EDDE: An Event-Driven Data Exchange to Accurately Introspect Cobot Applications

Emil Stubbe Kolvig-Raun
University of Southern Denmark
Universal Robots, PhD Fellow
eskr@universal-robots.com
eskr@mmmi.sdu.dk

Mikkel Baun Kjærgaard
University of Southern Denmark
Professor, Center for Software Engineering
mbkj@mmmi.sdu.dk

Ralph Brorsen
Universal Robots
Technology Lead, Customer Care
rbr@universal-robots.com

***Abstract*—Commercialized collaborative robots (cobots) are typically programmed in proprietary languages and employ interfaces that gather data at predefined frequencies, such as the Real Time Data Exchange offered by Universal Robots (UR). This approach challenges the observability of program execution during runtime. To address this issue, the paper proposes applying software architectural knowledge from the introspection of Cyber-Physical Systems (CPSs) to the robotics domain. The proposed solution is to replace the classical protocol interfaces with an Event-Driven Architecture, enabling the acquisition of program events in relation to state variables and thus, richer information concerning runtime. The proposed architecture, called the Event-Driven Data Exchange, is implemented and evaluated on a UR e-Series cobot in a practical study. The study highlights the significance of implementing software architectural knowledge in CPSs, such as cobots to extend observability and data availability.**

***Index Terms*—Event-Driven Architecture, Industrial Automation, Collaborative Robots, Program Monitoring, Debugging and fault localization, Data Collection, Software traceability**

I. Introduction

Cyber-Physical Systems (CPSs) often combine software and hardware from various fields, leading to diverse software architectures. Classical protocol interfaces limit the ability to thoroughly introspect these systems, resulting in difficulty obtaining precise runtime information for examining program execution events [1]–[3]. This issue persists in the robotics domain, potentially hindering debugging, optimization, and performance. As such, there is a significant opportunity to apply software architectural knowledge at the lowest levels of the software stack to address these challenges.

Robots can be programmed to execute various tasks within a work cell, also commonly referred to as the robot application. Safety and maintenance represent substantial concerns in the robotics field [4], [5]. To address safety, robots have been engineered with advanced safety mechanisms, allowing operation in near proximity to humans. This design approach has resulted in the development of collaborative robots (cobots).

In the domain of robotics, software contributions, such as the Robot Operating System (ROS), have been instrumental in promoting innovation. However, in the commercial sector, robots are often programmed using proprietary languages [7]. Additionally, cobot applications are increasingly complex, requiring more coding for regular maintenance and optimization due to the combination of hardware and software components. Therefore, robot programs are rarely a one-to-one transfer [2], [3]. The growth of robotic automation further underlines the importance of methods for comprehensive introspection of cobot applications to enable accurate analytics and examination of programming-related runtime execution [6].

Commercial cobot manufacturers offer classic frequency-based data collection interfaces, such as the Franka Emika Control Interface (FCI) [8], Fanucs' Remote Sensor Interface (RSI) [9] and Universal Robots (UR)' Real Time Data Exchange (RTDE) [10], which provide information about the cobot's state, i.e., force-torque sensor values, joint angles and velocities, or motor temperatures. However, the raw time series obtained from these interfaces are difficult to synchronize accurately with program lines, which increases processing requirements and reduces the accuracy of diagnostics and maintenance. Despite the availability of tools such as the URLogViewer [11], which enable the examination of robot state variables over time, the precise segmentation of time series and program execution events is still lacking in the robotics domain. This poses a challenge to debugging and validating cobot application programs, as time series alone are often insufficient to associate specific events with program lines.

This paper proposes, implements, and evaluates the integration of an Event-Driven Architecture (EDA) as a replacement for the classic frequency-based data generation model to enable accurate introspection of cobot applications. An EDA is a software architecture promoting the creation and reaction to events [12]. By introducing this architecture to the lowest level of the software stack, data generation can be segmented based on runtime events. This idea is evaluated in an industrial setting, highlighting the importance of low-level event-filtering in cobot applications.

We make the following contributions:

- The Event-Driven Data Exchange (EDDE), which accurately associates robot state information with program events.
- A generalizable architecture that can be replicated for cobots using proprietary languages.

This work is supported by the Innovation Fund Denmark for the project DIREC (9142-00001B).

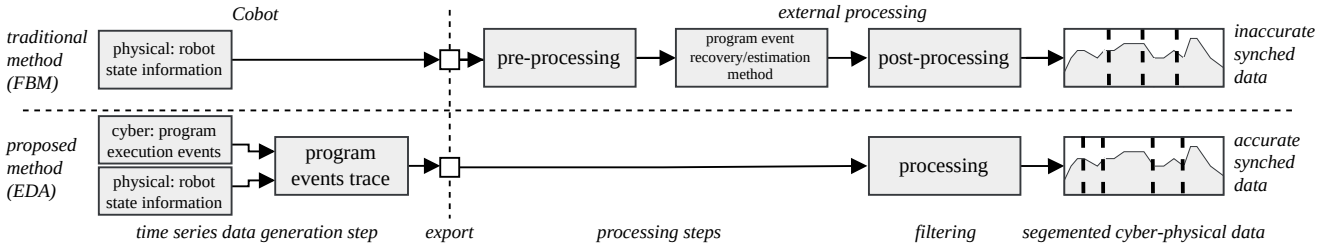

Fig. 1: Depicts how event-driven methods produces better time series data than frequency-based methods. The proposed method combines program execution events (cyber) with robot state information (physical) to create a single event trace.

- An extensive evaluation demonstrating how the new data can support cobot analytics and optimization methods.
- A practical study that highlights the benefits of applying software architectural knowledge to data generation in the robotics domain.

II. Related work

The adoption of software architectural methods from CPSs into robotics has led to increased traction from the software engineering community [1]–[3], [13], [14]. This interest is, in part, associated to the shift from domain-specific cobot applications to multi-robot systems, which introduces new requirements for the robot software architecture, e.g., observability [15].

A. Event-driven architectures in robotics

Decentralized service oriented architectures have been progressively adopted in the robotics domain [3], [13], partly due to early contributions such as ROS [7]. ROS provides a structured management and processing layer above a homogeneous communication layer, which unites different edge devices and supports scalable CPSs, thus enabling the integration of technological equipment and agents within a cobot application. To enhance system reliability, software architecture introspection techniques from CPSs, such as those discussed in [2], have been used to statically detect misconfiguration bugs that may occur during runtime initialization of system components. These bugs are defined by ROS launch files, which determine the procedural and configurable settings for system initialization.

Robot manufacturers commonly provide drivers that enable integration with ROS. In the processing layer of ROS, services utilizing event handlers, which implement the observer pattern, can be deployed to trigger on specific thresholds related to a data stream. This is likely facilitated with connection to a conventional data collection interface or data distribution protocol. Several methods can be employed to analyze program events using these interfaces. One such method is based on pattern recognition, which detects recurring structural occurrences in the extracted time-series data, also known as Motifs in machine learning [17]. However, model-based techniques can be highly inaccurate and never provide a precision greater than the resolution of the time series. Another approach is to integrate additional code lines into the original program, commonly known as ”code between the lines” [18]. This solution could trigger a digital output before and after each program line to indicate the data transition in a robotics program, but supplementary semantics increase code complexity. Both methods rely on data that may be off by one in each time step. While ROS event handlers promote the use of event-driven processing pipelines, they operate on top of frequency-based technologies.

B. Event-driven architectures for introspecting CPSs

Although a contribution of comprehensive event-driven architectures to enhance the introspection of cobot applications during runtime has yet to be realized, an example of how such a solution can benefit CPSs is presented by NEAT in the domain of energy-efficient smartphone applications [16]. NEAT provides software that autonomously fuses sensor readings with event logs extracted from the phone’s operating system, concurrently recording events from the Android kernel and system/user space, i.e., cyber data, and power measurements, i.e., physical data, merging them into a combined power trace. By implementing an EDA close to the data source, NEAT maintains cohesion and eliminates the challenge of accurately obtaining data during post-processing. This strategy to combine digital occurrences with physical consequences at the lowest level of the stack could inspire software solutions for introspection in the robotics domain by considering program execution events as the cyber component and the robot state information as the physical component.

III. System Concept

This work aims to develop a solution that allows for instantaneous fusion of robot state data with program execution events, enabling accurate introspection of cobot applications.

Figure 1 illustrates the disparity in processing requirements between the proposed EDA and frequency-based methods (FBM) for linking cyber and physical data. The use of FBM in extracting program information through frequency-dependent data collection interfaces necessitates multiple processing steps to obtain comprehensive robot state information in association with events. As earlier mentioned, these methods increase complexity.

The RTDE generates data outputs at a rate of 125 Hertz, while the FCI can generate data at up to 1 kilohertz [8], [10]. The precision obtained by frequency-based methods assuming perfect event recovery is at best in millisecond resolution, whereas recording data and program events in parallel allows for the merged composition to be logged accurately and exported with minimal overhead. Additionally, frequency-based

methods produce a fixed data amount that increases linearly with time, i.e., approximately 2000 bytes per push for the RTDE's inclusion of all robot state information.

By allowing the system to push data based on instantaneous relations between code lines and variables, redundant logging is eliminated, and the total amount of data is minimized. The additional output includes only an integer indicating the current sequential location of the code line within the proprietary programming language, which is 4 bytes in size.

IV. Event-Driven Data-Exchange

The classical data collection interface in the e-Series cobots provided by UR, the world's largest cobot manufacturer, is renewed by implementing the EDDE protocol. The EDDE is based on a push-based EDA [12], as shown in the component diagram in Figure 2. This presents the cobot as a CPS and illustrates how the robot program events traverse the system components and are ultimately exposed as output through the EDDE. This architectural placement does not require the use of both methods, but rather clarifies their placement in relation to one another.

Robot programs for e-Series cobots are written in URScript, the proprietary programming language of UR. This is not uncommon in industrial automation, as other examples include KRL, PDL, RAPID, AS, and KAREL. The architecture shown in Figure 2 is generalizable and transferable to other systems. However, unique to Franka Emika is that they provide a C++ library and mostly depend on ROS for deployment [20].

The URScript robot program is executed by the cobot's software system through a controller. The program lines are translated into an Abstract Syntax Tree (AST) that breaks them into functions and primitives. These functions, which are based on boilerplate code, take argument values from the script and may require physical time to execute, e.g., waypoint functions that move the cobot. Due to safety constraints, these functions are typically executed in discreet steps to ensure continuous validation and synchronization in the CPS intersection [21].

The EDDE follows an observer pattern, continuously monitoring data points and logging information at the exact time the cobot controller software transitions between steps, during runtime execution. By mapping step-wise program events to specific code lines and appropriate data points, the data collection can be replicated across robot systems assuming the necessary sensors are available. This approach offers the highest achievable precision for event-based introspection of robot programs during runtime execution, to our knowledge [16].

A. Extending data-availability

The consideration of architectural redesign extends to the availability of data with respect to robotic software observability. Conventionally, interfaces in the robotics domain provide access solely to the robot state information, which expresses the physically measurable conditions of the cobot and its internal components [8]–[10]. This includes joint and end-effector angles, velocities, accelerations, motor temperatures,

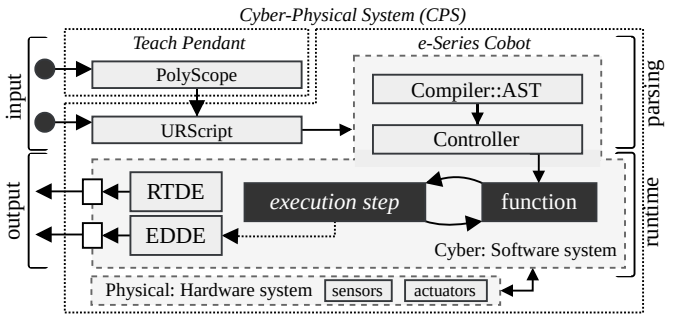

Fig. 2: Illustrates the positioning of the EDDE within the e-Series software architecture, as well as the control inputs and data outputs.

currents/voltage, as well as estimations of externally applied torques and forces, and collision information. In addition, the EDDE provides access to a multitude of information that was previously inaccessible through traditional data exposure interfaces. Specifically, the EDDE exposes information related to the entire cobot application, installation variables, and values in the computation model of the controller.

Cobot application: The EDDE exposes I/O communications, including configurable, digital, and analog signals, which are typically linked to the cobot through the end-effector, the base, or an external control box. This provides insight into the cobot application by allowing access to information on interoperable occurrences in the application, i.e., of great importance for comprehensive introspection, as the environment often affects the system's performance.

Installation variables: These can be introduced manually by the programmer or automatically during the installation of robot software, such as when installing a gripper. By allowing access to these variables in conjunction with precise event information, the EDDE facilitates high-resolution debugging of states in a robot program

Controller variables: The EDDE exposes important controller variables that provide information on program events and computational resources, such as CPU usage, memory usage, and execution time per step. It also includes angular values for joint positions and the determinant of the Jacobian matrix, which is useful for detecting singularities, i.e., where joints or the end-effector become blocked by physical constraints.

V. Evaluation

The implementation details of the EDDE in a proprietary software stack of a major robotics manufacturer are restricted. However, given the architectural representation and description of the paradigm, the methodology can be replicated. The addition of the EDDE to the stack was completed in three months and can be deployed with any e-Series cobot without impacting system response time, constrained only by a few system version requirements. To demonstrate the capabilities of EDDE, we conducted several practical studies in an industrial setting based on real measurements.

Well-informed interpretations of event-based system introspection begin with rich visualizations and the precision

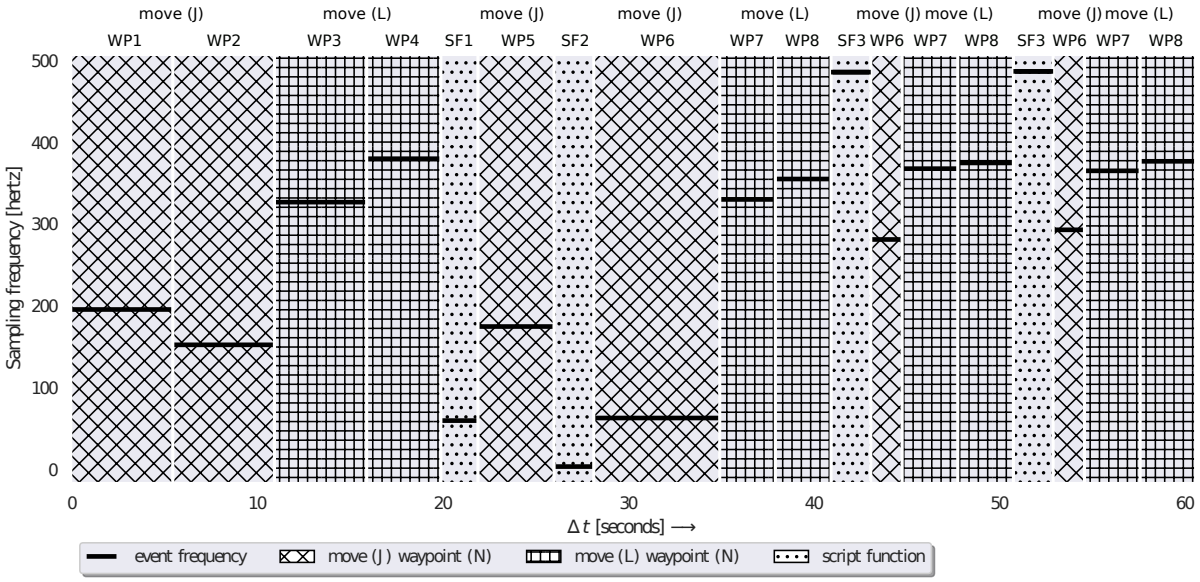

Fig. 3: Displays execution time per program line, thus event frequency during runtime, for an entire cycle. Covers Move (J) and Move (L) program lines involving Tool Center-Point (TCP) position translation, respectively, based on specified joint positions and movement along a line or plane.

provided by the EDDE interface is necessary to achieve results such as those presented in these studies.

We present the program illustrated in Figure 3. The focus of our evaluation is on a single program that encompasses the most frequently utilized features of a cobot. This program incorporates contrasting waypoint functions, several proprietary script functions, and a loop that iterates the same sequence of program lines thrice. Proprietary script functions refers to the script files that are separate from the main cobot program. The figure showcases the execution time for each line and presents information derived from a single program cycle. It also indicates the frequency of occurrences of events during the execution of each line, measured in hertz. Investigating the loop, our findings reveal that identical program lines can result in discrepancies during execution. These discrepancies are possibly linked to physical, environmental, or hardware inconsistencies and the computation of intermediate waypoints during each stepwise execution.

The findings emphasize the advantages of event-driven techniques for data generation. The amount of information varies significantly with respect to each program line due to the differing frequency of calculations performed during program execution. This implies that logging based on a fixed frequency can result in redundant or insufficient information. Moreover, although repetitive execution occurs based on identical lines, as in the case of waypoint 7, the number of events is inconsistent, leading to varying required, and acquired, sampling frequencies.

The placement of the EDDE in association with the execution model of the controller has enabled us to conduct a more thorough examination of the program, as depicted in Figure 4. The visualization provides a concrete description of the time required to execute instructions corresponding to each program line. It reveals the time spent on physical execution in relation to waypoint functions or waiting for the physical state to synchronize with the calculated state. Additionally, it offers insights into code efficiency by clearly indicating the processing time. This is particularly useful for script functions as it allows the programmer to evaluate the efficiency of proprietary implementations. Furthermore, the figure demonstrates the extent to which the instructions within each line depend on boilerplate code.

A. Comparing the measured behavior of robot state data

Figure 5 presents a comparison between trendlines obtained from the distinct interfaces. We compared the acceleration data collected from the EDDE and the RTDE interfaces to the baseline. Both measurements were recorded concurrently, and the timestamps obtained from the EDDE data were used to mark the transition from one program line to another in both plots. We restricted the x-axis range to approximately 25-30 seconds to emphasize the difference between the two signals.

The results indicate the difference in precision in relation to cohesion between robot state data and program events. The frequency of events recorded by the EDDE is directly linked to the previously mentioned event frequency. However, since events are associated with program lines and not acceleration, this value may not change with each push event. Nonetheless, the benefit of collecting data conditionally on program events is that we acquire information that pertains to changes in the application rather than unbiased information over continuous time.

By accurately segmenting program events, we can create distribution functions that show the performance of any available data type. When program lines are similar, such as different waypoint functions, these distributions can be compared and used to select methods for completing specific tasks. In Figure 6, we compare the TCP-position error calculated

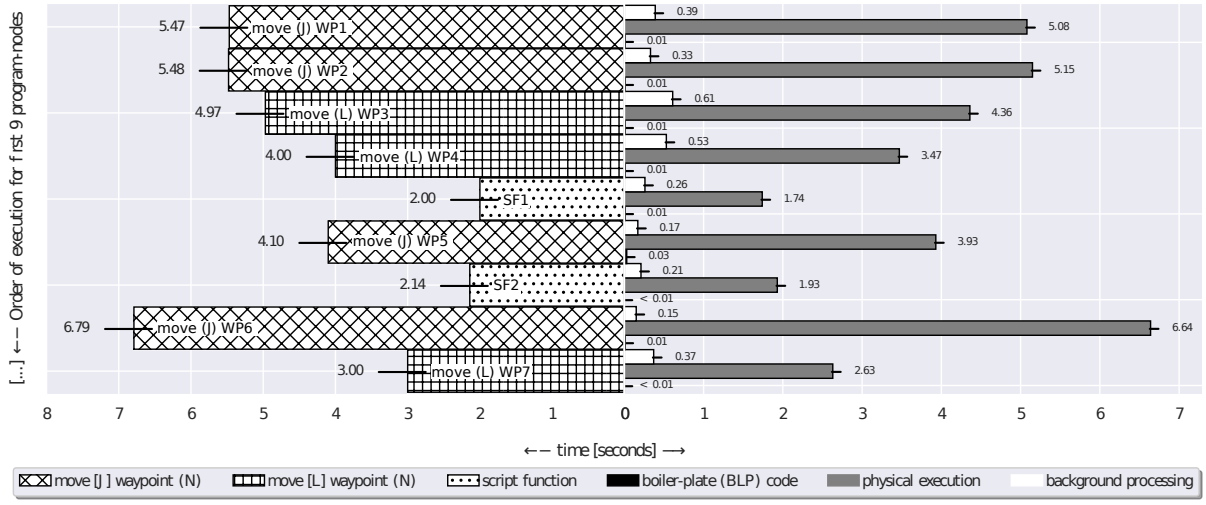

Fig. 4: Displays measurements of program execution, with each program line represented by a left-side histogram showing its associated execution time. On the right side, the fractional time for physical execution/synchronization, background processing, and boilerplate code is compared using a histogram. Includes the first 9 lines of the program also seen in Figure 3.

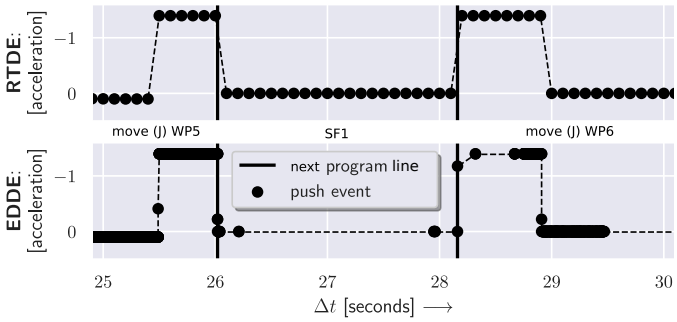

Fig. 5: Compares trendline behavior from parallel recordings using the traditional RTDE and the novel EDDE.

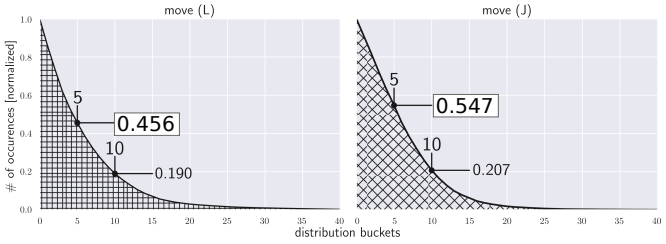

Fig. 6: Compares the distributions of calculated TCP-position errors in waypoints for Move (L) and Move (J) program lines.

between the two types of waypoint functions used in the program. The y-axis shows the normalized number of values for each bucket on the x-axis. A bucket corresponds to an interval from $((v_{max}-v_{min})/i_{max})*i$ to $((v_{max}-v_{min})/i_{max})*i+1$, where i ranges from 0 to 40. The results indicate that functions with move (L), 0.456, as the outset are slightly more precise than those based on calculations with move (J), 0.547.

The EDDE facilitates the visualization of program lines in relation to movement and proximity to a singularity, as shown in Figure 7. The trajectory's thickness and color represent the determinant value, indicating higher joint strain during the execution of waypoints with move (L). This emphasizes the need for program optimization, among others, including adjustments to the velocity, acceleration, and trajectory.

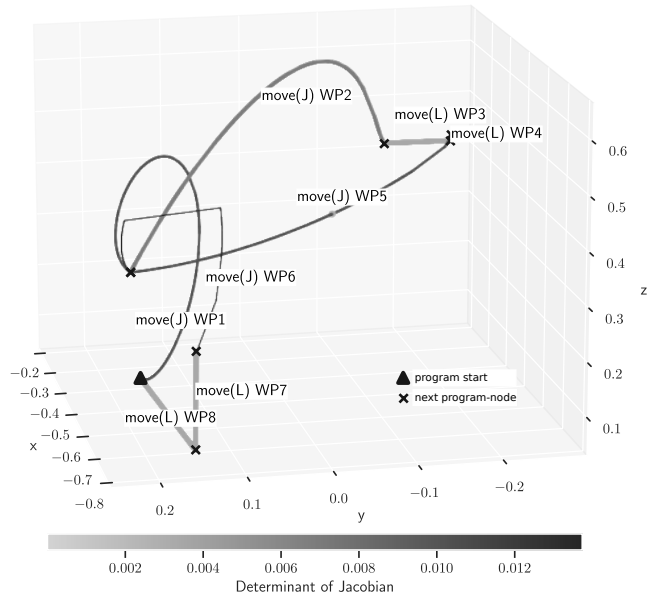

Fig. 7: Displays TCP position throughout one execution cycle, with correlation to the determinant of the Jacobian matrix to detect improper movement.

VI. Discussion

This work aims to demonstrate the potential of applying software architectures for introspecting CPSs to the field of robotics. The EDDE interface provides new opportunities for analyzing cobot applications by allowing access to data associated with specific program events. This enables programmers to debug different robot states within a program cycle and examine interactions with other application components, including I/O signals and collision information. By monitoring cobot runtime execution, faulty coding practices can be identified and application efficiency can be improved

through program optimization. This promotes the development of efficient and robust applications in the robotics domain.

The benefits of using the EDDE interface include the ability to compare the behavior and runtime execution of similar applications across different cobots. This solution can be scaled to a wider range of cobots from various manufacturers by starting with the association between the parsed proprietary programming language in the controller and the stepwise execution. However, scaling EDAs to the entire robotics domain would require a common standard that all industrial robots must adhere to. This would need to account for variations in the shape, size, and degrees of freedom of cobots from different manufacturers, e.g., while UR offers cobots with 6 joints, Franka Emika delivers cobots with 7 joints, creating a discrepancy in the calculation model.

The drawbacks of implementing the EDDE interface are primarily the need for updating interfaces in different cobot software systems and the additional 4-byte overhead. However, the benefits include the potential for data-driven models for optimization, such as improving cobot energy consumption through motion planning, as in [22]. With the EDDE interface, models can report based on independent program lines, leading to higher resolution and improved data quality for model training. Furthermore, the solution has applications in fields such as the building industry, where motor control for ventilation systems relies on pull-based data collection and machine learning for event recovery [19].

VII. Conclusion

In this paper, we have demonstrated the potential of an Event-Driven Architecture (EDA) to improve the classical protocol interfaces for data exposition in the robotics domain. By designing and implementing the Event-Driven Data Exchange (EDDE) in an industrial setting, we have shown that an EDA can provide precision and accuracy when introspecting Cyber-Physical Systems (CPS), such as collaborative robots (cobots). The EDDE's generalizable software architecture accommodates modern robot control solutions offered by many manufacturers, making the solution transferable. By incorporating access to I/O communication signals, installation variables, and computation model variables, the EDDE allows potential monitoring of occurrences in the entire interconnected cobot application. The practical study we conducted demonstrated that an EDA has the potential to grant access to new runtime information, such as the processing performed within each program line.

References

[1] S. Garcia, C. Menghi, P. Pelliccione, T. Berger and R. Wohlrab, "An Architecture for Decentralized, Collaborative, and Autonomous Robots," 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 75-7509, DOI:10.1109/ICSA.2018.00017.

[2] Christopher S. Timperley, Tobias D. ürschmid, Bradley Schmerl, David Garlan, and Claire Le Goues, Carnegie Mellon University, Pittsburgh, PA, USA, "ROSDiscover: Statically Detecting Run-Time Architecture Misconfigurations in Robotics Systems", 2022 IEEE 19th International Conference on Software Architecture (ICSA), DOI:10.1109/ICSA53651.2022.00019

[3] P. D. Francesco, I. Malavolta and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption," 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 21-30, DOI: 10.1109/ICSA.2017.24.

[4] T. P. Carvalho, F. A. Soares, R. Vita, R. da P. Francisco, J. P. Basto, and S. G. Alcalá, "A systematic literature review of machine learning methods applied to predictive maintenance", Computers and Industrial Engineering, vol. 137, 11 2019, DOI:10.1016/j.cie.2019.106024

[5] R. L. C. and H. Wang, "Diagnostic and prediction of machines health status as exemplary best practice for vehicle production system", IEEE 88th Vehicular Technology Conference (VTC-Fall), 2018, DOI:10.1109/VTCFall.2018.8690710

[6] International Federation of Robotics (IFR), "Executive summary world robotics 2021," Executive Summary (Report), 2021.

[7] Quigley, Morgan. "ROS: an open-source Robot Operating System." IEEE International Conference on Robotics and Automation (2009).

[8] Franka Emika, Franka Emika Control Interface, Documentation. [Online], Accessed: 22/11/22 https://frankaemika.github.io/docs/

[9] Modulo PI, Fanuc RSI Docs. [Online], Accessed: 23/1/22 http://support.modulo-pi.com/modulo-kinetic-manual/v4/en/topic/fanuc-rsi

[10] Universal Robots (Support), "Real-Time Data Exchange Guide", Documentation. [Online] Accessed: 23/11/22, Available: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/

[11] Universal Robots (Support), "UR Log Viewer Manual", Documentation. [Online] Accessed: 23/11/22, Available: https://www.universal-robots.com/articles/ur/robot-care-maintenance/ur-log-viewer-manual/

[12] Chandy, K.M. (2009). Event Driven Architecture. In: LIU, L., ÕZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA., pp 1040-1044. https://doi.org/10.1007/978-0-387-39940-9_570

[13] R. Pinciroli and C. Trubiani, "Model-based Performance Analysis for Architecting Cyber-Physical Dynamic Spaces," 2021 IEEE 18th International Conference on Software Architecture (ICSA), 2021, pp. 104-114, DOI:10.1109/ICSA51549.2021.00018.

[14] M. T. Moghaddam, H. Muccini, J. Dugdale and M. B. Kjægaard, "Designing Internet of Behaviors Systems," 2022 IEEE 19th International Conference on Software Architecture (ICSA), 2022, pp. 124-134, DOI:10.1109/ICSA53651.2022.00020.

[15] Federico Ciccozzi, Nico Hochgeschwender, Ivano Malavolta, and Andreas Wortmann. 2020. Report on the 2nd International Workshop on Robotics Software Engineering (RoSE'19). SIGSOFT Softw. Eng. Notes 44, 3 (July 2019), 38-40. https://doi.org/10.1145/3356773.3356804

[16] Niels Brouwers, Marco Zuniga, and Koen Langendoen. 2014. NEAT: a novel energy analysis toolkit for free-roaming smartphones. In Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys '14). Association for Computing Machinery, New York, NY, USA, 16-30, DOI:10.1145/2668332.2668337

[17] Dennis Janka, Felix Lenders, Shiyu Wang, Andrew Cohen, Nuo Li, "Detecting and locating patterns in time series using machine learning", Control Engineering Practice, Volume 93, 2019, 104169, ISSN 0967-0661, DOI:10.1016/j.conengprac.2019.104169

[18] Feichtner, J., Gruber, S. (2020). Code Between the Lines: Semantic Analysis of Android Applications. In: Holbl, M., Rannenberg, K., Welzer, T. (eds) ICT Systems Security and Privacy Protection. SEC 2020. IFIP Advances in Information and Communication Technology, vol 580. Springer, Cham, DOI:10.1007/978-3-030-58201-2_12

[19] Omid Ardakanian, Arka Bhattacharya, and David Culler. 2016. Non-Intrusive Techniques for Establishing Occupancy Related Energy Savings in Commercial Buildings. In Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '16). Association for Computing Machinery, New York, NY, USA, 21-30, DOI:10.1145/2993422.2993574

[20] Franka Emika, FCI, "libfranka, Installation on Linux". [Online], Accesssed: 11/11/22, Accessible: https://frankaemika.github.io/docs/installation_linux.html

[21] International Organization for Standardization. (2016). Robots and robotic devices, Collaborative robots (ISO/TS 15066). [Online], Accessible:https://www.iso.org/standard/62996.html

[22] J. Heredia, C. Schlette and M. B. Kjærgaard, "Data-Driven Energy Estimation of Individual Instructions in User-Defined Robot Programs for Collaborative Robots," in IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 6836-6843, Oct. 2021, doi: 10.1109/LRA.2021.3094781.