DINASORE: A Dynamic Intelligent Reconfiguration Tool for Cyber-Physical Production Systems

Eliseu Pereira, João Reis, Gil Gonçalves

SYSTEC - Research Center for Systems and Tecnologies
Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
Email: {eliseu, jpcreis, gil}@fe.up.pt

***Abstract*—The nowadays industrial digital revolution demands for software driven solutions where reconfiguration is one of the key enablers to achieve smart manufacturing by easy deployment and code reuse. Despite existing several tools and platforms that allow for software reconfiguration at the digital twin / edge level, it is most of the times difficult to make use of state of the art algorithms developed in the most popular programming languages due to software incompatibility. This paper presents a novel framework named *Dynamic INtelligent Architecture for Software MOdular REconfiguration* (DINASORE) that implements the industrial standard IEC 61499 based in Function Blocks (FB) in Python language for Cyber-Physical Production Systems' implementation. It adopts the 4DIAC-IDE as graphical user interface (GUI) to ease the design and deployment of FBs to quickly and on-demand reconfigure target equipment. The proposed framework provides data integration to third party platforms through the use of OPC-UA. The test scenarios demonstrate that the proposed framework 1) is flexible and reliable for different applications and 2) the CPU and memory workload linearly increases for a large amount of FBs.**

***Index Terms*—Cyber-Physical Systems, IEC 61499, Smart Manufacturing, Machine Learning**

I. Introduction

One of the key aspects of the nowadays fourth industrial revolution is the digitization of shop-floor entities like processes, equipment and components to increase their interoperability with users and information systems. In order to achieve digitization, an increased effort of standardization is required to create uniformed interfaces that promote a transparent communication among a set of heterogeneous entities. This standardization is often attained with the concept of digital twin (DT), and is often seen as a wrapper used to integrate any device or process into a network, where information can be easily accessed and shared [1]. In industry 4.0 context, a Cyber-Physical Production System (CPPS) is a distributed system of networked digital twins representing industrial processes, controllers, components, and any sort of information technology (IT) software. These CPPSs should allow for dynamic reconfigurability, software reusability and an external service orchestration [2]. On the one hand, by improving the accessibility via DTs, users can have a better grasp of the holistic shop-floor dynamics through the integration with information systems (vertical integration) such as Manufacturing Execution Systems (MES) or Enterprise Resource Planning (ERP). On the other hand, it is possible to explore new ways of data sharing among shop-floor entities (horizontal integration) promoting a distributed control system for continuous monitoring and process optimization.

With a change in paradigm from closed programmable logic controller (PLC) implementations to industrial PCs that allow a more flexible information sharing and storage, new opportunities taking advantage of this transparency can be explored. From multi-agent systems to artificial intelligence applications, democratizing data storage and sharing is key for the new advances in manufacturing systems, where machine learning is one of the key enablers of industry 4.0. This way, applications like artificial vision to detect production defects, predict when and why a certain component will fail or even explore new energy efficiency solutions are some examples of the well established importance of artificial intelligence in manufacturing.

Albeit not specific for manufacturing applications, there are several platforms developed in order to ease the development of such intelligent systems in a modular fashion, where the main idea is to accelerate the implementation of an end-to-end solution without the need to know the technical details of certain techniques. Some examples of this modular design and execution are Rapidminer Studio [3], Microsoft Azure Machine Learning Studio [4], Cloud AI from Google Cloud [5]. All these platforms have a strong graphical user interface (GUI) based in components that, through drag and drop, a complex machine learning system is possible to be built.

However, the use of such platforms in industrial applications, mainly at the level of control systems, is not straightforward. On the one hand, some of these are Cloud-based solutions, which is still an obstacle for specific industries nowadays due to the industry's policy, restricting the data access only to local network components. On the other hand, these modular designs do not implement any industrial standard, such as IEC 61499 adopted in 2005 and based in the function block (FB) definition that abstracts both software and hardware modules suitable for manufacturing requirements.

Based on this, the present paper proposes a framework called DINASORE for the execution of digital twins in Cyber-Physical Production Systems that is able to support the latest advances in machine learning. DINASORE stands

CEUR-WS.org/Vol-2739/paper_9.pdf

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

for *Dynamic INtelligent Architecture for Software MOdular REconfiguration* and is compliant with the 4DIAC platform [6] that implements the standard IEC 61499 [7]. The proposed framework is a Python environment for any system (embedded or not) that is able to run Python 3.6, or above, for the execution of function blocks (FBs) developed in Python language. The same way FORTE is used together with 4DIAC for the low level interaction and execution of C/C++ applications in industrial equipment, DINASORE is a similar implementation but for Python FBs, which makes possible the use of important machine learning packages such as TensorFlow, PyTorch, Keras and Theano.

From this perspective, with DINASORE it is possible to develop a distributed control system for industrial applications which is machine learning-enabled. This framework allows to overcome the difficulty to use and develop state of the art algorithms due to the C/C++ (FORTE) requirement. Therefore, all the latest developments in the Python community, from deep learning to optimization, can be used in industrial applications with DINASORE. It is capable of online-reconfiguration of Python FBs, where an easy to understand Python FB template is provided for customized developments. On top of that, and due to industrial requirements, each DINASORE environment has an embedded Open Platform Communications - Unified Architecture (OPC-UA) server with a data model to facilitate the integration with third-party platforms. The OPC-UA data model abstracts the concepts of equipment and device for fully industrial integration with other devices or information systems. In sum, the main contributions and differentiation of this work lie in:

1) A Python implementation of a digital twin that is integrated with 4DIAC and compliant with IEC 61499;
2) A Python template to build FBs compliant with DINASORE;
3) OPC-UA data availability using an OPC-UA server;

The remainder of the present work is organized in five more sections. A literature review is made in Section II, and the DINASORE implementation explained in Section III, from its architecture to the Python template definition to develop new FBs. The following section presents the test case scenarios defined and the main results obtained. Finally, Section V will discuss the results obtained and draw some conclusions and future work for the current implementation.

II. Related Work

One of the most widely used platforms for component-based design and execution of distributed control systems is 4DIAC [6], an Eclispe-based IDE that implements the standard IEC 61499 [7] based in function blocks (FBs). This platform has a vast set of functionalities, from FB design, system design based on a pipeline of FBs, to the deployment of this system in a distributed environment. From CNC applications [8], Distributed Time-Critical Systems (DTCSs) such as aerospace applications [9], to Smart Grids [10], the use of 4DIAC for the implementation of IEC 61499 is becoming well established due to its easiness of system design and execution for distributed environments.

There is already a great motivation in using these approaches, mainly the development of CPPSs using the IEC 61499 standard, as can be seen in literature. A use case that implements such an approach is presented in aluminum cold rolling mill plant demonstrator where the authors design and test an event-driven process based on IEC 61499 standard using OPC-UA [11]. Another similar use case that used the same philosophy is about Oil&Gas production where the authors have implemented a CPPS to deal with the complexity of equipment data on existing production processes, based on IEC 61499 and OPC-UA [12].

In [13] the authors implement a CPPS based in FORTE, and deployed the configuration system into a couple of Raspberry Pi 3 Model B that mimics an industrial process composed of a human-machine interface, a handling system and a conveyor belt. Further, the authors have integrated the OPC-UA into a more complex system [14], with the aforementioned scenario, plus a stack station. The major difference from the proposed framework is the use of FORTE and OPC-UA data model through a Service Interface Function Block (SIFB). In the case of DINASORE there is no SIFB since OPC-UA is embedded in specific FBs and automatically provides data in a OPC-UA server with no effort to the final user.

Regarding the technologies used to support the development of CPPS based on the IEC 61499, there are a set of platforms already available that can be used. Some examples are the Archimedes [15], FBDK [16] and FUBER [17] in Java language (Archimedes also supports C++); FBBeam [18] in Erlang; FORTE and nxtIECRT in C++; ISaGRAF [19] using IEC 61131-3 standard; Icaru-FB [20] and RTFM-RT in C [21]. Demonstrating the relevance of OPC-UA in such approaches, there's a work that presents an implementation of a Service Interface Function Block (SIFB) for OPC-UA communications in 4DIAC [22]. For a more comprehensive understanding of these platforms, there's also a small review about the topic [23].

Some effort has been applied to the integration of UML with the IEC 61499, where a case scenario composed by a distributing and sorting process using FESTO FMS-200 FORTE platform was built resulting in a CPPS [24]. Portability among NxtStudio, FBDK and 4DIAC platforms was also already explored [25] to have a cross-platform implementation of a CPPS, and increase the integration capabilities when building a CPPS.

Regarding all the works previously presented, the DINASORE framework presents an additional, but important, step towards the implementation of the IEC 61499 standard using Python language, and consequently, the use of state of the art machine learning algorithms in CPPSs. Together with the OPC-UA for vertical integration, DINASORE can be seen as a powerful framework that can accelerate and strengthen the next generation of smart industry.

III. Implementation

Similar to FORTE (4DIAC-RTE) portable implementation in C++ of IEC 61499, DINASORE[1] shares the same philosophy but in Python language. With all the latest artificial intelligence advances and current implementations in Python, DINASORE can be seen as a tool that enables advanced machine learning systems to execute as close to shop-floor equipment as possible. The further integration with OPC-UA for sharing a simple and intuitive data model allows for each digital twin DINASORE implementation easy to integrate with most information systems.

A. 4DIAC-IDE

The graphical user interface (GUI) integrated with the DINASORE is the 4DIAC-IDE, which enables drawing and deploying distributed configurations, based in FBs. The 4DIAC-IDE uses the Eclipse Project as a core development framework, providing a GUI based in a desktop application, with the typical Eclipse IDE appearance. The process of development of a new CPPS based in FBs has as main steps: 1) the definition of CPPS network configuration, specifying for each device its IP address and port used for 4DIAC communication, 2) the drawing of the FB pipeline, drag&dropping FBs and linking them through specific connections, modeling the required software architecture, 3) the mapping of specific FB to devices, and 4) the deployment of the FB pipeline to the corresponding DINASORE devices. Therefore, the 4DIAC-IDE GUI has several views directed to the user to implement different steps, e.g. the network configuration, the development and the deployment views. After the development and deployment of the FB pipeline, 4DIAC-IDE enables to monitoring (watch) the whole system for real-time visualization of the current state of each data and event inputs and outputs in the configuration. Additionally, the 4DIAC-IDE allows the interaction with the actual pipeline, triggering events, stopping the configuration, or cleaning the actual DINASORE runtime environment resources.

The standard IEC 61499 defines several rules at the industry level, including the composition and structure of FBs, which is the graphical representation of a set of functionalities. Each FB contains events and variables to communicate with other FBs, where each event allows to trigger the execution of a certain FB, while each variable stores the data (e.g. sensor measurements, algorithm outputs). The 4DIAC-IDE uses FBs as elementary components that connect among themselves, using both events and variables forming a functional pipeline. Considering that, there are three types of FBs defined by IEC 61499, namely:

1) Basic Function Blocks (BFBs): In simplistic terms, they are state machines that according to specific events are able to execute the corresponding algorithms;
2) Composite Function Blocks (CFBs): It is a composition of BFBs making up a network of FBs to model more complex system behaviors;
3) Service Interface Function Blocks (SIFBs): It allows to specify how FBs should interface in terms of both events and data connectors to other FBs that execute (mainly) in different physical platforms (Machine-2-Machine communication).

Additionally, the developer has the autonomy to implement their own FBs and integrate them in both 4DIAC-IDE and DINASORE runtime environment. The main type of FBs adopted in the DINASORE is the Basic Function Block (BFB), characterized by two files, an XML metadata file containing the FB structural information and a Python file implementing the code functionalities.

B. DINASORE

The main goal of DINASORE is to serve a gateway to machine learning into distributed control systems based on IEC 61499. As we believe the future design and deployment of CPPSs will definitely pass through the use of block-based technologies, such as 4DIAC-IDE and IEC 61499 as depicted in the related work, DINASORE can be seen as a key complementary technology to enrich the area of CPPS with artificial intelligence algorithms. Although integrated with 4DIAC-IDE, DINASORE is not designed in its root to execute in embedded systems, complying with real-time constraints as FORTE. The idea behind using Python language is to enable the latest advances in machine learning to integrate at the edge level with existing shop-floor equipment, without the need for cloud based processing. With the embedded implementation of OPC-UA in the DINASORE function blocks (FBs), any external data is automatically provided in a OPC-UA server for further system integration.

As for DINASORE implementation, there's the need to classify the used execution model type framing the technology into the IEC 61499 guidelines. One of the most well known categorization of Execution Control Chart (ECC) execution model was proposed by Ferrarini in [26], where 7 different classes were defined, from A0 to A6, exploring two dimensions, namely scan order and multitasking. The scan order refers to execution models that can have either fixed (predefined order), or dynamically (order calculated on-the-fly) FB execution during the ECC, while the multitasking dimension refers to no controlled ways of multitasking, where FBs execute in a multi-threading fashion; done by time slice allocation to each FB execution (preemptive scheduling) and done by FB slice, where each FB executes at a time (non-preemptive scheduling).

Additionally, there's a small and brief survey of run-time environment (RTE) platforms for IEC 61499 where 4 types of execution models are introduced [23]: 1) Buffered Sequential Execution Model (BSEM) [27]; 2) Cyclic Buffered Execution Model (CBEM) [16]; [26]; 3) Non-Preemptive Multithreaded Resource (NPMTR) [28]; 4) Preemptive Multithreaded Resource (PMTR) [23]. For a more formal definition of the BSEM, CBEM and NPMTR please refer to [29]. Despite the authors in the survey present a table that integrates RTE platforms with Ferrarini model and execution models, there's not

[1] Available Online: github.com/DIGI2-FEUP/dinasore

a one-to-one correspondence of Ferrarini model to execution models. This happens because there are some categories in the Ferrarini model without a corresponding execution model. This way, it is important to first fill this gap in terms of formal definition, and only then use it to frame the DINASORE. Hence, the complete set of Ferrarini categories is presented in the following list, with the corresponding execution models:

- A0: The execution of each FB is calculated on-the-fly depending on the input events of each one. One formalized execution model that meets this category is the BSEM;
- A1: The execution of each FB as a thread-object is made in parallel, like in a multi-threading fashion. We name this execution model as Multithreaded Resource (MTR);
- A2: To each of the executing FBs as a thread-object is given a small time slice of execution, where the allocation of time slice to FB is dynamically done. The most similar execution model is PMTR, briefly defined in [23]. However, due a dynamic scan order, we name this execution model as Buffered Sequential-Preemptive Multithreaded Resource (BS-PMTR);
- A3: Each FB as a thread-object should execute one at a time, and executed dynamically as soon as a notification is created. One formalized execution model that meets this category is the NPMTR. However, due a dynamic scan order, we name this execution model as Buffered Sequential-Non-Preemptive Multithreaded Resource (BS-NPMTR);
- A4: The execution of each FB is predefined beforehand. One formalized execution model that meets this category is CBEM;
- A5: To each of the active (that requires execution due to an event input) FBs as a thread-object is given a small time slice of execution according to a fixed order that follows a list or active FBs. This can be viewed as a PMTR, however, due to fixed scan order, we name this execution model as Cyclic Buffered-Preemptive Multithreaded Resource (CB-PMTR);
- A6: Each FB as a thread-object should execute one at a time according to a fixed order that follows a list or active FBs. This can be viewed as a special case of the NPMTR, however, due to fixed scan order, we name this execution model as Cyclic Buffered-Non-Preemptive Multithreaded Resource (CB-NPMTR);

TABLE I
AGGREGATION OF FERRARINI MODEL [26] WITH EXISTING [23] AND NEW EXECUTION MODELS FOR IEC 61499.

	Multitasking Implementation			
	Not Used	**Not Controlled**	**Time Slice**	**FB Slice**
Dynamic Order	BSEM (A0)	MTR (A1)	BS-PMTR (A2)	BS-NPMTR (A3)
Fixed Order	CBEM (A4)	x	CB-PMTR (A5)	CB-NPMTR (A6)

As for the DINASORE, the closest category is A2, where we have a thread-object per FB with all threads executing using a time slice scheduling and the execution order for all FBs is dynamically calculated by the received event inputs. Since Python language is used together with *threading* package, once we have multiple FBs thread-objects executing at the same time, a time slice scheduling strategy is used. This package uses an implementation called Global Interpreter Lock (GIL) that manages the execution time per thread, being around 5ms. This way, this implementation is not truly multithreaded (A1), but A2 using a similar approach as BS-PMTR.

The DINASORE execution model implementation, Figure 1, uses a producer-consumer pattern, where each FB performs in a different thread, which is both producer and consumer. The data transmitted between FBs addresses both events and variables, where the FB object stores the input events in a queue and the variables in register attributes. Thus, each FB object has an internal queue for the input events, waiting until it receives an input event in the data structure, reading after the variables' actual value, and processing the event's functionality. After completing the functionality execution, the FB object pushes the corresponding output events in the queue of the following connected FBs. The same thing happens to the variables where the FB updates their output variables and consequentially the input variables of the following linked FBs. The actual value of each FB event and variable is available through the monitoring/communication interfaces, i.e., using the OPC-UA server or 4DIAC-IDE *watch* option.

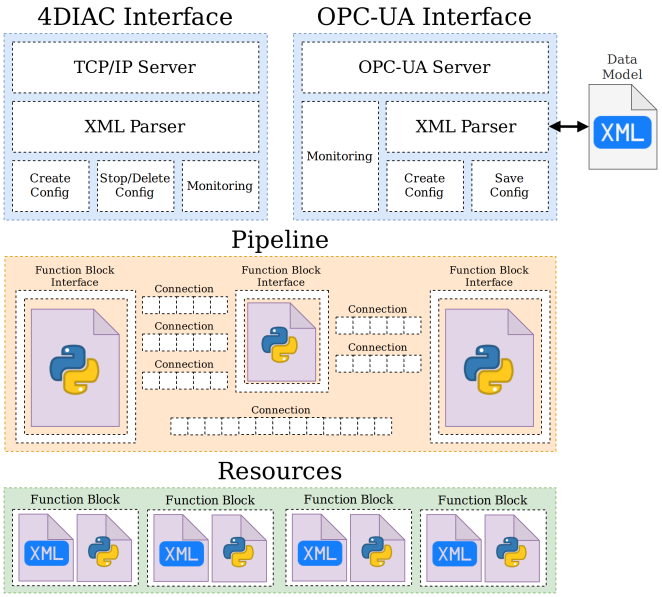

Fig. 1. DINASORE Architecture.

The FB thread-object requires two external files to execute that compose the FB itself. The first file is XML-based and contains the meta-information about the FB, e.g. the FB type, the FB class and the FB structure composed by input/output events and variables, and their details, including the data type

and the OPC-UA role (variable, method or none). Besides the DINASORE usage, the metadata file enables the 4DIAC-IDE to render the FB with their events and variables in the GUI. The second file composing the FB is a Python script encoding the FB functionalities, which requires the implementation of a class, assuming the Object-Oriented (OO) paradigm. That class requires the implementation of the *schedule* method, which receives as arguments the event name and respective value, triggering the FB execution, with the current input variable values. Then, according to the event received, the method selects the functionality to execute. After executing the functionality, the *schedule* method returns a list of output events and variables. Both *schedule* method arguments and output variables should follow the order specified in the metadata file.

Concerning the need for external communication between the DINASORE and other applications, e.g. 4DIAC-IDE and third party information systems, there are two different and independent communication interfaces integrated in the DINASORE, 1) the 4DIAC interface, which uses TCP/IP, and 2) the OPC-UA interface, which uses a data model XML file as a reconfiguration file. These two interfaces allow the reconfiguration of the current runtime workflows and the monitoring of each FB. The 4DIAC interface communicates with the 4DIAC-IDE using a TCP/IP connection, where the DINASORE interface executes a TCP server, receiving the commands from the 4DIAC-IDE enabling the creation, stop, and deletion of the configuration workflow and the runtime configuration monitorization. The interaction between 4DIAC-IDE and DINASORE for the configuration creation starts with several messages instantiating every FB (*Create FB*); each message contains the FB type and instance name. Then the 4DIAC-IDE sends the commands to create the connections (events and variables) between FBs (*Create Connection* and *Write Connection*), and finally the IDE requests the start of the FB threads (*Start Configuration*). After starting the workflow, the DINASORE enables the monitoring of each FB variable and event, through the *watch* option (the *Create Watch* message to activate the subscription; the *Read Watch* message to request the variable/event current value; and the *Delete Watch* message to unsubscribe). Additionally, the DINASORE allows from the 4DIAC-IDE to stop and reset (*Stop Configuration*) the configuration workflow, terminating the respective working threads and the remote trigger of an event (*Trigger Event*), performing a FB functionality in the DINASORE.

As an alternative, the DINASORE can use the OPC-UA Data Model as reconfiguration channel. This way, the DINASORE stores the configuration locally in the Data Model XML file, where registers all the used FBs, and their respective connections (events and variables). The FB XML meta-information classifies each FB in different sets, grouping them in 1) devices, 2) equipment, 3) services, 4) endpoints, and 5) start-points:

1) The device abstraction represents sensors, like sensors integrated using Modbus protocol;
2) The equipment representation uses a more complex structure allowing the aggregation of sensors and actuators, using events as relational connections;
3) The service type maps to FBs as a method and when an event is received, its execution is triggered. This is suitable for machine learning algorithms (e.g., Random Forest, SVM) or statistical operations (e.g., moving average, moving standard deviation);
4) The start-point type represents protocols interfaces that receive data, e.g. subscribing a MQTT topic or requesting TCP/IP data;
5) The endpoint type provides data through the defined interface, e.g. publishing in a MQTT topic and replying to a TCP/IP request.

Those representations make the development of new FBs easier by providing templates and specific behaviors to maximize the effectiveness of the Python code written in each FB. The (1) device, (2) equipment, and (4) start-point scripts adopt a loop template, which enables a cyclic execution of the FB. The (3) service and (5) endpoint types use the typical asynchronous approach executing when triggered.

Both OPC-UA and TPC/IP (4DIAC-IDE) interfaces are essential to DINASORE operation, enabling different features and capabilities. The 4DIAC communication interface allows the combination with a graphical user interface (GUI) for the development of workflows based on FBs. The OPC-UA Data Model, besides enabling the workflow reconfiguration, the primary purpose is to monitor the execution of a specific pipeline monitoring using the OPC-UA industrial protocol and the storage of the device configuration in an XML format. The configuration storage transforms the DINASORE in a more reliable and fault-tolerant platform, enabling the memorization of the current state of the workflow, being tolerant of power cuts restarting the runtime environment in the previous state. The Data Model XML file updates every time, when the DINASORE receives commands from the 4DIAC-IDE, for the creation, stop, and removal of configuration workflows.

IV. Test Case Scenarios

The DINASORE evaluation focuses on the validation of the tool in different scenarios. Those scenarios point out the platform advantages and disadvantages, besides establishing the most suitable applications, like sensing, control, or data processing. The first scenario focuses on the use of machine learning (ML) algorithms, in particular classification methods, for the detection of collisions in a real mini-robotic arm based in servo motors. The second example consists of a typical distributed control architecture composed of two components (gripper and robotic arm) that have to synchronize between them to perform the required operation. The third case shows the simulation of a manufacturing production line. Based on the simulation scenario, several experiments with increasing amounts of FBs allow to assess the scalability of the framework in terms of processing and memory usage.

A. Collision detection based on servo motors analysis

The main goal of the collision detection implementation is to transform a typical robotic arm into a collaborative one. That scenario uses a robotic arm based in servo motors that provide load metrics able to infer about possible collisions with obstacles. Based on this, the process 1) monitors each servo motor, 2) detects when one servo motor is in overload, and 3) stops the robotic arm if it collides with an obstacle. The network architecture contains two devices, both Raspberry's Pi: 1) responsible for monitoring the servo motors and check the motors overload (Figure 2 rose FBs), and 2) controlling the robotic arm, sending instructions to perform a particular task, and waiting to receive an overload alert to stop its execution (Figure 2 green FB).

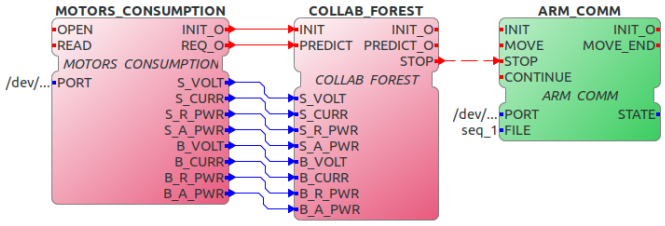

Fig. 2. Collaborative Robot Workflow.

The servo motors sensing component uses as hardware an Arduino Uno board (ATMega328P microcontroller), which collects data of the voltage and current for each motor, using a potential divider to measure the voltage and an amplifier to obtain the current. The Root Mean Squared (RMS) uses the last 250 samples to calculate the voltage and current RMS value, which allows the computation of the real and apparent power. This data processing is performed at the microcontroller level, transmitting the eight features (4 metrics of 2 servo motors) through the serial port to the Raspberry Pi, which uses a FB, implementing the device template, to read and parse the data. After parsing the data and calculating the respective lag features (moving average and standard deviation) for the last 10 samples, the overall features (total of 24 variables) feed a classification method. The model predicts if the robotic arm is performing with regular efforts (output at zero) or in overload/collision situations (output at one). The classification model training uses an offline dataset, collected from the serial port, containing the robotic arm performing different operations with and without collisions. Several classification algorithms, including Support Vector Machine (SVM), Random Forrest (RF), and Artificial Neural Networks (ANN), were validated using the precision, the recall, and the f1-score as performance metrics. The more accurate model was the RF, exported to perform online on the second pipeline FB, that implements the service template. The model output predictions generate a stop event sent, through multicast sockets, to the second Raspberry Pi, which controls the robotic arm. The communication between the two Raspberry Pis, due to the usage of UDP multicast sockets, adopts a paradigm producer-subscriber, where the collision detection component produces stop events when in overload, sent to the robotic arm control component. The robotic arm control component (green FB, using the device template), in a different Raspberry Pi, reads from a text file, the sequence of motor positions to perform, and sends them sequentially through the serial port to the embedded controller. The robotic arm performs, in a cycle, the list of instructions until it receives a stop event from the user, using the 4DIAC-IDE, to restart the operation.

B. UR5 Collaborative Robotic Arm and Gripper Control

The synchronized control between a Universal Robots 5 (UR5) robotic arm and a 3D printed gripper requires the usage of a CPPS composed of two Raspberry Pi, similar to the previous scenario. Considering that architecture, one component controls the servo motor that opens and closes the gripper (Figure 3 blue FBs), and the other sends commands to the UR5 robotic arm (Figure 3 purple FBs). The operation performed by both devices consists in 1) catch one object at position A, 2) go with the piece to position B, 3) go back to position A, 4) leave the object in position A, 5) go without the item to position B, 6) go without the object to position A, and restarts the cycle.

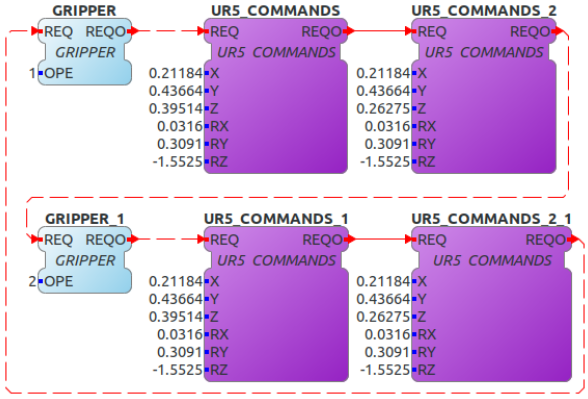

Fig. 3. UR5 Robotic Arm and Gripper Workflow.

The UR5 robotic arm uses an API to send commands through a TCP/IP communication channel with the physical controller, which enables the continuous flow of instructions (X, Y, Z positions and Rx, Ry, Rz rotations) to the UR5 robotic arm. The FB, using the service template, wraps the API function to move the robotic arm to a specific location (X, Y, and Z) with the rotation of the joints (Rx, Ry, and Rz). The gripper developed to pick up objects contains three different parts 1) a Raspberry Pi, 2) a servo motor that opens and closes the gripper, and 3) a 3D printed Polylactic Acid (PLA) casing. The movement of the servo motor opens and closes the 3D printed fingers, according to the instruction sent from the Raspberry Pi interface using the General Purpose Input/Output (GPIO). Thus, a FB, adopting the service template, encapsulates the control of GPIO pins using an input variable to establish the operation to perform (open or close). The communication in

the CPPS uses multicast sockets to send events between FBs that are in different devices (different colors), as used in the previous scenario.

C. Manufacturing Applications

The simulation of a manufacturing production line addresses several challenges, like the material tracking on the shop-floor or equipment sensorization. Regarding the flow of materials along the production line, the developed simulation includes each station of the process, with its attributes, like the current manufactured material and the respective transporter, the sensors associated, and the operation time. The simulated representation mainly serves as an advantage for the process industries, which have a sequential set of productive steps, being able to simulate different layouts to optimize productivity.

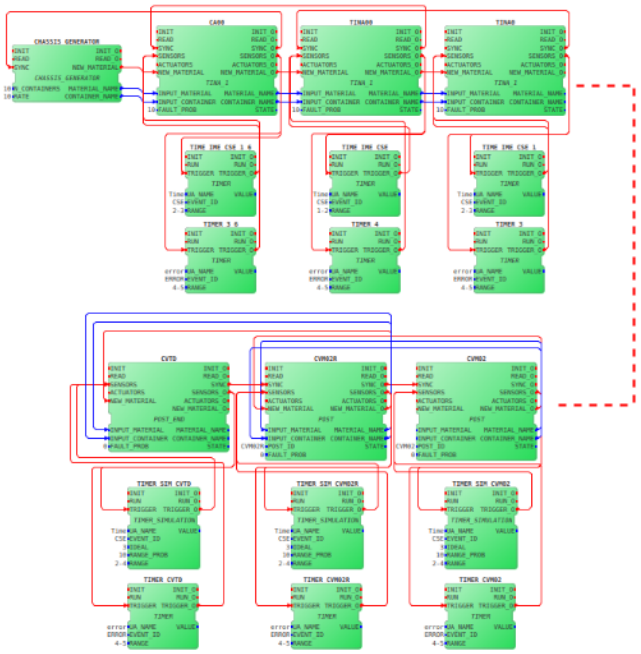

Fig. 4. Manufacturing Scenario.

Figure 4 presents partially the FB pipeline of the simulation implementation, where each simulated station uses three FBs to model its behavior. The main FB, that adopts the equipment template, represents a station and implements a state machine with the following states: 1) *unscheduled*, waiting for new material to produce; 2) *standby*, arrives a material and transporter associated and waits to start the operation; 3) *productive*, working in the operation of the station; and 4) *error*, a stochastic state to simulate an extra time producing the material. The additional two FBs, implemented using the service template, allow to simulate the production time and error time (if any). All the series of three FBs represent the entire production line with its characteristic sequential U shape and bifurcations. Additionally, this approach enables the integration of real components with in this simulation environment, turning the pipeline more accurate.

D. Performance Evaluation

The manufacturing line simulation scenario serves as a basis for performance and workload evaluation for the DINASORE framework. By using a series of three FBs that represent a station it is possible to assess the CPU and memory usage with a varying number of FBs. The *htop* monitoring tool (linux) was used enabling the profiling of each process in the operating system. Typically, the memory usage has a constant value for the same input parameters; however, the CPU usage has high variability, which requires more samples to obtain an accurate estimation. The higher number of collected samples for the same scenario, generates a more substantial variability, computed in the form of standard deviation. Figure 5 considers both metrics, representing the collected samples for the memory and the average and standard deviation for the processing usage.

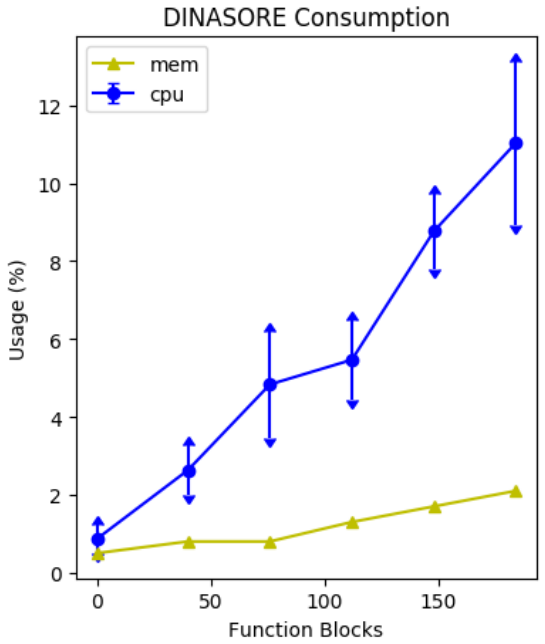

Fig. 5. DINASORE Performance Results.

The main property to evaluate on the DINASORE is how an increasing number of FBs influences the framework usage of computational resources. This kind of test requires a large number of FBs (up to 200) focusing on the DINASORE management of the runtime environment. The hardware specifications of the host machine are 16GB of RAM and an Intel Core i7 processor, with 12 cores and a frequency of 2.20GHz. The analysis of the results indicates a growing trend in both metrics with an increasing number of FBs. That trend follows the natural behavior of computational systems: more complexity causes more resources consumed, with the CPU curve following a rate of approximately 18 FBs per 1% of CPU usage. Those values prove the reliability and scalability of the developed solutions, considering the target hardware, which varies from low computational power devices (e.g., Raspberry

Pi) to high performance machines (e.g., servers). Such experiments' main intention is solely to validate the performance of DINASORE, not considering CPU and memory usage in terms of FB functionality (e.g., training a deep neural network).

V. Conclusion

Looking in perspective, the DINASORE framework enables the deployment of powerful Python algorithms for CPPSs, following industrial standards globally adopted, like the IEC 61499 and OPC-UA protocol. The proposed framework increases the flexibility of the traditionally closed and hard to re-configure industrial systems, being a step forward the high-mix low-volume paradigm. The validation scenarios prove the flexibility and reliability of the DINASORE, giving the necessary freedom to developers for a multitude of different application implementations, like sensor integration, equipment control, data processing, or communication protocols. The performance evaluation demonstrates the scalability in a DINASORE runtime environment with an increasing amount of FBs. Nevertheless, the DINASORE transforms a heavy-weight local application into a distributed solution performing in a clusters of devices. Considering that, the platform provides capabilities to scale up the solutions for a larger amount of devices, isolating the applications into FBs, and communication through popular IoT protocols, like MQTT or OPC-UA.

As for the future work, and assuming that DINASORE enables the use of computation-intensive machine learning solutions, the exploration of speculative computation concept, as implemented in [30] for FB-based systems, will be explored. The central idea is to implement this concept as a background process in DINASORE so the execution can be accelerated transparently. An additional objective is the continuous implementation of new methods using the FB structure to easily experiment with them in different industrial applications due to the FB portability between systems.

Acknowledgment

INDTECH 4.0 - New technologies for intelligent manufacturing. Support on behalf of IS for Technological Research and Development (SI à Investigação e Desenvolvimento Tecnológico). POCI-01-0247-FEDER-026653

References

[1] G. Gonçalves, J. Reis, R. Pinto, M. Alves, and J. Correia, "A step forward on intelligent factories: A smart sensor-oriented approach," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[2] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," *International Journal of Distributed Sensor Networks*, vol. 12, no. 1, p. 3159805, 2016.

[3] O. Rittho, R. Klinkenberg, S. Fischer, I. Mierswa, and S. Felske, "Yale: Yet another learning environment," in *LLWA 01-Tagungsband der GI-Workshop-Woche Lernen-Lehren-Wissen-Adaptivität*, no. 763. Citeseer, 2001, pp. 84–92.

[4] M. Corporation. (2018) Microsoft azure machine learning studio. [Online]. Available: https://azure.microsoft.com/pt-pt/services/machine-learning-studio/

[5] G. AI. (2017) Google cloud ai. [Online]. Available: https://cloud.google.com/products/ai/

[6] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel, "Framework for distributed industrial automation and control (4diac)," in *2008 6th IEEE International Conference on Industrial Informatics*, July 2008, pp. 283–288.

[7] G. Cengic, O. Ljungkrantz, and K. Akesson, "Formal modeling of function block applications running in iec 61499 execution runtime," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*, Sep. 2006, pp. 1269–1276.

[8] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa, "A novel open cnc architecture based on step-nc data model and iec 61499 function blocks," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 560–569, 2009.

[9] C. C. Insaurralde, "Modeling standard for distributed control systems: Iec 61499 from industrial automation to aerospace," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 2016, pp. 1–8.

[10] F. Andrén, T. Strasser, and W. Kastner, "Model-driven engineering applied to smart grid automation using iec 61850 and iec 61499," in *2014 Power Systems Computation Conference*. IEEE, 2014, pp. 1–7.

[11] T. Terzimehic, M. Wenger, A. Zoitl, A. Bayha, K. Becker, T. Müller, and H. Schauerte, "Towards an industry 4.0 compliant control software architecture using iec 61499 & opc ua," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2017, pp. 1–4.

[12] M. V. García, E. Irisarri, F. Pérez, M. Marcos, and E. Estévez, "Engineering tool to develop cpps based on iec-61499 and opc ua for oil&gas process," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2017, pp. 1–9.

[13] M. V. García, F. Pérez, I. Calvo, and G. Morán, "Building industrial cps with the iec 61499 standard on low-cost hardware platforms," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–4.

[14] M. V. García, F. Pérez, I. Calvo, and G. Moran, "Developing cpps within iec-61499 based on low cost devices," in *2015 IEEE World Conference on Factory Communication Systems (WFCS)*. IEEE, 2015, pp. 1–4.

[15] K. Thramboulidis and A. Zoupas, "Real-time java in control and automation: a model driven development approach," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1. IEEE, 2005, pp. 8–pp.

[16] V. Vyatkin and J. Chouinard, "On comparisons of the isagraf implementation of iec 61499 with fbdk and other implementations," in *2008 6th IEEE International Conference on Industrial Informatics*. IEEE, 2008, pp. 289–294.

[17] G. Cengic, O. Ljungkrantz, and K. Akesson, "Formal modeling of function block applications running in iec 61499 execution runtime," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 2006, pp. 1269–1276.

[18] L. Prenzel and J. Provost, "Fbbeam: An erlang-based iec˜ 61499 implementation," in *IEEE International Conference on Industrial Informatics (INDIN'19)*, 2019.

[19] J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, A. Zoitl, J. Chouinard, H. Mayer, and A. Kopitar, "The iec 61499 function block standard: Software tools and runtime platforms," *ISA Automation Week*, vol. 2012, 2012.

[20] L. I. Pinto, C. D. Vasconcellos, R. S. U. Rosso, and G. H. Negri, "Icaru-fb: An iec 61499 compliant multiplatform software infrastructure," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 3, pp. 1074–1083, 2016.

[21] P. Lindgren, M. Lindner, A. Lindner, D. Pereira, and L. M. Pinho, "Rtfm-core: Language and implementation," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, 2015, pp. 990–995.

[22] S. Kožár and P. Kadera, "Integration of iec 61499 with opc ua," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–7.

[23] L. Prenzel, A. Zoitl, and J. Provost, "Iec 61499 runtime environments: A state of the art comparison," in *17th International Conference on Computer Aided Systems Theory (EUROCAST 2019)*, 2019.

[24] E. X. Castellanos, C. A. Garcia, C. Rosero, C. Sanchez, and M. V. Garcia, "Enabling an automation architecture of cpps based on uml combined with iec-61499," in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2017, pp. 471–476.

[25] A. Hopsu, U. D. Atmojo, and V. Vyatkin, "On portability of iec 61499 compliant structures and systems," in *2019 IEEE 28th International*

Symposium on Industrial Electronics (ISIE). IEEE, 2019, pp. 1306–1311.

[26] L. Ferrarini and C. Veber, "Implementation approaches for the execution model of iec 61499 applications," in *2nd IEEE International Conference on Industrial Informatics, 2004. INDIN'04. 2004*. IEEE, 2004, pp. 612–617.

[27] G. Cengic and K. Akesson, "Definition of the execution model used in the fuber iec 61499 runtime environment.[in:] industrial informatics, 2008. indin 2008," in *6th IEEE International Conference on*, 2008, p. 301.

[28] C. Sunder, A. Zoitl, J. H. Christensen, V. Vyatkin, R. W. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J. L. Martinez-Lastra, and F. Auinger, "Usability and interoperability of iec 61499 based distributed automation systems," in *2006 4th IEEE International Conference on Industrial Informatics*. IEEE, 2006, pp. 31–37.

[29] G. Cengic and K. Akesson, "On formal analysis of iec 61499 applications, part b: Execution semantics," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 2, pp. 145–154, 2010.

[30] D. Drozdov, V. Dubinin, and V. Vyatkin, "Speculative computation in iec 61499 function blocks execution—modeling and simulation," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE, 2016, pp. 748–755.