

Towards Flexible and Adaptive Human-Robot Collaborations via Event-driven Microservices

Ronny Seiger

Institute of Computer Science, University of St.Gallen

St.Gallen, Switzerland

ronny.seiger@unisg.ch

Abstract—Robotic co-working relies on the perception of the human and the robot’s surroundings to ensure safe operations. Building the respective means for sensing into the hardware and software of a robot is expensive and results in inflexible, non-extensible routines that are hardwired into the robot’s programming. In this work we investigate the application of modern software architecture principles to this problem. We propose an architecture that relies on event-driven microservices to provide fine-grained access to robots and other machines. Complex event processing (CEP) allows to augment the microservices with advanced reactive behavior by incorporating events from external sensors, process them, and control the machines accordingly. The programming of the CEP applications is facilitated by SQL-like queries that support domain experts with flexibly connecting sensors, specifying their processing, and corresponding actions.

Index Terms—Software architecture, human-robot collaboration, event processing, microservices

I. INTRODUCTION

Cobotics and human-robot collaborations are of increasing importance in modern manufacturing environments to provide small and medium-sized enterprises with means for automation, high-precision processing, and advanced manufacturing capabilities [1]. Apart from rather high costs for the corresponding robot hardware that is able to actively sense and enable safe human-robot collaborations, the actual programming of the cobots is resource intensive and inflexible due to many routines and reactions to external events being hardwired in the robots’ programming. This leads to a low adaptability of the cobots to changing environments and rather rigid scenarios for human-robot collaborations. In this work we investigate the application of modern software architecture principles and techniques—focusing on *Event-driven Microservices*—to enable a flexible high-level programming of low-cost robots that incorporate different types and numbers of external sensors to adaptively react to human interactions and their surroundings.

In this work in progress paper, we approach factory automation from a software architectural viewpoint and propose an architecture that relies on microservices as basic building blocks to control robots and other actuators. These microservices are augmented with event-driven, reactive behavior based on a complex event processing (CEP) engine that is capable of fusing and processing events from a multitude of sensors to derive higher level events and context to then instruct the microservices. By leveraging the CEP platform’s SQL-like programming language, we can flexibly specify the involved

sensors, their processing, and the resulting actions to control machines. We present a proof-of-concept prototype using multiple sensors to flexibly control two low-cost robotic arms and a conveyor belt as a use case of human-robot collaborations.

II. BACKGROUND AND PRELIMINARIES

1) *Event-driven Microservices*: The microservices architectural style is commonly used to develop modern software systems across different disciplines and industries. Microservices represent functionally cohesive, independently deployable units that are developed based on the decomposition of the problem domain and non-functional requirements. Among others, microservices promise high levels of scalability, fault-tolerance, and extensibility. In combination with event-driven behavior implemented by event processing components that asynchronously receive, process, and react on events, *event-driven microservices* additionally exhibit a high level of responsiveness [2], which are useful non-functional properties in adaptive factory automation and human-robot collaborations.

2) *Robotic Co-working Setup*: We set up a lab environment to simulate robotic co-working scenarios involving up to two low-cost robotic arms, a conveyor belt, and external sensors for the human to either directly control the machines, or to sense the human’s presence and movements. Fig. 1 depicts the setup with the two robots—*Robot Left* and *Robot Right*—and the conveyor belt as main machines. Furthermore, a total of 13 sensors of different types are available. As an extension of the scenarios, we have a smart coffee maker and a small-scale smart factory available [3]. This setup allows us to simulate human-robot collaborations and automation scenarios of different complexity, not only investigating the effects of sensed actions and user commands on one robot, but also on other machines in a partially automated production line.

III. FLEXIBLE HUMAN-ROBOT COLLABORATIONS VIA EVENT-DRIVEN MICROSERVICES

A. Software Architecture

The software architecture that we introduce is based on *event-driven microservices*. As depicted in Fig. 2, we propose to implement or augment each machine with one *microservice* controlling the corresponding physical hardware and exposing the machine’s functionality via a REST API or similar. The software applications controlling the external *sensors* publish the sensor data to a message broker on one topic (message

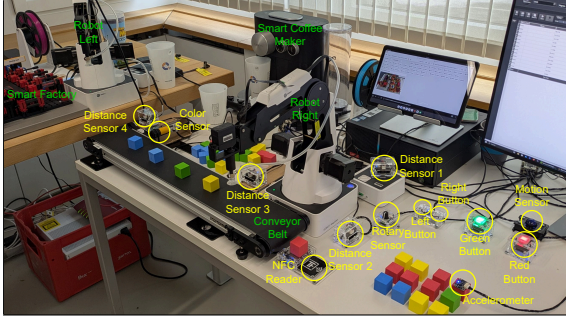


Fig. 1. Robotic co-working setup with machines (green) and sensors (yellow).

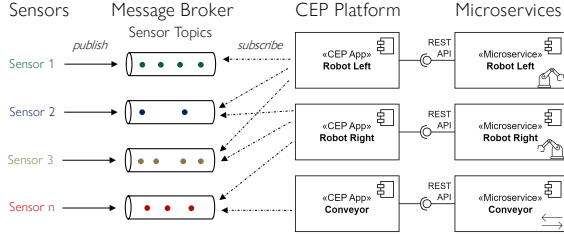


Fig. 2. Software architecture with event-driven microservices.

channel) per sensor to allow a fine-grained access. These sensor streams are picked up by *CEP Apps* running on a CEP platform for event processing (cf. Section III-C). One CEP app exclusively instructs one microservice via its API based on the result of the event processing inside the CEP app. This way, we non-invasively extend the microservices controlling the machines with event-driven behavior, while still allowing other applications and clients to use services' existing APIs.

B. Microservices for Machine Control

One of the main decisions in developing a microservice refers to its size (i.e., the scope of its functionality) to achieve a cohesive design and address relevant architectural driving forces [2]. We decided to have one microservice per machine, which facilitates fault-tolerance, extensibility, and maintenance, adhering to the single responsibility principle for fine-grained control. This decision is based on previous experience related to the available small-scale smart factory [4].

a) Robot Control Service: Since we focus on architectural aspects and interactions among services, the business logic within one microservice is kept rather simple and just exemplifies potential human-robot collaborations. The workflow to be executed by one robot is programmed using the *teach-in method* [5], [6]. For replaying these workflows, we rely on the robot's embedded programming to move between given coordinates. The microservice controlling one robot supports: 1) activating the execution of a recorded step-by-step workflow moving the robot arm with a specified speed and number of iterations; 2) changing of the movement speed during operations; and 3) pausing/resuming and stopping of the workflow execution. These commands are exposed via a RESTful API. Each robotic arm is controlled by one instance of the robot control service to enable independent control.

b) Conveyor Belt Control Service: The service controlling the conveyor belt supports: 1) starting it with a specific speed and direction; 2) changing its speed and direction; and 3) stopping it. These commands are exposed via a RESTful API. Both, the conveyor belt service and robot service do not implement any behavior to react to external events, their functionality can only be actively triggered via their APIs.

c) Sensor Service: The size and nature of the application(s) sending sensor readings to the message broker might vary and be tied to the specific sensors, their processing capabilities and their locations. Here we do not impose any restrictions but only assume that sensor readings are either published at a continuous frequency, or only when values change, on a one-topic-per-sensor basis to a message broker.

C. Complex Event Processing Topologies and Patterns

CEP enables the processing of vast amounts of events from multiple event streams to identify patterns and perform operations deriving higher level events [7]. CEP apps running on the CEP platform are at the heart of the software architecture. They act as mediators between the sensors and microservices by consuming readings from the sensors, processing them, and activating the microservices accordingly. Regarding the event processing logic inside a CEP app, we distinguish between two different types of event-driven behavior to support human-robot collaborations: 1) *Active Machine Control* by the user, and 2) *Reactive Machine Control* based on user presence and movements. Following, we conceptualize the corresponding event correlations, patterns and topologies to describe different types of $(n : m)$ relations between n sensors and m machines. Together with a schematic topology, we provide *Queries* in the event processing language (EPL) *StreamingSQL* employed in the CEP platform *Siddhi* [8] that we are using. The EPL simplifies the *wiring* of sensors with machines to also enable domain experts to specify event-driven behavior.

1) Active Machine Control: The active machine control scenarios illustrate the triggering of commands by the user via the existing sensors. We distinguish between continuous control based on events that are continuously emitted from a sensor to adjust the operation of a machine, and the single-command based control to trigger an action from a single discrete sensor event. Fig. 3 demonstrates the continuous control: the values from the rotary sensor are used to control the speed of the conveyor belt, with either forward or backward rotation. Listing 1 shows the EPL query to *filter* (line 2) and *translate* (line 3) the relevant events containing the rotary position as an attribute on the rotary stream into the corresponding speed and backward direction. Complex events are emitted to the *ConvSpeed Stream*, which instructs the conveyor belt via its API. A similar query is required to move the belt forward. Both queries are part of the *Conveyor* CEP app.

An example for the single-command based control with one sensor instructing multiple machines is given in Fig. 4. Here we simulate an emergency stop where the pressing of the red button instructs all three machines to stop. The query to filter the events from the red button stream for events with state

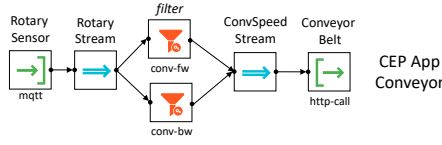


Fig. 3. One sensor continuously controlling one machine.

```
@info(name='conv-bw')
1 from RotaryStream[position < 0]
2 select (position/150)*-1 as speed, -1 as direction
3 insert into ConvSpeedStream;
4
```

Listing 1. Query for translating the negative values of the rotary sensor into commands for the conveyor belt to move backwards.

pressed (line 2) and to send a stop event to the left robot is given in Listing 2. This query is part of the *Robot Left* CEP app, similar queries are in the Robot Right and Conveyor apps.

2) **Reactive Machine Control:** Reactive machine control becomes important when a human collaborates with the robot or works in its vicinity. External sensors perceive the human's actions and react to ensure workplace safety. In Fig. 5 we present the case of a motion sensor indicating the presence of a human and corresponding reactions. The two CEP apps encode a different reaction to the absence of human motions: the speed of the left robot's operations is increased as it is responsible for automated tasks, interacting with the conveyor belt and smart factory; the right robot is stopped as it is interacting with the human, who has left the workplace. Listing 3 shows the query to evaluate the motion sensor and to instruct the left robot. The sensor is continuously emitting its status, we are only interested in the changes from a detected motion to no motion. We use the stateful *sequence* event pattern [7] to analyze two subsequent events on the stream to detect the occurrence of a change in the attributes (lines 2–3) and subsequently emit an event to increase the robot speed (line 4).

To improve the perception of the human actions, we add a distance sensor to the motion sensor which allows us to make the robot control more fine-grained. Fig. 6 presents the new topology, which considers the values of both sensors. Before being able to evaluate their attributes together, a *join* of the

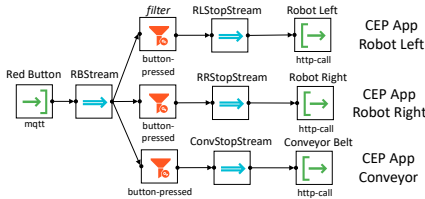


Fig. 4. One sensor instructing three machines to execute the same command.

```
@info(name='button-pressed')
1 from RBStream[state_pressed == 1]
2 select 'stop' as stop insert into RLStopStream;
3
```

Listing 2. Query for stopping the left robot based on the pressing of a button.

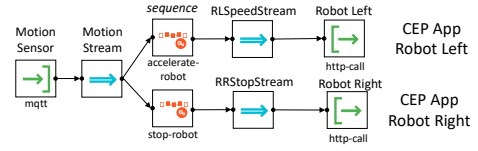


Fig. 5. One sensor instructing two machines to execute different commands.

```
@info(name='accelerate-robot')
1 from every e1 = MotionStream, e2 =
2 MotionStream[(e1.motion == 1) and (e2.motion == 0)]
3 select 60 as speed insert into RLSpeedStream;
4
```

Listing 3. Query for accelerating the left robot based on a change in the motion sensor.

events on the two streams within a specific time window is required [7]. Listing 4 shows the joining of the sensor events in a 2 seconds time window with a preceding filtering of values (lines 2–4) as we are only interested in active motions and distances in a certain range to slow down the robot.

IV. DISCUSSION

The event topologies presented in Section III-C demonstrate the advantages of using CEP to add reactive behavior to microservices for machine control. The correlations and processing of different numbers of arbitrary sensors and their effects can be flexibly specified using combinations and chains of CEP patterns (e.g., filtering, sequence and pattern detection, aggregations and joins [7]). This allows for specifying advanced behavior including sensor prioritization and regulations regarding the human-robot interaction and real-time constraints as CEP patterns. Thereby, the EPL queries abstract from imperative programming concepts with their SQL-like syntax to facilitate end-user programming in a *low-code* style. Microservices provide fine-grained, isolated access to machine functionality, which can be flexibly *wired* to the respective sensors to adapt to the current machine operations and context.

The examples showcase sensors and machines in a simplified lab setup to investigate novel software architectural approaches towards flexible and adaptive machine control. More advanced sensors could be integrated, but they need pre-processing and discretization to be suitable for the event

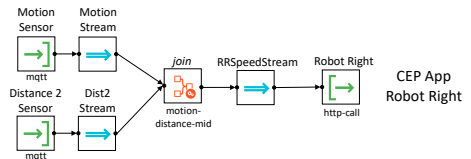


Fig. 6. Two different sensors fused to adjust the speed of one machine.

```
@info(name='motion-distance-mid')
1 from Dist2Stream[(dist > 10) and (dist <= 30)] as D
2 join MotionStream[motion == 1]#window.time(2 sec) as M
3 on D.location == M.location
4 select 20 as speed insert into RRSpeedStream;
5
```

Listing 4. Query for joining two sensors to adjust the speed of the right robot.

pattern specifications. Moreover, machines and robots, including their workflows, are simplified as our focus is on software architecture with human-robot collaboration being a specific use case which benefits from reactive behavior enabled via a multitude of sensors. The proposed software architecture combining microservices with CEP is generic and can also be applied to other use cases in the context of factory automation and beyond. One significant limitation of our proposal is the aspect of reactive behavior in real-time, which is relevant to ensure safety and security. We acknowledge that with the proposed distributed system, low-cost sensors and robots, and software components, we cannot ensure real-time guarantees regarding the reactive behavior. A deeper investigation of real-time aspects and capabilities of the CEP engine and microservices with a focus on distributed, real-time CEP with multiple CEP engines and event hierarchies to move CEP closer to the edge devices is subject to future work. This also comprises the handling of concurrent access to the microservices' functionalities that are exposed via REST.

V. RELATED WORK

This work touches upon multiple relevant disciplines of research. A plethora of related work on human-robot collaboration has been developed over several decades with advanced hardware and software-based solutions [1]. In our work, human-robot collaboration is a use case that requires reactive behavior and context perception to adapt to the user [9]. The actual control of a robot is hidden inside the microservices and thus, it can be very simple but also advanced as long as an API to manipulate the robot is available. The work in [10] discusses important operation modes a cobot should support to ensure the safe collaboration with humans. These modes could be implemented inside the machine-controlling microservices and changed using the CEP-based context perception within our proposed software architecture. The exemplary use of microservices to implement robot-related tasks in home-care [11], delivery [12], and in manufacturing scenarios [3], [13] demonstrates the increasing importance of software architecture in industrial automation.

As discussed in [14], CEP and machine learning are mostly used for real-time data analytics, but active machine control and actuation have been out of scope so far. Barricelli et al. propose in [15] to leverage the means for pattern specification in event processing to visually program the IoT, which could also be extended to enable end-user programming of robots [6]. Using and abstracting the presented CEP queries to a visual programming approach for end-users is subject to our future investigations. The novel aspect and contribution of this work in progress is a flexible software architecture based on modern software engineering principles to allow for flexible, software-based programming and automation.

VI. CONCLUSION AND FUTURE WORK

We present an approach towards enabling adaptive and flexible human-robot collaborations via event-driven microservice. The focus is hereby on a novel software architecture

that combines the flexibility of microservices with powerful complex event processing (CEP) to implement advanced reactive behavior based on data from an arbitrary number and processing of sensors. The SQL-like syntax used in the CEP platform allows to easily specify the relevant sensors, their processing, and the corresponding reactions in a generic way, which makes CEP a suitable candidate for many automation tasks that require reactive behavior.

In future work, we will investigate the integration of more advanced sensors (e.g., cameras) to improve the multi-modal detection of the human movements and attention in human-robot co-working scenarios. Furthermore, we will conduct performance tests of the system to investigate the impact of the event processing on real-time behavior and we will conduct end user-related studies to evaluate the ease-of-use when programming the reactive behavior. In this line, we will also extend our use cases to other automation scenarios.

REFERENCES

- [1] A. Ajoudani, A. M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, and O. Khatib, "Progress and prospects of the human-robot collaboration," *Autonomous robots*, vol. 42, pp. 957–975, 2018.
- [2] M. Richards and N. Ford, *Fundamentals of Software Architecture: A Modern Engineering Approach*. "O'Reilly Media, Inc.", 2025.
- [3] R. Seiger, L. Malburg, B. Weber, and R. Bergmann, "Integrating process management and event processing in smart factories: A systems architecture and use cases," *Journal of Manufacturing systems*, vol. 63, pp. 575–592, 2022.
- [4] R. Seiger and L. Malburg, "Revision of a smart factory software architecture from monolith to microservices," in *Int. Conf. on Enterprise Design, Operations, and Computing*. Springer, 2024, pp. 203–220.
- [5] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Proceedings of the Australasian conference on robotics and automation*, vol. 1, 2003, pp. 1–3.
- [6] G. Ajaykumar, M. Steele, and C.-M. Huang, "A survey on end-user robot programming," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–36, 2021.
- [7] O. Etzion and P. Niblett, *Event processing in action*. Manning Publications, 2010.
- [8] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, and V. Nanayakkara, "Siddhi: A second look at complex event processing architectures," in *Proceedings of the 2011 ACM workshop on Gateway computing environments*, 2011, pp. 43–50.
- [9] D. Hostettler, S. Mayer, and C. Hildebrand, "Human-like movements of industrial robots positively impact observer perception," *International Journal of Social Robotics*, vol. 15, no. 8, pp. 1399–1417, 2023.
- [10] U. Aßmann, C. Piechnick, G. Püschel, M. Piechnick, J. Falkenberg, and S. Werner, "Modelling the world of a smart room for robotic co-working," in *Model-Driven Engineering and Software Development (MODELSWARD)*. Springer, 2018, pp. 484–506.
- [11] C. Di Napoli, G. Ercolano, and S. Rossi, "Personalized home-care support for the elderly: a field experience with a social robot at home," *User Modeling and User-Adapted Interaction*, vol. 33, no. 2, pp. 405–440, 2023.
- [12] M. Schrick, J. Hinkeldeyn, M. Thiel, and J. Kreutzfeldt, "A microservice based control architecture for mobile robots in safety-critical applications," *Robotics and Autonomous Systems*, vol. 183, 2025.
- [13] R. Seiger, M. Franceschetti, and A. Abbad-Andaloussi, "A process to non-invasively augment legacy iot systems using business processes and microservices," in *Proceedings of the 14th International Conference on the Internet of Things*, 2024, pp. 1–9.
- [14] J. Wanner, C. Wissuchek, and C. Janiesch, "Machine learning and complex event processing: A review of real-time data analytics for the industrial internet of things," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 15, 2020.
- [15] B. R. Barricelli and S. Valtolina, "A visual language and interactive system for end-user development of internet of things ecosystems," *Journal of Visual Languages & Computing*, vol. 40, pp. 1–19, Jun. 2017.