World Wide Web (2024) 27:7
https://doi.org/10.1007/s11280-024-01237-8

Adaptive retrofitting for industrial machines: utilizing webassembly and peer-to-peer connectivity on the edge

Otoya Nakakaze[1] · István Koren[2] · Florian Brillowski[3] · Ralf Klamma[1]

Received: 30 April 2023 / Revised: 19 November 2023 / Accepted: 6 December 2023 /
Published online: 25 January 2024

Abstract
Leveraging previously untapped data sources offers significant potential for value creation in the manufacturing sector. However, asset-heavy shop floors, extended machine replacement cycles, and equipment diversity necessitate considerable investments for achieving smart manufacturing, which can be particularly challenging for small businesses. Retrofitting presents a viable solution, enabling the integration of low-cost sensors and microcontrollers with older machines to collect and transmit data. In this paper, we introduce a concept and a prototype for retrofitting industrial environments using lightweight web technologies at the edge. Our approach employs WebAssembly as a novel bytecode standard, facilitating a consistent development environment from the cloud to the edge by operating on both browsers and bare-metal hardware. By attaining near-native performance and modularity reminiscent of container-based service architectures, we demonstrate the feasibility of our approach. Our prototype was evaluated with an actual industrial robot within a showcase factory, including measurements of data exchange with a cutting-edge data lake system. We further extended the prototype to incorporate a peer-to-peer network that facilitates message routing and WebAssembly software updates. Our technology establishes a foundational framework for the transition towards Industry 4.0. By integrating considerations of sustainability and human factors, it further extends this groundwork to facilitate progression into Industry 5.0.

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2022*
Guest Editors: Richard Chbeir, Helen Huang, Yannis Manolopoulos and Fabrizio Silvestri.

✉ Otoya Nakakaze
otoya.nakakaze@rwth-aachen.de

✉ István Koren
koren@pads.rwth-aachen.de

Florian Brillowski
florian.brillowski@ita.rwth-aachen.de

[1] Chair of Databases and Information Systems, RWTH Aachen University, Ahornstraße 55, Aachen 52074, Germany

[2] Chair of Process and Data Science, RWTH Aachen University, Ahornstraße 55, Aachen 52074, Germany

[3] Institute of Textile Technology, RWTH Aachen University, Otto-Blumenthal-Straße 1, Aachen 52074, Germany

Keywords Industry 4.0 · Retrofitting · Edge computing · WebAssembly · Peer-to-Peer

1 Introduction

The digital transformation infused by the fourth industrial revolution (Industry 4.0) [1] promises huge opportunities based on new data-driven capabilities. The concept recommends interconnected information technologies such as Internet of Things (IoT) to exploit previously inaccessible data sources. Data can help companies target areas where they can improve their processes to make their manufacturing operations more efficient. In a similar vein, tracking data related to energy and water use, as well as other resources, allows companies to identify areas where they can reduce consumption, thereby enhancing sustainability. Lastly, sensors and actuators can be used to address issues of robot-human collaboration, e.g., to avoid collisions. In summary, the rapid expansion of Industry 4.0 places increasing pressure on manufacturing companies to swiftly adapt and transform their factory operations. However, in today's shop floors, long-term investments in legacy machines without networking capabilities prevail. The process of replacing an entire machine shop with new equipment is not only expensive and unsustainable, but also leads to undesired downtime. Specifically, small and medium-sized enterprises (SMEs) often face difficulties in handling the initial costs and implementation complexities associated with adopting smart manufacturing environments [2].

In contrast to the advantages of using new technology in manufacturing, the latest information technology might threaten the role of the current workforce. For instance, workers in the operation technology area are not always familiar with IT; new tools might require high-level knowledge. Whereas the primary concern of the fourth industrial revolution is digital transformation, Industry 5.0 emphasizes a human-centric approach [3–5]. The fifth industrial revolution involves the integration of cutting-edge tools with workers. Augmented, virtual, or mixed-reality technologies support people to watch and analyze the production process. Also, uncertain settings in actual physical systems can be simulated in the virtual environment. Collaborative robots assist the human force with dangerous or high-load tasks. In addition, they help us with some processes that need very high precision. Furthermore, the next industrial concept focuses on sustainability and resilience. Sustainable production targets energy efficiency and reduction of waste, such as recycling. Given the dynamic global landscape, marked by factors such as climate change, pandemics, and geopolitical shifts, there is an increasing demand for adaptable production designs to ensure resilience and continuity [6].

Retrofitting refers to the low-cost upgrade of existing equipment [7]. It allows for efficient upgrades by attaching devices, enabling rapid modernization and extending the life of machines. For instance, by monitoring the vibrations of legacy machines with cheap sensors, machine learning models are able to predict breakdowns caused by faulty parts [8]. In addition, existing production lines and know-how can continue to be used without retraining employees; retrofitting reduces the gap between the existing and the newly deployed technology, resulting in simpler worker empowerment.

In practice, it is not easy to retrofit production lines, as they consist of different control systems and electromechanical components [9]. There is currently no one-stop solution that can be deployed in a modular and uniform manner. Existing retrofitting examples are either specialized on a particular use case (e.g., [9, 10]) or too general (e.g., the commercial *LEGIC*

XDK Secure Sensor Evaluation Kit[1]); both cannot be easily fitted to custom use cases with heterogeneous machine interfaces. Web technologies, in turn, are excellent in addressing device heterogeneity. For instance, JavaScript runs on front- and backend alike.

However, JavaScript is not ideal for running on microcontrollers, as features such as dynamic typing incur a large overhead. We therefore propose the use of WebAssembly (in the following, we use the term's abbreviation Wasm interchangeably) [11]. It is a low-level language with a compact binary format that gets processed with near-native performance in a sandboxed execution environment. Although WebAssembly is a relatively new technology, it is already utilized for many use cases like serverless computing [12, 13], and resource-constrained embedded systems [14].

This paper is an extension of our research paper presented at the 23rd International Conference on Web Information Systems Engineering. We significantly extended the manuscript by adding significantly more related work, details on our implementation for NodeJS, and an entirely new perspective on peer-to-peer computing at the edge. The added peer-to-peer capabilities add functionality for data exchange and software updates over-the-air, and strengthen the reasoning for using WebAssembly.

In this paper, we present retrofitting with Wasm and investigate its capability to access machine interfaces and performing data processing tasks on the edge. Our architecture follows a state-of-the-art data lake setup, involving edge-based sensors, a cloud-based message broker, and a time-series database for long-term storage. We describe the conceptual design, demonstrate the implementation and analyze it using a laptop, a single-board computer, and a microcontroller in a real-life setting in a showcase factory.

The structure of this paper is as follows. First, Section 2 presents related work, discussing challenges of retrofitting and a technical background on Wasm. The conceptual design is subject of Section 3. Section 4 discusses the performance of our prototype. We present our approach with a peer-to-peer network in Section 5. Section 6 discusses the outlook of our work. Finally, Section 7 concludes the paper and discusses possible future research directions.

2 Related work

This section first presents our research methodology. Second, we discuss existing works related to edge computing, retrofitting, and WebAssembly.

2.1 Review methodology

In conducting this study, we investigated existing retrofitting methods and issues, as well as the current state of WebAssembly, its application, and research. The main used databases are *Scopus* and *Google Scholar*. Table 1 shows detailed terms used for searching related literature. In the *Scopus* database, we searched for titles, abstracts, and keywords defined by authors, and the search was limited to articles and conference papers. Intending to research low-cost solutions of edge computing and retrofitting or the suitability of Wasm on the limited device, we added the keyword microcontroller or constrained device to search terms. Also, some combinations include the word Industry 4.0 or 5.0. While the Scopus database returned a few highly related papers, a search with Google Scholar shows a lot of hits, but only a few of them match our purpose.

[1] cf. https://www.xdk.io/

Table 1 Used terms for search and the results

Key combinations	Scopus	Google scholar
“WebAssembly”	270	3210
“WebAssembly” AND (“retrofit*” OR “legacy”)	7	502
“WebAssembly” AND “edge computing”	23	407
”WebAssembly” AND (”microcontroller” OR ”constrained device”)	9	158
“WebAssembly” AND ”industry 4.0”	2	79
”Retrofit*” AND (”microcontroller” OR ”constrained device”)	63	2840
”Industry 4.0” AND (”retrofit*” OR ”legacy”) AND “microcontroller”	5	897
”Industry 5.0” AND (”retrofit*” OR ”legacy”)	3	834
”Edge computing” AND ”IIoT” AND ”Industry 4.0”	97	6120
”Edge computing” AND ”microcontroller*” AND ”industry 4.0”	7	1500

Since few papers related to WebAssembly and retrofitting or edge computing, we searched these keywords separately. Figure 1 shows the keywords network visualization related to WebAssemby created using VOSviewer[2], a software for bibliometric networks visualization. Retrofitting, Industry 4.0 or manufacturing are not in the network, i.e., WebAssembly has no connection with these keywords. However, at least links with “IoT” and “edge” are shown. In the screening phase, we filtered the results for each theme separately. We decided that papers dealing with low-cost solutions and the next industrial revolution have high relevance. We selected works related to Wasm with various use cases since it is a relatively new technology, and there are few high-relevance articles. In the following section, we discuss related works for each theme separately.

2.2 Edge computing

Industrial applications for data processing and control systems require latencies that distant cloud-based services cannot offer [15]. Edge computing is a paradigm for data processing that takes place at or near the edge of a network, as opposed to in a centralized data center. Commonly, it is associated with the Internet of Things, especially when it comes to sensors and actuators [16]. The edge paradigm is also suitable for real-time data analytics tasks [17], like Edge AI [18]. Consequently, an edge architecture that offers processing power in close proximity to industrial machines serves as an ideal solution for retrofitting such equipment.

Mourtzis et al. [19] have devised an edge computing platform to calculate the remaining life of industrial machines. Microcontroller units (MCU) send the data to the edge nodes and then classify the data using machine learning. After the MCU processes the necessary sensor data to distribute the computational load, the Raspberry Pi receives the data, creates a spectrogram, classifies it using a support vector machine, and then transmits it to a database in the cloud. After further analysis, the result will be applied to the digital twin. This framework also takes advantage of the ultra-low latency and increased bandwidth offered by 5G cellular networks.

[2] https://www.vosviewer.com/

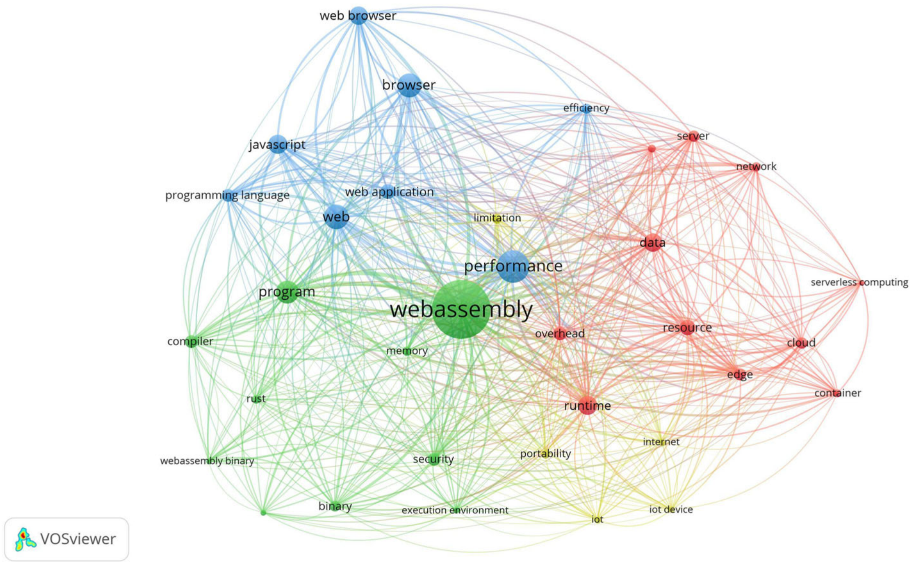

Figure 1 Search terms network visualization (WebAssembly)

Zhu et al. [20] introduce an edge computing framework managed by AI scheduling that realizes energy efficiency. Each edge node executes scheduling tasks based on a pre-learned machine-learning model from the central computer. An AI converter adjusts the AI model to heterogeneous edge devices. According to the experiment, their small testbed reduces energy consumption by 20% compared to FIFO.

Chen et al. [21] present an edge gateway consisting of microcontrollers. Their architecture distributes typical tasks of IIoT gateway to multiple MCUs due to the limited performance of tiny devices. Their approach realizes low-cost and energy-efficient management of communication in the industrial network on the edge.

These solutions promise real-time computation on the shop floor and optimization of production with data and machine learning. Also, some approach provides low-cost implementation, for example, with MCUs. However, data is not always available in factories because of legacy machines.

2.3 Retrofitting in manufacturing

Instead of buying new equipment, retrofitting can meet digitalization demands by installing cost-effective devices and sensors on existing machines. It allows for seamless component updates when technology advances and thereby extends the machine's life [7], rendering it particularly effective for SMEs [22, 23]. Sustainability including economic, environmental and social aspects is mentioned as a major reason for retrofitting by Ilari et al. [24]. The authors present a methodology to evaluate new purchase vs. smart upgrades and discuss different types of retrofitting. A review by Jaspert et al. criticizes that despite obvious reasons, sustainability aspects are largely neglected by academic literature [25]. The following sections discuss works about retrofitting methods for integrating legacy manufacturing and state-of-the-art technology and practical implementations.

2.3.1 Bridging legacy manufacturing systems with industry 4.0 technologies

The integration of new technologies with old shop floors is a significant challenge, and many factories have yet to start on digital transformation. This section reviews research on retrofitting methods for Industry 4.0.

Lins et al. [26] present a retrofitting methodology for integrating legacy machines with technology towards Industry 4.0. The technical requirements are categorized into infrastructure, communication, and application. The first category, infrastructure, includes IoT sensors attached to industrial machines and switches and servers for data transmission. The communication requirement includes integrating different communication technologies, real-time communication, etc. Also, they suggest the employment of Software Defined Networks (SDNs) to provide flexibility. In addition, the application category lists the integration of additional technologies with applications already in use, such as databases and cloud computing, and the addition of real-time monitoring. A rough standardization can be expected by using this and observing use cases with similar requirements.

Mourtzis et al. [27] present a system that uses AR (augmented reality) to assist in deciding whether retrofitting or recycling should be performed on machines in operation. Specifically, the system uses a head-mounted display (HMD) to scan the machine and suggests recycling the equipment or components used for retrofitting sold by the manufacturer in the GUI. The system facilitates digital transformation by eliminating the need for employees to research and enter machine model numbers. It also has an advantage in terms of sustainability in terms of recycling.

Guerreiro et al. [9] introduce smart retrofitting for rapid upgrades. They define short, mid, and long term retrofit phases based on the lean method, allowing for gradual upgrades. They also state that this upgrade requires employee motivation and that the use of smart devices allows them to contribute to the production process.

2.3.2 Case studies and applications of retrofitting in manufacturing

Guerreiro et al. [9] installed external embedded devices for a tool wear measurement of a drilling process in an actual manufacturing plant. The measurement results are stored in a database, and employees can get real-time data with augmented reality (AR) glasses. Mourtzis et al. [27] present an assist system for deciding retrofitting based on automatic recognition and AR, and their experimental implementation using HMD is tested with a CNC machine. Lins and Rabelo Oliveira [10] propose the standardization of retrofitting based on the RAMI 4.0 architecture for cyber-physical production systems [28]. They modernize the robot arm ED-7220C from ED Corporation by using the Linux-based single-board computer Beagle Bone Blue (BBBlue), the Open Platform Communications Unified Architecture (OPC-UA) for communication, and the programming language Python. As a result, the retrofit brought improvements in energy consumption and response time. However, the suitability for other industrial equipment is not explained in detail, and the heterogeneity of the machines is not fully considered.

Amongst commercial solutions, *LEGIC XDK Secure Sensor Evaluation Kit* is a prototyping board including various sensors and wireless network interfaces costing around €200. One of the advertised use cases is its mounting on a robotic arm to capture motion data over its built-in gyroscope, and forward it to a database in the cloud. Based on our comprehensive review of existing literature and market offerings, there is no uniform, standardized way to tackle retrofitting. Either, solutions are too specific, targeting a one-of-its-kind use case, or they are too general, merely acceptable for prototyping. We postulate that the limited availability of ready-made toolkits and software development kits also hinders the further adoption of retrofitting. In the next section, we analyze the recent WebAssembly byte code standard that is able to run on different hardware platforms and is therefore a good candidate for edge-oriented sensor data processing.

2.4 WebAssembly and its use cases

WebAssembly [29] is a portable compact binary format that processes data with near-native performance in a sandbox environment. Wasm can be compiled from many programming languages[3]. This feature allows software written in multiple high-level programming languages to run on the web. The WebAssembly Text Format (WAT), is a human-readable format available to see the compilation results. Listing 1 shows an exemplary "add" function.

Listing 1 WebAssembly text format: *Add* example [30]

```
(module
    (func (export "add") (param i32 i32) (result i32)
        local.get 0
        local.get 1
        i32.add
    )
)
```

The basic unit of code in WebAssembly, both binary and WAT, is a module [30]. Wasm's design places particular emphasis on portability and security, and Wasm binaries are developed to run in isolated environments independent of the device's system. The WebAssembly System Interface (WASI) extension allows Wasm to interact with the underlying operating system.

AssemblyScript (AS) is a version of TypeScript designed specifically for WebAssembly (Wasm) with stricter typing rules and restrictions[4]. Developers can write code with the syntax of JavaScript or TypeScript, both of which are highly popular programming languages [31]. Since AS is compiled into statically typed Wasm binaries ahead of time, dynamic typing is not usable. As Wasm runs without parsing and re-optimizing, its execution is faster than JavaScript. In particular, it is necessary to provide explicit types for function arguments and outputs. As a result, direct compilation from TypeScript or JavaScript to the low-level language is not feasible, rendering their respective libraries incompatible.

Initially, WebAssembly was developed to cater to web browser applications that demanded high computational power, such as advanced gaming or multimedia processing. However, its usage has since expanded to encompass non-browser applications, including serverless computing and the Internet of Things. Due to its memory and execution safety guarantees, WebAssembly is inherently tailored to short-lived functions for serverless applications. For instance, Hall and Ramachandran present a runtime [12], but the framework does not integrate dynamic deployment capabilities. *Sledge* is an optimized runtime for serverless functions running with WebAssembly [32]. A performance benchmark is presented by Mendki, measuring faster startup times compared to service containers, but slower speed than native applications [13]. Similarly, Napieralla [33] investigates performance for edge computing. The WASI-based virtualization has an advantage in startup time and size; in contrast, it

[3] cf. https://github.com/appcypher/awesome-wasm-langs

[4] https://www.assemblyscript.org/

suffers from overhead during operation. To this end, *Wasmachine* is a WebAssembly operating system with ahead-of-time compilation to native binary that speeds up the execution of commonly-used IoT and fog applications by up to 11% compared to Linux [34]. Other work focuses on binary code analysis for security and language analysis optimizations [35, 36]. A number of commercial vendors now offer edge-oriented runtimes for WebAssembly, e.g., *Cloudflare*, *Fastly* and *wasmCloud*.

Developers have successfully utilized WebAssembly on wearable devices, such as a pulse sensor [14]. Mäkitalo et al. discuss WebAssembly as enabler for liquid Internet of Things applications [37]. Their runtime allows live code migration within IoT settings. The *WiProg* approach has similar goals and uses ESP32 devices and the wasm3 library, like our prototype [38]. A list of projects using WebAssembly is compiled on the *Made with WebAssembly* website[5]. For instance, it contains use cases like CAD, PDF viewers and whole database implementations. The *Awesome Wasm* repository available on GitHub[6] lists a number of runtimes and embeddings outside the browser. However, we are not aware of any related work that employs WebAssembly with industrial machines. Along with the pressure on companies to align their production with sustainability goals, we see enormous potential for this technology. It can offer a way to extend asset-heavy industrial machines using familiar programming languages and tools, to share code between machines in an agile way.

3 WebAssembly-based collision detection system

In this section, we present our conceptual design and prototype for retrofitting legacy industrial machines using Wasm. We begin by providing an overview of our industrial showcase, the *human-robot collision detection* system, and subsequently derive the requirements for our design. For instance, the communication between the device and a machine uses RS-232 as the most common serial interface amongst industrial equipment (alongside RS-422 and RS-485). For our edge device, an RS-232 port or a corresponding converter is necessary. In addition, a runtime for the Wasm module is mandatory on the edge hardware.

3.1 Showcase: industrial robot collision detection

Modern small robotic arms, also known as Cobots, are equipped with sensors for collision detection and safety features, including the ability to stop upon contact with the human body. In contrast, such safety measures are not typically present in larger robots. Attaching communication devices, sensors, and cameras to such robots allows to collect data needed to calculate the real-time distance between a person and the robot. To measure the real-world applicability of our concept and prototype, we selected the use case of a robotic arm within a demonstration factory at the institute for textile engineering at our university. Figure 2 shows the workshop, a KUKA KR-150-2 robot arm [39] and a KR C2 controller. The robot's movement area is five meters wide and four meters high, as it assists humans in moving textile material between preform table and CNC-cutter. It has six axes and is designed for a maximum load of 150 kg. The stretched arm's length is 2700 mm. KR C2 runs a program written in KUKA Robot Language to get and send axis values in byte sequences over an RS-232 9-Pin D-Sub serial port. The controller has a submit interpreter to perform other

[5] cf. https://madewithwebassembly.com/

[6] cf. https://github.com/mbasso/awesome-wasm

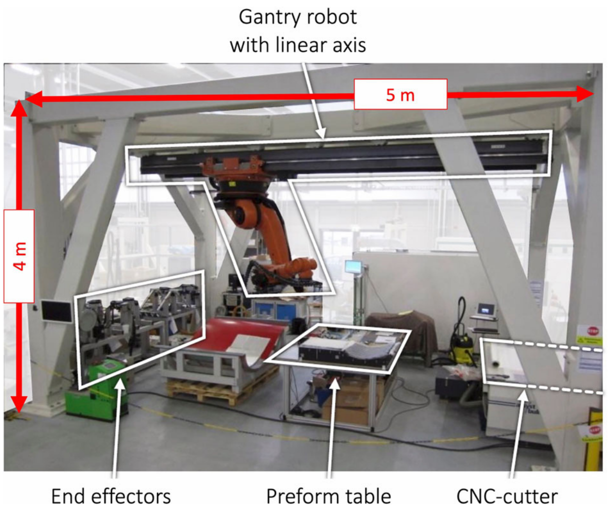

Figure 2 Preform workshop with industrial robot

tasks in parallel with the main task. The device receives values of axes as byte sequences and converts them to numeric types.

In the current setup, an anti-collision system for the robot KUKA KR 150-2 is running on a standalone PC developed in Java. The system uses infrared and time-of-flight cameras to locate the person's position and then uses the axis data sent from the robot's controller to calculate the distance. As the robot continues to move between transmitting data and receiving a stop command, a safety distance must be maintained from workers. This distance, which is proportional to the robot's speed, depends on communication latency and calculation speed. If the current minimal space drops below this distance, the system halts the robot. Latency is one of the primary challenges associated with this implementation. It needs an impractical long safety distance because of the time of the data transmission. The length depends on the movement speed of the robot; it is about $3.6\,m$ for $1\,m/s$. As humans and robots work in shifts, collaboration is impossible with this latency. The used robot is a typical legacy machine that requires serial communication for sending data. For these reasons, this use case fits to realize and demonstrate the retrofitting using WebAssembly. We implement the transmission of axis data of the KUKA robot using an inexpensive device and calculate the distance between the arm and an object for collision prevention in the same environment.

3.2 Requirements for retrofitting with edge devices

Based on the general motivation of data-intensive Industry 4.0 use cases, and the capabilities needed for retrofitting, we derive the following requirements:

Low-level hardware access Many aged industrial Programmable Logic Controllers (PLCs) provide at least a serial communication interface. Therefore, any retrofitting framework must be able to interface with it.

Networking interface In order to be able to forward the acquired sensor data, (wireless) communication interfaces must be available.

Agile updates The fast-changing smart manufacturing landscape requires rapid update cycles resulting in, e.g., software improvements and bugfixing.

Standards-based solution Our foremost requirement is a lightweight approach that results in a more user-friendly, versatile, and cohesive Internet of Things. This excludes proprietary runtimes and libraries.

Our design should be capable of standardizing the serial data format before forwarding. This conversion is required as one of the tasks to be performed by the device. Finally, the processed data is stored in a data lake, a data repository in which raw data from sources are stored without structuring [40]. However, network communication is power- and resource-consuming and can be a heavy burden for less powerful devices. Therefore, the communication overhead must be as low as possible. The program for the device must be lightweight enough to run in a restricted environment.

As shown in the previous section, industrial machines can be retrofitted by proprietary embedded systems, but these do not offer the demanded flexibility. To realize agile changes in manufacturing, we intend to build on the strength of web technologies regarding heterogeneous device access. To this end, we consider WebAssembly an effective means for the computation of edge devices due to its size and performance.

3.3 Conceptual architecture

Figure 3 provides an overview of the process involved in retrofitting an industrial robot, which can be broken down into three distinct phases: data reading via serial communication, data processing, and data transmission. Generally, industrial machinery employs Programmable Logic Controllers (PLCs). The controller also manages data transmission and reception; it reads the data from the machine and sends it to the device through the serial port. Thus, our device communicates with the PLC. Usually, the controller's task is implemented using a controller-specific programming language. The device connected to the serial port monitors its interface and detects data arrival. The received data is filtered, structured, and computed, before the results are sent via MQTT. To handle the influx of data from multiple machines, our design employs specialized software on the data lake side that enables data stream ingestion.

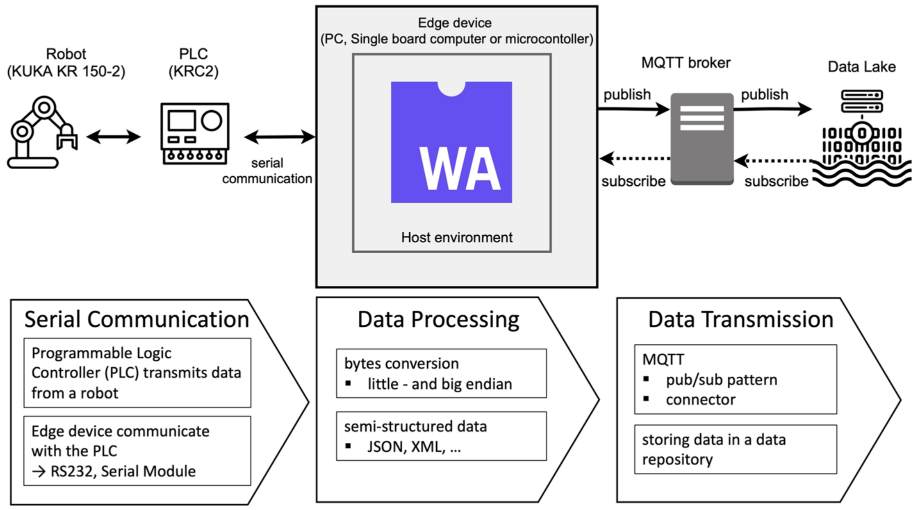

Figure 3 An overview of retrofitting an industrial robot

3.4 Realization

We implemented prototypes for single-board computers and microcontrollers as open source software[7]. As single-board computer, we use a Raspberry Pi 4 with Raspberry Pi OS, a 1.5GHz quad-core CPU, and 8GB of RAM (around €30). The used microcontroller is an ESP32, often used for IoT projects (around €4). Despite its low cost, it has a dual-core CPU, 512KB of RAM, and 4MB of flash memory. WASI does not yet support I/O and network sockets, we thus decided to bypass it by using a Wasm runtime that provides respective interfaces. Our prototype for Raspberry Pi uses a NodeJS runtime. The restricted microcontroller is insufficient for a full JavaScript engine, the binary code therefore runs with the wasm3 interpreter[8].

3.4.1 Single-board computer host

Figure 4 shows the component diagram of the single-board computer host implementation. To run the Wasm module on NodeJS, it needs to be instantiated. The *AssemblyScript Loader* based on *Wasm JavaScript API*[9] performs instantiation and memory operations. The exchange of high-level data types such as strings between JavaScript and Wasm uses a shared linear memory. Our prototype uses the *as-bind* library[10] to simplify the exchange of high-level data types. A `SerialPort`[11] instance is used to access the USB port. Our implementation, which leverages AssemblyScript for data processing, performs several functions, including data conversion from bytes to numbers, detecting error values, and generating JSON data. Each time the serial communication event listener detects new data, it calls a method in the Wasm module. The implementation of sending data uses MQTT.js[12].

3.4.2 Microcontroller host

We wrote the runtime host for the microcontroller in the Arduino programming language. The implementation uses wasm3, a library of the WebAssembly interpreter for Arduino. Regarding their tasks, there is no difference to the single-board host. However, the Wasm file for a single-board computer is not runnable on a microcontroller because the AssemblyScript Loader targets only the NodeJS environment. Besides, supporting the AS standard library by wasm3 is not complete; for example, two-dimensional arrays declared like `Array<Array<f64>>` cause an allocation out of the defined memory. As a result, we utilized alternative methods for distance calculation and JSON conversion that are better suited to our specific use case. In future versions of the library, this distinction will likely change and enable fully isomorphic code [41] between all hosts.

`Serial`, included in the Arduino programming language standard library is used to communicate with the controller. If data arrives, it calls the Wasm module with the corresponding call function. The conversion of JSON data uses methods written in the Arduino programming language because the conversion targeting string and the concatenation of string in AS

[7] cf. https://github.com/internet-of-production/WasmRetrofittingESP32

[8] https://github.com/wasm3

[9] https://www.w3.org/TR/wasm-js-api-2/

[10] https://github.com/torch2424/as-bind

[11] https://serialport.io/

[12] https://github.com/mqttjs/MQTT.js

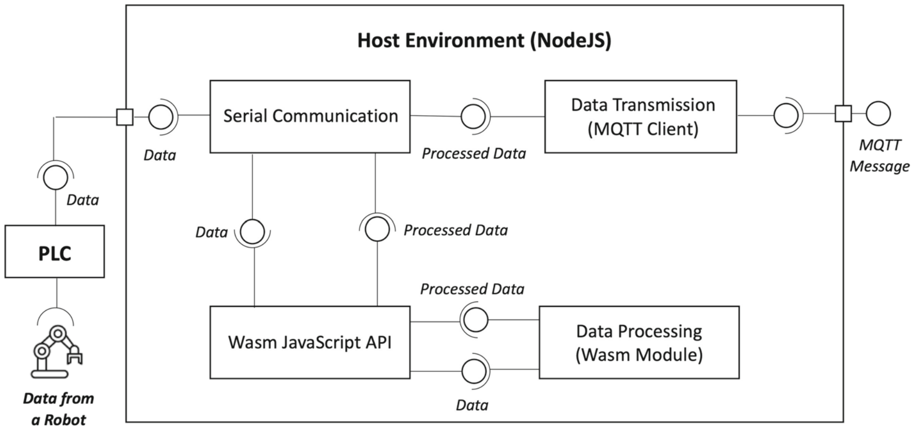

Figure 4 Components of the edge device implementation for the single-board computer

are not fully supported by the interpreter yet. Once the byte data has been converted, Wasm passes the keys and values for the JSON to the Arduino. This is achieved by utilizing pointers and invoking a C++ method that converts a string of fixed-length UTF-16 to UTF-8. By utilizing this approach, a JSON data string that incorporates settings such as SSID and broker IP for WiFi and MQTT connections is transferred from Wasm to Arduino. Instead of `Array` types, we use `load` and `store` functions. WiFiClientSecure, a library of the Arduino core for the ESP32, is used for the network connection using TLS, and PubSubClient[13] is used for MQTT communication.

3.4.3 Data stream ingestion

Figure 5 shows our data stream setup. The edge device sends processed data to the MQTT broker; our implementation uses the open-source *Eclipse Mosquitto*. As it is only utilized as a middleware between the client and server, a data stream ingestion mechanism is required in the cloud to handle the anticipated high volume of data. We use an Apache Kafka endpoint, which is a popular open-source distributed messaging system[14]. It is used for handling high throughput, low latency messaging. In order to receive data from Mosquitto, Kafka must subscribe to a specific topic. Therefore, we connected it to the open-source *Fluentd*[15] as data collector. Fluentd has high throughput and low resource consumption, and more than 500 community-contributed plugins are available. The plugins used here are fluent-plugin-kafka[16] and Fluent::Plugin::Mqtt::IO [17]. We deployed InfluxDB[18] as time-series database in the data lake. Fluentd also connects Kafka and the data lake with the plugin influxdb-plugin-fluent[19]. Furthermore, our prototype uses Docker containers and Kubernetes, an open-source container orchestration; Kafka, InfluxDB, and Fluentd are deployed in our cloud cluster.

[13] https://pubsubclient.knolleary.net/
[14] https://kafka.apache.org/
[15] https://www.fluentd.org/
[16] https://github.com/fluent/fluent-plugin-kafka
[17] https://github.com/toyokazu/fluent-plugin-mqtt-io
[18] https://www.influxdata.com/
[19] https://github.com/influxdata/influxdb-plugin-fluent

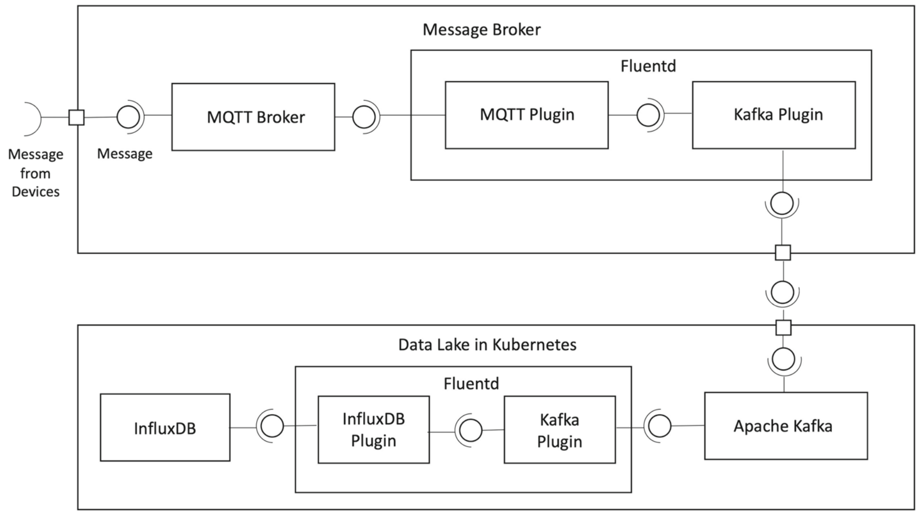

Figure 5 Data stream ingestion setup

4 Evaluation of WebAssembly on the edge

This section evaluates our implementation in a production setting. First, we measured the performance of the distance calculation and the required program size. In addition, we tested the round-trip time (RTT) between the edge device and the data lake to evaluate the feasibility of our edge scenario. Finally, we discuss the reusability and limitations of our prototype.

4.1 Performance of the distance calculation use case

We implemented the distance calculation between a point in space and the robot arm's end as a base for collision prevention. This calculation is done with a product of 4x4 matrices according to homogeneous transformation [42, 43]. We evaluated the application performance of our system on three different hardware platforms with varying computation power: a laptop, a Raspberry Pi, and an ESP32 microcontroller. The laptop, with sufficient computing power, was used to run the single-board computer implementation, while the Raspberry Pi provided a slightly restricted environment. The ESP32, as a strictly limited device, was also used for evaluation. The used laptop was a MacBook Pro 2020 with macOS Big Sur version 11.2.1, a 1.4 GHz Quad-Core processor, and 16 GB of RAM. The KRC2 sends each axis data numbered one to six and external one separately but in a fixed order every loop. Since all the axis data is needed to calculate the spatial coordinates of the robot, it checks the number of saved items after every reading and conversion cycle. In this computation, we used a simulation of previously recorded data that we replayed via serial communication. The data were randomly generated within the specified value range and therefore contained no errors; ESP32 sent bytes at the measurement on the laptop and the Raspberry Pi 4, and a JavaScript program transmitted values through the USB port for the microcontroller. The laptop executed the single-board computer implementation. We sent the machine data 100 times for each device. It did not perform JSON conversion and transmission of the results during these measurements. Table 2 shows the measurements.

Table 2 Time measurements of distance calculation on three device types

Device	Messages #	Average (*ms*)	First Call (Max.) (*ms*)
Laptop	100	0.230	1.451
RasPi	100	1.060	5.800
ESP32	100	3.093	9.576

On average, the laptop was the fastest at 0.23 *ms*, the Raspberry Pi 4 at 1.06 *ms*, and the ESP32 consumed about three times the average processing time of Raspberry Pi. The maximum processing time is due to the first call of the function. NodeJS, an execution environment built on V8 that is a JS engine that performs JIT compilation, and wasm3 decode and compile Wasm when calling the function. These runtimes have a disadvantage in overhead at calling functions, but Wasm does not need the optimization by a JIT compiler.

In the implementation of our previous work discussed in Section 3.1, 1 *ms* is required in proportion to the number of spheres to reflect the axis values to the robot model. The time required to calculate the distance between the robot's end and a fixed point in space using the Raspberry Pi in this paper is 1.06 *ms* on average. Assuming the calculation is in proportion to the number of coordinate points set on robot parts, the same level of processing speed is achievable by using Wasm.

4.2 Storage utilization of our program

The single-board computer host requires NodeJS, which takes about 100 MB to install, and the Wasm file is 22 KB in size, including both JSON conversion and coordinate calculation. The main JS file is 4 KB, but the dependencies require a large 26.6 MB. The implementation for microcontrollers uses about 800KB of ESP32 external flash. The size of the written program is about 750 KB, and the global variables are about 50 KB. Since 320 KB is reserved for storing variables, the remaining flash memory of approximately 2.9 MB is available for RAM and the file system.

4.3 Latency between edge and data lake

We measured the round-trip time between the device connected to the KRC2 and Apache Kafka running in the data lake to confirm the need for edge computing in the highly latency-sensitive collision use case. A cable connected to the RS-232 serial port, further extended by a USB converter cable, linked the KRC2 and the edge device. Sending messages from the device to a local laptop used MQTTS. HTTPS was used between the computer and the data lake. Each device subscribed to a specific topic to record the arrival time of the response.

First, we executed the single-board computer host implementation on the local laptop to perform the measurements. In this case, Eclipse Mosquitto and Fluentd ran on the same platform. Second, we measured the RTT with a Raspberry Pi 4. The MQTT messages were sent to the broker on the laptop through the Ethernet cable. The on-site download speed was 82 Mbps, and the upload speed was 76 Mbps at the time of measurement, but the rate was not always stable. The average RTT was about 60 *ms* for both edge devices. In the Java implementation of our previous work, all collision prevention processes were done at the machine, and the RTT from the data transmission to the stop operation were around 50 *ms* in total. Therefore, an improvement of the response time with calculating distance at the centralized data lake is impossible.

4.4 Reusability and limitations of our prototype

The implementation based on NodeJS uses only JavaScript and AssemblyScript. AS also uses the Node project environment with common tools such as the Node package manager (`npm`). In particular, as-bind enables the exchange of high-level data types between the two languages, thus simplifying programming. We realized JSON conversion and distance calculation using only AssemblyScript. Therefore, if the same execution engine is available, the Wasm file is reusable. Furthermore, when calling new external functions in JavaScript, it is possible to import them by simply using modules of Wasm instances. The AS library provides a developer-friendly coding environment, and using it can simplify and improve the readability of the source code, but it also creates dependencies. The implementation for microcomputers using wasm3 does not include dependencies on such language-specific libraries. However, the interpreter is still under development and does not fully support all the possibilities of AS. For example, to use two-dimensional arrays for matrix calculation, developers must manage linear memory using store and load functions due to opcode detection issues.

Our prototype shares common limitations of WebAssembly. For example, Wasm does currently not allow hardware-specific features like GPU-based matrix multiplication [12]. Although ESP32 has a dual-core processor, the runtime does not support threads. Wasm uses linear memory, but unmapped pages are not available, so any read or write within the allocated space will succeed; an attacker does not need to consider page faults at the memory access [44].

The results of our evaluation, especially with regard to latency to the data lake, must be taken with a grain of salt. Collision detection frameworks using machine learning will inevitably have different requirements, which in turn will lead to adapted recommendations for placement at the edge or in the cloud. For instance, we did not perform long-term operations over several weeks and in changing environmental conditions like hot temperatures, which may lead to overheating issues with our microcontroller hardware. Our showcase study clearly focuses on the KUKA industrial robot. However, its communication architecture with an attached PLC is a well-established pattern in machine shops. We are therefore confident that results are transferable to other production equipment; we are currently targeting further use cases within the workshops of our university and associated industry partners.

5 Retrofitting with easy update of Wasm in P2P networks

As mentioned in Section 3.2, agile updates are crucial for retrofitting legacy industrial equipment and achieving smart manufacturing. The use of Wasm on edge devices enables easy and fast task updates without the need to upload the full binary generated from an embedded programming language like C [45]. However, the implementation presented above did not provide an easy method for updating the Wasm module. While users can generate new Wasm files from various programming languages, uploading the latest change to our prototype requires familiarity with tools such as *NodeJS* and *Platform IO*. In addition, Section 4.1 shows that ESP32 suffers from limited resources and has lower throughput than Raspberry Pi, so a solution is to be considered. On the one hand, enterprises can build networks only with more powerful devices like single-board computers in their own factory. However, despite their higher computational power, devices like the Raspberry Pi are often not the most efficient choice for specific industrial applications due to their larger size and higher power consumption compared to microcontroller units (MCUs) like the ESP32. Furthermore, the

ESP32 is specifically designed for IoT applications, offering a better balance of performance, power efficiency, and physical footprint for tasks that do not require the full capabilities of a single-board computer. This makes it an ideal choice for cost-effective and energy-efficient retrofitting in industrial environments. Therefore, prioritizing the use of ESP32 over more resource-intensive devices aligns with the goals of sustainable and efficient manufacturing. Moreover, our solution presented in Section 4 covers only communication with industrial machines and the MQTT broker, but not with other edge devices. Peer-to-peer (P2P) networks of microcontrollers realize fast information exchange between industrial robots; they promise a real-time update of processes in manufacturing. For example, Carnevale et al. discuss reconfiguration of MCUs over the air in mesh networks [46]. They realized firmware updates in networks consisting of heterogeneous devices, ESP32 and Raspberry Pi. This section therefore presents an extension of our system to allow for retrofitting with Wasm, while communicating data streams and software updates over a peer-to-peer network.

5.1 Extension requirements

In order to realize more flexible and usable retrofitting with Wasm, we derive the following requirements.

- **Easy update of Wasm:** The program introduced in Section 3 forces users to use and learn specific tools before modifying the Wasm module. A more user-friendly way is desired instead.
- **Communication between edge devices:** Microcontrollers for retrofitting should be able to communicate with each other in smart manufacturing.
- **Performance improvements:** Microcontrollers suffer from restricted resources. Therefore, a way to reduce the load, such as task distribution, is required. It is also important for the cost reduction of retrofitting.

In the following section, we discuss solutions to achieve each requirement.

5.2 Updating the wasm module

In an ideal scenario, system users should be able to modify Wasm modules using their preferred language, without the need for additional skills or knowledge of specific software. However, our current design does not address this issue. For instance, our implementation for the ESP32 platform requires users to have a basic understanding of the *Platform IO* toolchain to upload new Wasm files to the microcontroller. This initial process can be time-consuming, requiring users to read documentation, install software and plugins, configure the environment, and upload new files. Such tasks may be complicated by unexpected errors, such as configuration mistakes. To address this issue, we propose a more straightforward approach to replace the Wasm binary. In the next section, we present our proposed solution, which allows users to update Wasm modules more efficiently, without requiring any specific software or tools.

Figure 6 illustrates two possible methods for updating a Wasm module on the microcontroller. While uploading files through the serial interface requires specific software and cables, wireless communication is more convenient. ESP32 comes with WiFi and Bluetooth Low Energy (BLE) chips, enabling two possible approaches for uploading the binary: an embedded web server on the ESP32, and uploading the file via BLE. The first method involves connecting a device like a PC to a WiFi access point created by the microcontroller to access

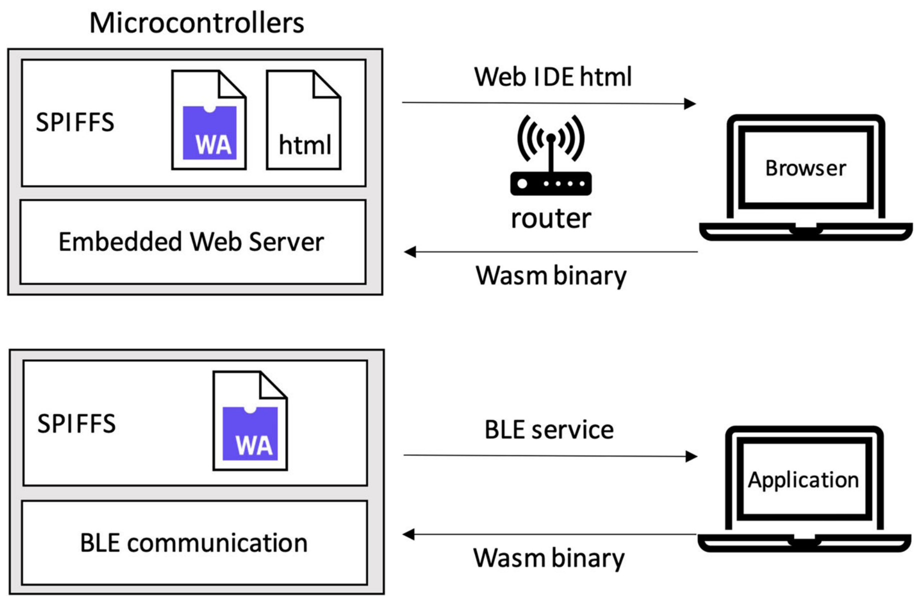

Figure 6 Two possibilities for over-the-air updates using a Web IDE and BLE

a user interface with a web browser [47]. While this web application does not require the installation of specific software, the source code files are stored in the flash memory. The second approach separates the functionality for modifying the Wasm binary from the devices, but BLE communication is necessary to send the generated binary. BLE is a wireless communication protocol designed for low-power devices, such as sensors and wearables. It is a subset of Bluetooth technology and is used for short-range communication between devices. BLE consumes less power than traditional Bluetooth and is therefore ideal for IoT devices that require long battery life. It has a range of up to 30 meters and can transfer data at rates of up to 1 Mbps. BLE has a maximum packet length of 512 bytes, which means that the sender must fragment large data, and a packet management system is necessary.

5.3 P2P and mesh networks

P2P networks facilitate direct communication between edge devices and can potentially resolve latency issues. This section discusses simple wireless P2P networks and mesh networks. Since this work uses Raspberry Pi and ESP32, we focus on communication over WiFi and Bluetooth, which are available on these boards. A simple P2P network can be implemented without deep knowledge, for example, by using libraries. ESP-Now is a wireless communication protocol over WiFi for ESP32, which can be used with an Arduino library[20]. However, it still requires initial setup by adding a new device into the group; manual MAC or IP address adjustment may be needed. Mesh networks can reduce these setup costs. A new node only needs specific information, like a token shared in the community, to participate in the mesh, and new participants will be added by other nodes automatically. Moreover, the topology is fixed in a simple P2P network, whereas it can be dynamically optimized in a

[20] https://www.espressif.com/en/products/software/esp-now/overview

mesh network. However, managing a mesh network can be costly, and the added functionality might slow down other processes. Additionally, WiFi mesh communication may occupy WiFi connectivity. After reading data from machines, attached devices transmit the result to other components like an MQTT broker, but both Raspberry Pi and ESP32 have only one WiFi module by default. Therefore, data should be sent in different ways, such as Bluetooth or serial interface. BLE mesh can resolve this problem because the WiFi module is available for other purposes. However, BLE has a strictly limited packet size. ESP-BLE-MESH[21] allows only 23 bytes, including header information.

5.4 Flexible task assignment with wasm in P2P networks

In production lines, sensors are responsible for specific tasks, but the load is not uniformly distributed among them. The central node in a star topology often receives more messages than other child nodes, and edge nodes with lower loads should take over data processing tasks. However, typical embedded device programs are dependent on hardware architecture, and reassigning jobs between heterogeneous devices is challenging. Our design with Wasm enables real-time load management between sensor nodes, thanks to its portability feature. In order to monitor and analyze the entire system, we need to obtain the status information of all nodes. Therefore, an access point that collects these data is required, but this central node may not be reachable from all other sensors due to distance. By using a P2P network, edge devices do not have to be located near the central node because the neighbor can forward the message to the following node. Based on the analysis of data, tasks can be reassigned in the network. For example, a web application displays a graph of the current data stream, and users can manually change the destination of data or move the Wasm module from one node to another so that a node with a low load can process it.

5.5 Experimental setup

In previous sections, we presented possible solutions for updating Wasm, P2P networks, and load management between microcontrollers. Based on these techniques, we implemented an experimental setup for retrofitting. Figure 7 shows an overview of our system.

The employed ESP32 has the same specification as the prototype in Section 3. This implementation covers the following functionality with respect to the new requirements.

- Updating Wasm with a web application through BLE and WiFi-mesh
- Communication between nodes via WiFi-mesh
- Visualizing and modifying the data stream network
- Moving Wasm binary between nodes

To establish communication between the edge devices, we explored the possibility of using a mesh network, specifically using ESP32s to build a WiFi-mesh network with ESP-WiFi-Mesh. Since the packet size in BLE-mesh is limited, we opted for WiFi to transmit the Wasm binary between the devices. To enable access to the data stream, a BLE service is provided via an access point. However, due to memory constraints, it is not possible for ESP32 to simultaneously run the mesh, BLE, and Wasm functionalities. To address this limitation, we implemented three types of nodes: one for mesh, one for BLE service, and one for Wasm functionalities, rather than deploying all modules on a single board.

[21] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp-ble-mesh.html

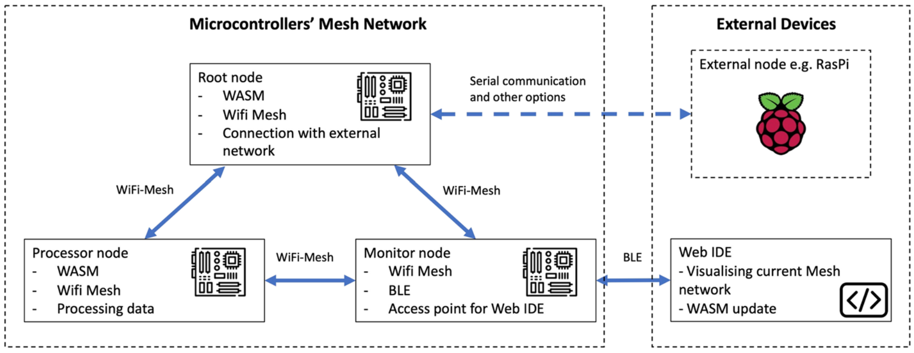

Figure 7 The retrofitting system using WiFi-mesh network and Wasm update via BLE

- **Root node** provides connection with external networks/devices. For example, this node gathers data from other mesh-network's members and transmits to an external device.
- **Data processor node** executes Wasm module that processes logged data, e.g., filtering, and sampling.
- **Monitor node** has an access point via BLE (bluetooth low energy). A user can access this node with browser, get a view of current network, write new code, and update Wasm on an arbitrary node in the network.

Only the root node is connected to an external router, allowing it to communicate with other networks via WiFi. Other options for communication include serial communication with an external device. The web application allows users to view a data stream graph and provides a code area for generating new Wasm binaries. Users access the monitor node via web Bluetooth by scanning for the BLE service with a UUID. After a successful connection, the application requests the latest status of the data stream and displays a graph based on the achieved routing table. Any changes made to the graph, such as adding links, will be applied to the actual mesh network. Users can choose a target node by clicking and upload a new Wasm binary. Additionally, adding a new microcontroller to the network is a simple process that only requires the MAC address of the root node to be added. However, in the following section, we will discuss the limitations of this extension and possible improvements.

5.6 Limitations and possible extensions

Our extension provides various advantages, such as easy updates of the Wasm binary, quick deployment of new edge devices in the mesh network, and user-friendly modification of the data stream. However, there are still some limitations and challenges to be addressed. Firstly, the ESP-WiFi-Mesh protocol we use is specific to ESP devices and cannot be applied to non-ESP devices. Furthermore, WiFi communication can be unstable due to resource capacity issues, especially when the root node connects with an external network. To address this, adding a second WiFi board or using BLE communication can help, but it also makes the design more complex. Secondly, the initial setup of our framework is not straightforward, as detailed configurations are needed to enable WiFi and BLE coexistence. Moreover, developers need to use specific tools to modify message types, as ESPs transmit messages with a specific number indicating the content. Additionally, management for packet fragmentation and defragmentation is required due to size limitations of BLE and WiFi. Despite these challenges, our extension with WiFi-mesh and BLE can potentially facilitate task distribution and process optimization in the microcontroller's network with Wasm.

6 Outlook on retrofitting with WebAssembly

This section presents potential applications of our implementation for manufacturing and discusses challenges. In the previous section, we mentioned load balancing in a mesh network to address the limited computing power. As a concrete solution, the application for the network's status visualization and task reassignment is presented in Section 5.4. The concept of the next industrial revolution, especially Industry 5.0, recommends the employment of digital twins, digitally represented physical systems [3, 4]. For example, a digital twin application displays a 3D model of a real shop floor in which the production status is synchronized in real-time. This virtual environment provides more accurate prediction and simulation. Therefore, it realizes more efficient and optimized manufacturing; for example, digital twins can prevent serious faults in the physical setting and reduce financial losses [6]. However, the software that makes this possible must transmit data to indicate the status of robots. In other words, it requires a large upgrade to use a digital twin in an old factory. Although retrofitting can expand communication and other functions that help monitor and analyze the current status, simulating the binary code compiled from a traditional language like C for the microcontroller on the target software would further complicate the system. By taking advantage of Wasm's portability, the binary code running on the simulation can be used as is and vice versa. In this way, workers can deploy specific components on the software more simply. For example, it might enable receiving Wasm binaries from an additional microcontroller, simulating them in the virtual environment, optimizing the task, and then uploading them again.

However, there are various challenges in implementing this design. One problem is that Wasm runs in a sandbox environment and does not have direct access to the machine interface like I/O ports. With the aim of providing access, necessary functions must be defined in advance in a corresponding language depending on the runtime environment. Although different developers use various technologies, the definition of the functions must be the same because one must specify the required interface before generating Wasm binaries; for instance, in wasm3, the name, number of parameters, type, etc., are required to pass functions in the specified language to Wasm. These must be standardized, but WASI does not currently support functions such as I/O port access. Although it is possible to use libraries such as WASIX[22], this is not a standard, and there is concern that it will complicate the production line.

Furthermore, our prototype has challenges in communication. Our implementation uses ESP's specific protocol, ESP-WIFI-Mesh, which does not allow devices with other architectures, such as Raspberry Pi, to join the network. Therefore, a bridge is necessary that connects different networks consisting of various devices. For example, a communication management point like an MQTT broker must be set up to forward information from the ESP mesh network to other devices or networks. Wasm may realize quick configuration for communication between these disparate machines. Currently, WASI does not support HTTP communication or some of the socket features, but WASI reports[23] that many developers would like to see these features added. If Socket functionality is fully supported, it would be

[22] https://wasix.org/

[23] https://www.cncf.io/reports/the-state-of-webassembly-2023/

possible to implement communication protocols in Wasm, and device-independent binary files would help upgrade the network.

According to the results in Section 4, the ESP32 is unsuitable for complex calculations such as those involving matrices in real-time when using Wasm. Nevertheless, in order to keep costs down, we can assign simple tasks such as reading, converting, and transmitting data from the machine interface to the microcontroller, and complex calculations can be left to more powerful devices such as the Raspberry Pi. It is also necessary to determine which device to use based on the computing power, response time, and communication distance that meets the requirements for the intended use.

Also, one must consider security terms. BLE saves power resources, but its security level is lower compared with other standard protocols like HTTPS and Bluetooth. Therefore, attackers might target this as the entry point. For example, Lacava et al. [48] report BLE 4.0 does not prevent a man-in-the-middle attack if the *Just Works* mode is used in the pairing method. They also mention the potential threat of energy exhaustion and DoS attacks targeting IoT devices. Besides, BLE 5.0 has a vulnerability to key negotiation of Bluetooth (KNOB) attacks. In addition, the security of mesh networks must be considered. For example, ESP-WiFi-Mesh uses WPA2, which can be intercepted by key reinstallation attacks (KRACK) to decrypt encrypted messages [49]. In addition, one should be careful about updates after retrofitting. Newly installed devices may have the latest security measures. However, old gadgets and machines may still be using outdated protocols and encryption methods, which could be used as an intrusion route. Since the latest version may also have a security hole, one should consider adding another security system; for instance, several papers propose intrusion detection systems using machine learning for BLE networks [50, 51].

7 Conclusions

In this paper, we presented a conceptual design and prototypical implementation of retrofitting industrial machines on the edge using WebAssembly. To the best of our knowledge, it is the first open implementation using WebAssembly to make data available directly from proprietary, legacy industrial machines that do not have their own network connection. Our prototype uses AssemblyScript and the associated binding generator to pass high-level structures between the runtime and the Wasm module. Because certain features of AssemblyScript are not supported by the wasm3 interpreter, we added the necessary bindings with function calls written in C++ and Arduino. By comparing devices with distance calculation times, we found that the first function call caused overhead on all of them due to the interpreter and the JIT compiler, which an ahead-of-time compiler could improve [34]. Running Wasm on the Raspberry Pi showed that the byte code has near-native performance for the distance calculation use case.

Furthermore, we demonstrated how to update Wasm binaries on microcontrollers in mesh networks using a user-friendly way to upload new binaries over wireless networks. While our experimental implementation suffers from the limited resources and specific WiFi-mesh protocol, it shows the potential of using peer-to-peer networks in managing load balancing in edge devices with Wasm.

As future work, we plan to extend our showcase to other industrial machines within our demonstration factory. In addition to that, evaluation for more complex calculations and longer operation time is needed. The WASI standard will likely become the backbone of our implementation work, once socket connections are available. We are confident that

WebAssembly will become a core component in data processing pipelines, as it can target any place in the edge-cloud continuum. In the long run, Wasm can become the catalyst in achieving a uniform development and deployment environment for Industry 4.0 and beyond.

Author Contributions Otoya Nakakaze wrote the main manuscript text with significant contributions and editing by István Koren. Florian Brillowski provided the use case environment and verified the contextual correctness. Ralf Klamma supervised the research and edited the original conference publication.

Funding Open Access funding enabled and organized by Projekt DEAL. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2023 Internet of Production - 390621612.

Availability of data and materials The source code of our prototype is available via https://github.com/internet-of-production.

Declarations

Competing Interests The authors declare that they have no conflict of interest.

Ethical Approval Not applicable.

References

1. Kargermann, H., Wahlster, W., Helbig, J.: Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Final report of the Industrie 4.0 Working Group. https://en.acatech.de/wp-content/uploads/sites/6/2018/03/Final_report__Industrie_4.0_accessible.pdf, Berlin (2013)
2. Masood, T., Sonntag, P.: Industry 4.0: Adoption challenges and benefits for SMEs. Computers in Industry. **121**, 103261 (2020) https://doi.org/10.1016/j.compind.2020.103261
3. Müller, J.: Enabling technologies for industry 5.0. Technical report (2020)
4. Directorate-General for Research and Innovation (European Commission), Breque, M., De Nul, L., Petridis, A.: Industry 5.0: Towards a Sustainable, Human Centric and Resilient European Industry. Publications Office of the European Union, Luxembourg (2021)
5. Nahavandi, S.: Industry 5.0-a human-centric solution. Sustainability (Switzerland). **11**(16), (2019) https://doi.org/10.3390/su11164371
6. Maddikunta, P.K.R., Pham, Q.-V., B, P., Deepa, N., Dev, K., Gadekallu, T.R., Ruby, R., Liyanage, M.: Industry 5.0: a survey on enabling technologies and potential applications. Journal of Industrial Information Integration. **26**, 100257 (2022) https://doi.org/10.1016/j.jii.2021.100257
7. Stock, T., Seliger, G.: Opportunities of sustainable manufacturing in industry 4.0. Procedia CIRP. **40**, 536–541 (2016) https://doi.org/10.1016/j.procir.2016.01.129
8. Carvalho, T.P., Soares, F., Vita, R., Da Francisco, R.P., Basto, J.P., Alcalá, S.G.S.: A systematic literature review of machine learning methods applied to predictive maintenance. Comput. & Ind. Eng. **137**, 106024 (2019). https://doi.org/10.1016/j.cie.2019.106024
9. Guerreiro, B.V., Lins, R.G., Sun, J., Schmitt, R.: Definition of smart retrofitting: first steps for a company to deploy aspects of industry 4.0. In: Hamrol, A., et al. (eds.) Advances in manufacturing. Lecture notes in mechanical engineering, pp. 161–170. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-68619-6_16

10. Lins, T., Rabelo Oliveira, R.A.: Cyber-physical production systems retrofitting in context of industry 4.0. Comput. & Ind. Eng. **139**, 106193 (2020) https://doi.org/10.1016/j.cie.2019.106193
11. World Wide Web Consortium: WebAssembly Core Specification. https://www.w3.org/TR/wasm-core-1/ (2019)
12. Hall, A., Ramachandran, U.: An Execution Model for Serverless Functions at the Edge. In: Landsiedel, O., Nahrstedt, K. (eds.) Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 225–236. ACM, New York, USA (2019). https://doi.org/10.1145/3302505.3310084
13. Mendki, P.: evaluating webassembly enabled serverless approach for edge computing. In: 2020 IEEE Cloud Summit, pp. 161–166. IEEE, Harrisburg, PA, USA (2020). https://doi.org/10.1109/IEEECloudSummit48914.2020.00031
14. Jacobsson, M., Willén, J.: Virtual machine execution for wearables based on WebAssembly. In: Sugimoto, C., Farhadi, H., Hämäläinen, M. (eds.) 13th EAI International Conference on Body Area Networks, pp. 381–389. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-29897-5_33
15. Aceto, G., Persico, V., Pescapé, A.: Industry 4.0 and Health: Internet of Things, Big Data, and Cloud Computing for Healthcare 4.0. Journal of Industrial Information Integration. **18**, 100129 (2020) https://doi.org/10.1016/j.jii.2020.100129
16. Dustdar, S., Murturi, I.: Towards IoT Processes on the Edge. In: Aiello, M., et al. (eds.) Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future vol. 12521, pp. 167–178. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73203-5_13
17. Nastic, S., Rausch, T., Scekic, O., Dustdar, S., Gusev, M., Koteska, B., Kostoska, M., Jakimovski, B., Ristov, S., Prodan, R.: A serverless real-time data analytics platform for edge computing. IEEE Internet Comput. **21**(4), 64–71 (2017). https://doi.org/10.1109/MIC.2017.2911430
18. Rausch, T., Hummer, W., Muthusamy, V., Rashed, A., Dustdar, S.: Towards a Serverless Platform for Edge AI. In: 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19). USENIX Association, Renton, WA (2019)
19. Mourtzis, D., Angelopoulos, J., Panopoulos, N.: Design and development of an edge-computing platform towards 5G technology adoption for improving equipment predictive maintenance. Proc. Comput. Sci. **200**, 611–619 (2022). https://doi.org/10.1016/j.procs.2022.01.259
20. Zhu, S., Ota, K., Dong, M.: Green AI for IIoT: energy efficient intelligent edge computing for industrial internet of things. IEEE Transactions on Green Communications and Networking. **6**(1), 79–88 (2022). https://doi.org/10.1109/TGCN.2021.3100622
21. Chen, C.-H., Lin, M.-Y., Liu, C.-C.: Edge Computing gateway of the industrial internet of things using multiple collaborative microcontrollers. IEEE Network **32**(1), 24–32 (2018). https://doi.org/10.1109/MNET.2018.1700146
22. Burresi, G., et al.: Smart retrofitting by design thinking applied to an industry 4.0 migration process in a steel mill plant. In: 2020 9th Mediterranean Conference on Embedded Computing (MECO) (2020). https://doi.org/10.1109/MECO49872.2020.9134210
23. Keshav Kolla, S.S.V., Lourenço, D.M., Kumar, A.A., Plapper, P.: of Industrial Internet of Things (IIoT). Procedia Computer Science. **200**, 62–70 (2022) https://doi.org/10.1016/j.procs.2022.01.205
24. Ilari, S., Carlo, F.D., Ciarapica, F.E., Bevilacqua, M.: Machine tool transition from industry 3.0 to 4.0: a comparison between old machine retrofitting and the purchase of new machines from a triple bottom line perspective. Sustainability. **13**(18), 10441 (2021) https://doi.org/10.3390/su131810441
25. Jaspert, D., Ebel, M., Eckhardt, A., Poeppelbuss, J.: Smart retrofitting in manufacturing: a systematic review. J. Clean. Prod. **312**, 127555 (2021). https://doi.org/10.1016/j.jclepro.2021.127555
26. Lins, T., Augusto Rabelo Oliveira, R., H. A. Correia, L., Sa Silva, J.: Industry 4.0 Retrofitting. In: 2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 8–15 (2018). https://doi.org/10.1109/SBESC.2018.00011
27. Mourtzis, D., Angelopoulos, J., Panopoulos, N.: Recycling and retrofitting for industrial equipment based on augmented reality. Procedia CIRP. **90**, 606–610 (2020). https://doi.org/10.1016/j.procir.2020.02.134
28. DIN: 91345: Reference Architecture Model Industrie 4.0 (RAMI4.0) (2016)
29. Haas, A., et al.: Bringing the Web up to Speed with WebAssembly. In: Cohen, A., Vechev, M. (eds.) Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017, pp. 185–200. ACM Press, New York, USA (2017). https://doi.org/10.1145/3062341.3062363
30. Mozilla and individual contributors: Understanding WebAssembly text format. https://developer.mozilla.org/en-US/docs/WebAssembly/Understanding_the_text_format (2021)
31. GitHub, Inc.: The State of the Octoverse. https://octoverse.github.com (2020)
32. Gadepalli, P.K., McBride, S., Peach, G., Cherkasova, L., Parmer, G.: Sledge: a Serverless-first, Light-weight Wasm Runtime for the Edge. In: Proceedings of the 21st International Middleware Conference, pp. 265–279. ACM, Delft Netherlands (2020). https://doi.org/10.1145/3423211.3425680

33. Napieralla, J.: Considering WebAssembly Containers for Edge Computing on Hardware-Constrained IoT Devices. Master thesis, Blekinge Institute of Technology, Karlskrona, Sweden (2020). https://www.diva-portal.org/smash/get/diva2:1451494/FULLTEXT02
34. Wen, E., Weber, G.: Wasmachine: Bring IoT up to Speed with A WebAssembly OS. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 1–4. IEEE, Austin, TX, USA (2020). https://doi.org/10.1109/PerComWorkshops48775.2020.9156135
35. Lehmann, D., Pradel, M.: Wasabi: A Framework for Dynamically Analyzing WebAssembly. In: Bahar, I., et al. (eds.) Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 1045–1058. ACM, New York, USA (2019). https://doi.org/10.1145/3297858.3304068
36. Stievenart, Q., Roover, C.: Compositional information flow analysis for WebAssembly programs. In: 2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 13–24. IEEE, Adelaide, SA, Australia (2020). https://doi.org/10.1109/SCAM51674.2020.00007
37. Mäkitalo, N., Mikkonen, T., Pautasso, C., Bankowski, V., Daubaris, P., Mikkola, R., Beletski, O.: WebAssembly modules as lightweight containers for liquid iot applications. In: Web Engineering, vol. 12706, pp. 328–336. Springer Nature, Cham (2021). https://doi.org/10.1007/978-3-030-74296-6_25
38. Li, B., Dong, W., Gao, Y.: WiProg: A WebAssembly-based approach to integrated iot programming. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pp. 1–10. IEEE, Vancouver, BC, Canada (2021). https://doi.org/10.1109/INFOCOM42981.2021.9488424
39. KUKA Roboter GmbH: KUKA Serie 2000: The all-rounders in the high payload range. https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/pf0020_kr_1502_en.pdf (2020)
40. Quix, C., Hai, R.: Data lake. In: Sakr, S., Zomaya, A. (eds.) Encyclopedia of Big Data Technologies. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63962-8_7-1
41. Mikkonen, T., Pautasso, C., Taivalsaari, A.: Isomorphic internet of things architectures with web technologies. Computer **54**(7), 69–78 (2021). https://doi.org/10.1109/MC.2021.3074258
42. Craig, J.J.: Introduction to Robotics: Mechanics and Control, 3rd edn. Pearson/Prentice Hall, Upper Saddle River, N.J (2005)
43. LaValle, S.M.: Planning Algorithms. Cambridge University Press, USA (2006). https://doi.org/10.1017/CBO9780511546877
44. Lehmann, D., Kinder, J., Pradel, M.: Everything Old is New Again: Binary Security of WebAssembly. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 217–234. USENIX Association, Online (2020)
45. Gurdeep Singh, R., Scholliers, C.: Warduino: a dynamic webassembly virtual machine for programming microcontrollers. MPLR 2019, pp. 27–36. Association for Computing Machinery, New York, USA (2019). https://doi.org/10.1145/3357390.3361029
46. Carnevale, L., Ruggeri, A., Martella, F., Celesti, A., Fazio, M., Villari, M.: Multi hop reconfiguration of end-devices in heterogeneous edge-iot mesh networks. In: 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1–6 (2021). https://doi.org/10.1109/ISCC53001.2021.9631500
47. Koren, I.: A standalone WebAssembly development environment for the internet of things. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds.) Web engineering, pp. 353–360. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-74296-6_27
48. Lacava, A., Zottola, V., Bonaldo, A., Cuomo, F., Basagni, S.: Securing bluetooth low energy networking: an overview of security procedures and threats. Comput. Netw. **211**, 108953 (2022). https://doi.org/10.1016/j.comnet.2022.108953
49. Vanhoef, M., Piessens, F.: Key reinstallation attacks: forcing nonce Reuse in WPA2. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 1313–1328 (2017). https://doi.org/10.1145/3133956.3134027
50. Lacava, A., Giacomini, E., D'Alterio, F., Cuomo, F.: Intrusion detection system for bluetooth mesh networks: data gathering and experimental evaluations. In: 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pp. 661–666 (2021). https://doi.org/10.1109/PerComWorkshops51409.2021.9430966
51. Sivanandam, N., Ananthan, T.: Intrusion detection system for bluetooth mesh networks using machine learning. In: 2022 International Conference on Industry 4.0 Technology (I4Tech), pp. 1–6 (2022). https://doi.org/10.1109/I4Tech55392.2022.9952758