (HPCS), 2013 International Conference on*. IEEE, 2013, pp. 382–389.