# Autonomous Mobile Robots with Business Process Management Systems at the Edge

Ronny Seiger[1], Sara Pettinari[2]*, and Lukas Malburg[3,4]*

[1] RWTH Aachen University, Aachen, Germany, ronny.seiger@rwth-aachen.de
[2] Gran Sasso Science Institute, L'Aquila, Italy, sara.pettinari@gssi.it
[3] Trier University, Trier, Germany, lukas.malburg@uni-trier.de
[4] German Research Center for Artificial Intelligence (DFKI), Germany, lukas.malburg@dfki.de

**Abstract.** Business process management systems (BPMS) are widely used key components in software architectures of purely digital enterprise applications to organize, execute, and analyze business processes. More recently, BPMS have gained attention from the Internet of Things (IoT) and Cyber-Physical Systems (CPS) communities to be leveraged for automation and orchestration of sensors, actuators and more complex devices controlled by software. We present an experience report of using executable business processes and BPMS to orchestrate autonomous mobile robots. The focus of our investigations is on using robots as edge devices acting as local execution platforms for BPMS. We benchmark the impact of a common BPMS on the robot's resources during autonomous navigation and compare with traditional client-server deployments. The experimental results with real robots controlled by a Raspberry Pi demonstrate the feasibility of using common, standard-compatible BPMS in edge computing scenarios to orchestrate CPS.

**Keywords:** Business Process Management Systems · Autonomous Robots · Edge Computing · Software Architecture · Cyber-physical Systems.

## 1 Introduction

Business Process Management Systems (BPMS) are well-recognized and widely accepted general-purpose components for organizing, modeling, automating, and analyzing business processes in enterprise architectures. In software architectures of digital software systems, BPMS are used for combining message/event-driven behavior with human tasks and service orchestrations that are specified in executable process models (e.g., following the industry standard BPMN 2.0 [13]). Only recently, BPMS have started to receive attention as orchestrators of (business) processes in Internet of Things (IoT) and Cyber-Physical Systems (CPS), where they could also be leveraged for process automation and process mining [7].

In contrast to related approaches investigating the use of BPMS in classical desktop applications or client-server architectures for orchestrating IoT or CPS,

* These authors contributed equally.

we focus in this experience report on using BPMS in resource-constrained edge computing environments, namely to coordinate the mission of autonomous mobile robots [4, 20]. Unlike static IoT devices, mobile robots typically operate under strict resource constraints, intermittent connectivity, and real-time actuation needs. As a result, the placement and execution of orchestration logic become architectural design decisions rather than deployment choices. Using standard-compatible BPMS in this context facilitates (1) local ad-hoc coordination even under constrained connectivity, (2) robot mission programming based on graphical process notations, and (3) analysis and mining of processes executed by the robots based on event logs created by the BPMS.

In this work, we present a **software architecture** for coordinating autonomous mobile robots using an open-source BPMS as the central orchestration component. The architecture is designed to demonstrate the feasibility of using BPMS in mobile edge computing scenarios. The BPMS interacts with the robots via dedicated service interfaces that invoke the robots' existing capabilities. During their mission, the robots collect environment data from external sensors via, which could be useful, e.g., in hazardous environments or disaster scenarios where reliable connectivity is not available [11]. We conduct extensive **experiments** in a laboratory setting using *real* robots with different software architecture variants to benchmark the impact of the BPMS on the robot's constrained resources. We compare edge-computing scenarios in which the BPMS runs locally on the robot with classical client-server approaches in which the BPMS executes on an external machine. The results indicate that local execution introduces only marginal resource overhead. Energy consumption is even less in this setup compared to more network-intensive client-server settings. With locally deployed BPMS, the robots achieve a longer runtime, lower communication latencies, and increased autonomy through local processing and decision-making.

This experience report paper is organized as follows. Section 2 introduces preliminaries including scenarios and research objectives. Section 3 discusses related work. Section 4 presents the approach of using BPMS for orchestration of robots including example processes and software architecture. Section 5 evaluates the approach. Section 6 concludes the paper and presents future work.

## 2 Preliminaries

In this section, we present the mobile autonomous robots we use for our investigations, and we outline the research objectives, including non-functional requirements (NFRs), that drive this experience report.

### 2.1 Robot Setup

*Robot hardware and software.* We use two autonomous mobile robots (model: TurtleBot 4 Pro[5]), depicted in Fig. 1 (left), as representative CPS acting as

[5] `https://www.turtlebot.com/`

edge devices. Each robot is equipped with a Raspberry Pi 4 running Ubuntu and the Robot Operating System 2 (ROS2) [9]. The Raspberry Pi acts as the primary computing unit for robot control and other software components. It operates under limited computational resources (CPU and memory), WiFi-based connectivity, and battery-based power supply. The robots integrate perception and actuation capabilities, including LiDAR, camera-based sensing, and obstacle detection. These capabilities are provided via ROS2 and are the functional basis for higher-level orchestration logic. In addition, one robot (namely TurtleBot 0, cf. Fig. 1) is equipped with environmental sensors that measure air quality, including temperature, humidity, air pressure, $CO_2$ levels, and ambient light.

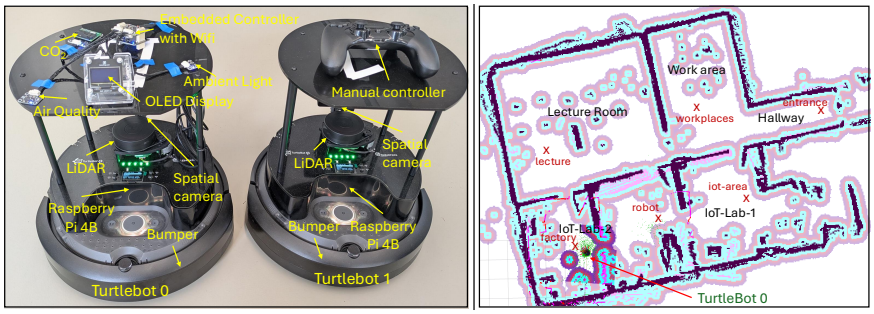



Fig. 1: Left: TurtleBot 4 robots with different sensors for autonomous navigation used in experiments. Right: Floor map with one TurtleBot robot in its current location and its current perception, different areas (black text labels) and targets (red text labels) are marked as a basis for autonomous navigation in RViz tool.

*Autonomous navigation capability.* ROS2 provides built-in functionality for localization, mapping, and autonomous navigation. In our setting, a shared map (cf. Fig. 1, right) is used by both robots to enable a goal-based movement between predefined coordinates. Navigation is considered as a capability exposed by the robot's control stack. The internal mechanisms for localization and path planning remain encapsulated within ROS2. External components can invoke this capability via higher-level interfaces. To decouple orchestration logic from low-level coordinate handling, we introduce a semantic abstraction layer by mapping $(x, y)$ coordinates to named locations in the laboratory, which are used as navigation targets for process models and robot missions.

### 2.2 Research Objectives

Given the capabilities of autonomous mobile robots, we identify the *orchestration of one or multiple robots* as the main functional objective of this work. Indeed, while individual robots' capabilities (e.g., navigation) can be executed autonomously, a higher-level mission logic requires explicit orchestration (e.g.,

task sequencing, multi-robot coordination, exception handling). We argue that this orchestration should be *decoupled* from the low-level robot control systems and realized via a standalone software component.

BPMS have been successfully applied to orchestrate CPS and IoT systems [3, 22, 26], and have also been explored for modeling and executing robotic missions [4, 6] through domain-specific engines. In this report, we investigate how BPMS can be applied as general-purpose software components to robot orchestration. Using BPMS with robots allows mission logic to be specified independently of robot-specific control. Business process models provide an explicit representation of control flow and enable modular composition of executable workflows (e.g.,in BPMN 2.0) [4], supporting a clear separation between orchestration and low-level actuation. On top of these premises, the following non-functional requirements (NFRs) related to the BPMS guide our investigations:

*(NFR1)* *Process Modeling Standard Compatibility*: The BPMS should support the execution of processes modeled using the standard notation BPMN 2.0 [13] to make robot programming accessible for domain experts and process engineers via existing business process modeling tools.

*(NFR2)* *Community acceptance and open source*: The BPMS should be widely used and an accepted solution for BPM-related tasks, ideally available as an open source artifact. It should not be tailored to robotic control, but instead represent a general-purpose orchestration platform applicable across domains.

*(NFR3)* *Interaction with external systems*: The BPMS should facilitate the interaction with the robot control software and with external systems.

*(NFR4)* *Process observability*: The BPMS should log process executions to enable process mining analysis of the robot's activities. Especially when the robot is acting autonomously, process analysis becomes essential to gain insights into the actual workflow executions.

Furthermore, we identify the following constraints ($C$) related to the mobile robots (cf. Sec. 2.1) as important aspects influencing architectural decisions:

*(C1)* *Constraint computing resources*: The robot relies on a Raspberry Pi that runs all the relevant software to control it, including a Linux-based operating system and ROS2. In contrast to a fully fledged computer or server, available CPU and memory resources are constrained.

*(C2)* *Limited connectivity*: The robot relies on wireless connectivity via WiFi. Connectivity range and bandwidth are limited, may be interrupted, or may be completely unavailable.

*(C3)* *Limited power supply*: The robot relies on a battery for power supply. Its runtime is limited and it is correlated with the amount of computing and communication done on the Raspberry Pi.

*(C4)* *Hardware platform*: The Raspberry Pi uses an ARM-based CPU. This partially limits the software components that it can run.

From these constraints, we derive the need to support local, decentralized process orchestration and decision-making to instruct robots with low latency

and minimal impact on their resources – an ideal use case for *edge computing* [27]. In many use cases for mobile autonomous robots (e.g., in disaster scenarios or hazardous environments), we cannot assume that constant, high-bandwidth connectivity to servers running the BPMS for orchestration is available.

We formulate the following two research questions that guide this experience report: **(RQ1)** How can a BPMN-compatible BPMS be used to orchestrate autonomous robots? **(RQ2)** What is the resource impact of running the BPMS directly on a mobile robot as an edge computing device?

## 3 Related Work

In this section, we discuss related work in the context of Business Process Management (BPM) for robots. We focus on three areas: (1) BPMS for IoT/CPS orchestration, (2) BPM for Robotic Systems, and (3) BPM for edge computing.

*BPMS for IoT/CPS Orchestration.* Chang et al. [3] investigate the application of BPM technologies for modeling and executing processes in CPS, proposing a service-based architecture with a dedicated management layer. Similarly, in [22], the authors introduce a layered systems architecture in which a common open-source BPMS orchestrates smart factory components via web services. Schönig et al. [19] present a bi-directional communication architecture for linking IoT sensors with business processes via message brokers. Other approaches extend BPMN with IoT-specific concepts and rely on custom execution engines [8,10,21]. Marrella et al. [11] and Valderas et al. [26] apply BPM concepts in dynamic and IoT-enhanced settings using layered or microservice-based architectures. From the discussed approaches, only one architecture explicitly discusses using common, standard-compatible BPMS [22], while the others rely on proprietary modeling extensions and engines.

While these works demonstrate the applicability of BPM to CPS and IoT, none discusses edge computing scenarios, and robots are only considered as process participants, but not as runtime platforms for BPMS.

*BPM for Robotic Systems.* Several works apply BPM concepts to robotic coordination and mission modeling. De la Croix and Lim [6] propose an architecture for modeling and executing BPMN processes onboard robots in a space exploration scenario. The work in [4] presents a framework for modeling and enacting multi-robot missions in BPMN, supported by a ROS-compatible execution engine. Rey et al. [16] and Neubert et al. [12] integrate workflow management with robotic systems in industrial or laboratory settings, typically hosting the BPMS externally. In [14] the authors introduce a context-aware workflow architecture that extends traditional BPMS with Industry 4.0 standards and a real-time context analyzer. Further approaches enhance BPMN-based mission modeling with model-driven [18] or multi-agent techniques [2]. Emerging approaches leverage *process mining* techniques to analyze executed missions. In [17] and [5], the authors discuss the use of process mining for multi-robot missions, generating event logs from robot activities and corresponding mined process models.

Overall, we observe an increasing adoption of BPM concepts in the robotics domain, including robot coordination. However, none of the approaches applies standard compatible, off-the-shelf engines. We came across only one work that discusses robots as BPMS runtimes in the context of distributed process execution architectures [20]. However, the work uses a self-developed formalism and engine for business process modeling and execution.

*BPM for Edge Computing.* Recent research addresses BPM applications to edge computing. Reiter et al. [15] propose a framework helping to decide which process mining tasks (e.g., preprocessing, conformance checking, model discovery) should be performed centrally or decentrally at the edge. Andersen et al. [1] introduce a distributed conformance checking approach that enables online conformance checking in a decentralized setting by checking the data locally and aggregating and alerting a central entity. Finally, Song et al. [25] present a distributed process mining framework for scalable IoT data analysis. These contributions focus on distributing specific analysis tasks between edge and cloud infrastructures. In contrast, our work investigates the deployment of BPMS directly on edge devices, namely mobile robots, as runtime platforms for business process executions.

To summarize, there is a growing interest in applying BPM-related techniques to distributed edge computing scenarios. However, the characteristics of the underlying execution platforms and edge devices are typically not the primary focus of these works.

## 4 Orchestration of Autonomous Robots with BPMS

In this section, we first present two reference scenario processes involving the autonomous mobile robots. We then present the software architecture of the orchestrated robot navigation system.

### 4.1 Reference Scenario Processes

We model the business processes to define the mission and coordination of the robots using the standard BPMN 2.0 formalism [13]. These processes are instantiated and executed by the BPMS. While BPMN 2.0 provides a rich formalism to model business processes, we focus on the interaction of the BPMS with the robots via *Service Tasks* that instruct a robot to move to a certain named location. More complex processes can be composed around this functionality.

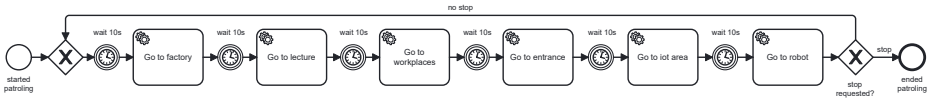



Fig. 2: Executable continuous robot patrol mission in BPMN 2.0, instructing the robot to go to different places via service tasks and briefly wait via timer events.

**Continuous Robot Patrol with Sensing** The process prescribes the mission of one robot that autonomously and continuously patrols a specific area. The robot is instructed to move to specific named locations in that area, stay at each location, and then move to the next location, driving in circles. The corresponding process model is shown in Fig. 2, using BPMN service tasks to invoke robot functionality and timer events for robot dwelling. An exclusive gateway is used to decide whether the robot should stop based on an external signal. Additionally, as presented in Sect. 2.1, we use the external sensors mounted to TurtleBot 0 to collect localized, contextual environment data during its patrolling. The sensing functionality operates as a separate component and is not linked to the business process to separate the navigation processes from sensor data collection. This configuration enables mission scenarios that combine mobility with optional environmental monitoring. Such scenarios are useful in settings where fixed sensing infrastructure is unavailable or impractical, e.g., during disasters or in hazardous areas, or in dynamic environments, such as construction sites.

**Decentralized Robot Coordination** The second scenario complements the previous process by introducing a simplified robot-robot coordination mission. In this setting, both TurtleBot robots run their own local BPMS instances and execute process instances independently. As depicted in Fig. 3, a process is started on TurtleBot 0 via a BPMN message event indicating critical conditions, for example, derived from local sensor data processing. This event triggers the execution of the service task *Activate emergency processes*, which invokes a new process instance on TurtleBot 1 to handle the emergency situation. Subsequently, TurtleBot 1 is instructed to navigate to the emergency location through a BPMN service task that calls its autonomous navigation capability. Upon arrival, an emergency-handling routine is initiated, abstracted here as a sub-process. In a practical use case, we imagine TurtleBot 0 to be continuously patrolling and analyzing environmental data via its sensors in a hazardous environment. Once it detects a critical condition (e.g., sudden high temperatures, low humidity, drop in air pressure, increased brightness, and fast increase in $CO_2$ levels–all together indicating a fire), it informs TurtleBot 1 via a local ad-hoc or mobile network to handle the emergency. TurtleBot 1 is equipped with fire-extinguishing materials that can be released after the autonomous driving to the emergency location. To target these materials, TurtleBot 1 could use its camera to identify hotspots.

### 4.2 Software Architecture of the BPMS-based Robot Orchestration

The software architecture is designed to fulfill the main functional requirement of orchestrating autonomous mobile robots, along with addressing the identified NFRs and constraints (cf. Sect. 2.2). The architecture decouples orchestration logic from robot control via explicit service interfaces and adapter components. In particular, our investigation focuses on RQ1, which examines the role of BPMS as central architectural components for robot orchestration. Fig. 4 shows the architecture components of the BPM-based robot orchestration system in two

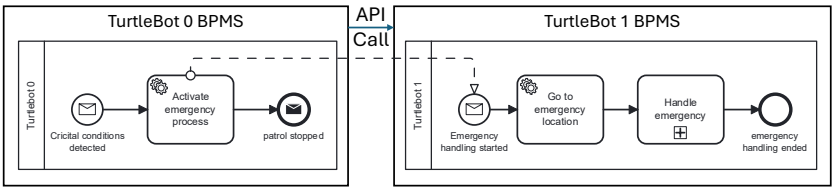



Fig. 3: Decentralized robot-to-robot (r2r) coordination, TurtleBot 0 and TurtleBot 1 each running a local BPMS instance and interacting with each other.

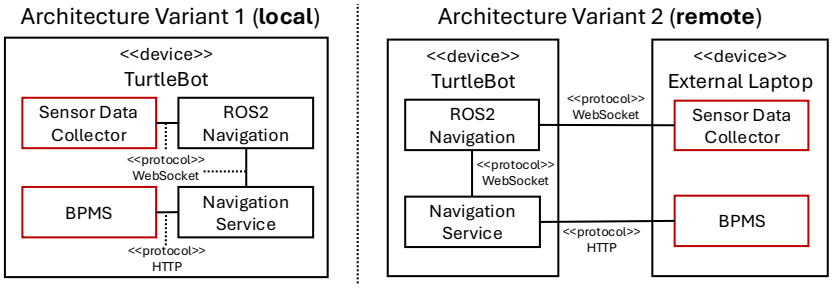



Fig. 4: Software Architecture with two variants (BPMS local on robot and BPMS remote on external laptop) investigated in evaluations.

architecture variants, which will be investigated in RQ2: Variant 1) the BPMS running locally on the robot, and Variant 2) the BPMS running on an external laptop. The architecture comprises the following components:

- *BPMS:* As specified in Sect. 2.2, the orchestration component must be deployable as a standalone software component. Based on NFR1–4, we identify *Camunda Platform 7* BPMS as a suitable system. It supports the BPMN 2.0 standard (NFR1) and provides a graphical process modeling tool. It is a mature, widely adopted open-source platform (NFR2) that offers mechanisms for interacting with external systems via connectors (NFR3). Furthermore, the platform records execution events in a relational database, enabling the extraction process logs for process mining (NFR4). The Java-based BPMS runs as a web application in a Tomcat server inside a JVM compatible with the ARM-based Linux of the Raspberry Pi (Constraint C4, cf. Sect. 2.2).
- *ROS2 Navigation:* The ROS2 Navigation consists of multiple components implementing the autonomous navigation functionality, running locally on the robot. The navigation functionality is provided via ROS2 and used as-is.
- *Navigation Service:* The Navigation Service is a self-developed microservice that sends navigation *goals* to ROS2 via the WebSocket protocol through *rosbridge*[6]. This high-level interface encapsulates the complex ROS2 internals, allowing us to remotely interact with the robots by triggering the goal navigation and monitoring the arrival while keeping safety-related features and

[6] `https://github.com/RobotWebTools/rosbridge_suite`

actuations during navigation encapsulated by ROS2 [24]. The Navigation Service exposes an HTTP API to receive requests for moving the robot to named locations managed by the service. The BPMS acts as a client (NFR4), sending navigation requests via its built-in *HTTP connectors*. These requests are modeled and configured in the business processes as BPMN *service tasks* (cf. Sect. 4.1). Other robot capabilities have to be encapsulated and exposed via (micro-)services to be invoked from the BPMS in a similar way.
- *Sensor Data Collector:* This self-developed component collects data from the environment sensors (cf. Fig. 1) via the sensor SDK provided by the manufacturer. Sensor data is recorded and linked with the robot position from the ROS2 navigation when one of the sensor values changes. The embedded controller of the sensors is equipped with a WiFi module. This allows us to run the Sensor Data Collector directly on the robot to record sensor data via USB (cf. Fig. 4, variant 1) or to run the Sensor Data Collector on an external computer and collect data via WiFi (variant 2).

## 5 Evaluation

The software architecture variants described in Sect. 4.2 were successfully implemented and deployed. The prototype enables modeling robot missions as BPMN process models with service tasks. These models are deployed to and executed by the BPMS. Each service task calls the Navigation Service, which invokes the robot's autonomous navigation. The BPMS logs the process executions, which can be analyzed using process mining (cf. Fig. 5). The implementation satisfies the main functional requirement and NFR1–NFR4, thereby addressing RQ1 (cf. Sect. 2.2). We evaluate the impact of deploying a common, standard-compatible BPMS directly on a mobile robot (RQ2) in a case study. In this context, the constrained robot resources C1–C4 (cf. Sect. 2.2) become relevant.

### 5.1 Experimental Setup

The experiments compare the architecture variants (i) running the BPMS and the Sensor Data Collector locally on the robot and (ii) executing these components on an external laptop (cf. Fig. 4). All other components remain unchanged.

*System Specifications:* The *edge computer* controlling the TurtleBot is a Raspberry Pi 4 with 4 GB of main memory and a quad-core Cortex-A72 (ARM v8) 64-bit CPU at 1.5 GHz. It is connected via 5 GHz WiFi (IEEE 802.11a) to the local network and powered by a battery with 2000 mAh capacity. The Raspberry Pi runs Ubuntu Linux 22.04 and ROS2 (Humble). The *external laptop* is a Lenovo ThinkPad T480s with 16 GB of main memory and an Intel Core i7-8550U CPU with 4 cores at 1.8 GHz. It is connected via Gigabit Ethernet to the local network and it has a steady power supply. The laptop runs Ubuntu Linux 22.04. The *BPMS* used on both devices is the Camunda Platform 7 (version 4.13) running in Apache Tomcat (version 9.0.33) inside a JVM (version 1.8).

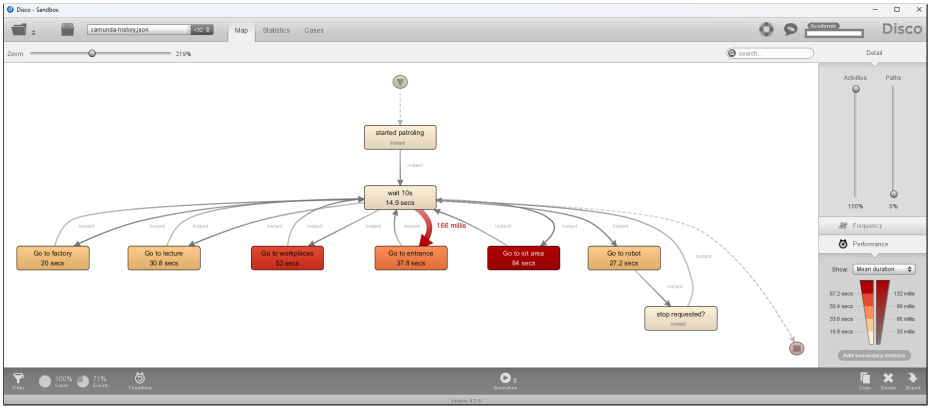



Fig. 5: Process analysis of one example robot patrolling process in a common process mining tool, showing activities and their mean durations in a directly-follows graph. This data was logged by the BPMS during process executions.

Table 1: Experiment overview by deployment variant (local vs remote) with total distances and runtimes, data volume, and event counts.

| Variant | Runs | Runtime [h] | Rounds | Distance [m] | Nav events | Sensor log [MB] |
|---|---|---|---|---|---|---|
| local | 5 | 0.90 | 18 | 422.55 | 261 | 8.04 |
| remote | 7 | 1.81 | 13 | 229.66 | 266 | 8.01 |
| all | 12 | 2.71 | 31 | 652.21 | 527 | 16.05 |

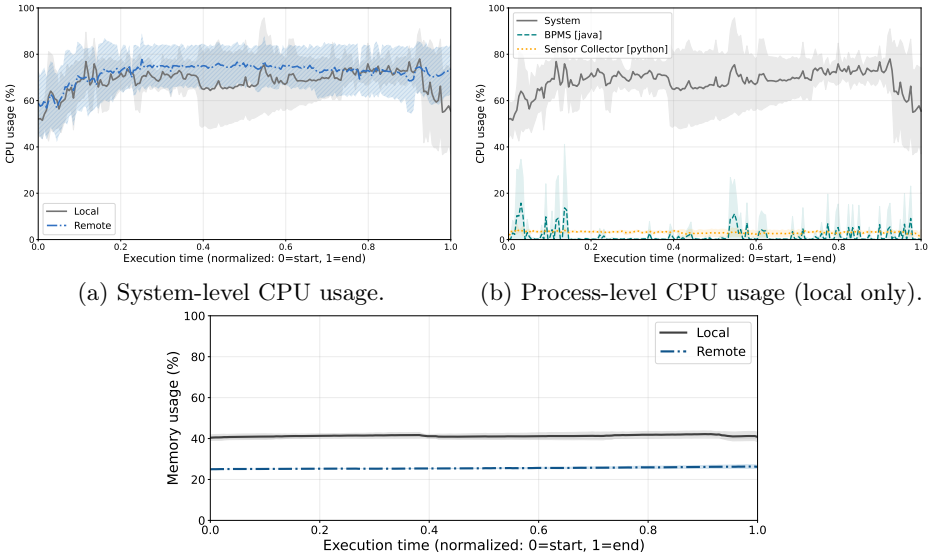
*System and Process Metrics:* The evaluation focuses on quantifying the resource impact of the BPMS and the Sensor Data Collector with respect to RQ2. In both variants, CPU and memory usage are measured over time via *pidstat* at the process and system levels. For the local variant, measurements are collected on the TurtleBot robot. For the remote variant, measurements are collected from the TurtleBot robot and external laptop to enable a direct comparison.

Battery levels and traveled distance of the TurtleBot are recorded over time using ROS2 interfaces. The BPMS records timestamped start and end events of process activities. The Navigation Service logs events related to the navigation (start and end). These measurements allow us to derive the latency between the BPMS and the Navigation Service.

### 5.2 Scenario 1: Continuous Robot Patrol with Sensing

The first process evaluated is the robot patrol business process shown in Fig. 2. Table 1 provides an overview of the collected data, including the number of experimental runs, their total runtimes, the number of rounds in the laboratory, and the total distance traveled. It also reports the number of navigation events (start/end pairs) and the size of data collected from the sensors.

**Results** We analyzed the ***system-level CPU usage*** for both architectural variants. Fig. 6a compares the aggregated CPU usage of the Raspberry Pi for the variants (*local* and *remote*) over normalized execution time. Fig. 6b shows the CPU utilization of the Raspberry Pi for the *local* variant. The horizontal axis represents normalized time in the interval [0,1], enabling comparison of runs with different absolute durations. The shaded area represents the standard deviation.




(a) System-level CPU usage.

(b) Process-level CPU usage (local only).

(c) System-level memory usage.

Fig. 6: Computing resources consumptions on robot over normalized time.

For both variants, CPU usage remains relatively stable over time. No progressive CPU increase or resource saturation is observed, indicating that business process execution does not introduce a computational load. The majority of CPU consumption is attributable to ROS2 navigation. Considering the total system CPU usage across all runs (cf. Fig. 6a), the *local* variant shows a mean CPU usage of $\sim 69.63\,\% \pm 7.68$, while the *remote* variant shows a mean CPU usage of $\sim 72.13\,\% \pm 10.72$. This slightly higher mean is caused by a single run with an average CPU usage of approximately 90 % due to extensive navigation-related tasks. Excluding this outlier, the remote variant results in a mean CPU usage of $\sim 68.21\,\% \pm 2.94$. The difference between the two variants does not indicate a significant increase in CPU load due to edge deployment of the BPMS.

For the *local* deployment, we analyzed relevant ***process-level CPU usage*** (cf. Fig. 6b). The BPMS (Java process) shows a mean CPU usage of $\sim 1.19\,\% \pm 0.65$. Brief spikes were only observed during the initial calls to the Navigation Service. The Java-based BPMS does not exhaust computational resources on the robot. The Sensor Collector (Python process) shows a continuous mean CPU

usage of $\sim$3.30% $\pm$ 0.05, only a small fraction of the available CPU resources. The results also indicate the robot's ability to perform its default actions (i.e., navigation) while acting as an environmental sensor data collector.

Overall, the results show that deploying the BPMS on the Raspberry Pi does not lead to a substantial increase in CPU utilization compared to the remote variant (RQ2). The overhead from the BPMS remains below 2% on average. This suggests that (i) edge deployment of BPMS is computationally feasible on the mobile robots, (ii) process orchestration does not compete significantly with navigation workloads, and (iii) there still remains CPU headroom available.

> ***Observation.*** *Deploying a BPMS at the edge introduces only marginal CPU overhead in a resource-constrained robotic environment.*

Fig. 6c shows the ***memory usage*** over normalized execution time for the *local* and *remote* deployments. For both variants, memory utilization remains constant over time. No increasing trend or progressive growth is observed, indicating that BPMS execution does not introduce increasing memory stress during the mission execution. The *local* deployment shows a mean memory usage of $\sim$41.49 % $\pm$ 1.15, whereas the *remote* deployment $\sim$25.55 % $\pm$ 0.48. The difference corresponds to the additional memory required by the BPMS runtime execution on the robot. However, the standard deviation remains small in both configurations, and no memory saturation was observed. Although local deployment increases memory usage compared to the remote variant, more than 50% of the memory remains available. The local deployment corresponds to $\approx$ 1.66 GB of the available 4 GB RAM on the Raspberry Pi (against the $\approx$ 1.02 GB for remote deployment), leaving $\approx$ 2.3 GB of memory available for additional processes.

> ***Observation.*** *Edge deployment of the BPMS increases memory usage, but this remains stable and within available system capacity.*

In Fig. 7, the mean ***battery usages*** per meter and per minute are shown for both variants. The *remote* variant exhibits a higher battery drop both per traveled meter (0.453 pct/m) and per minute (2.077 pct/min) compared to the *local* variant (0.415 pct/m and 1.537 pct/min, respectively). These results indicate that executing the BPMS locally on the robot does not increase energy consumption. On the contrary, the remote deployment results in a higher battery usage, which can be attributed to the communication overhead where orchestration requests are transmitted over the network rather than processed locally. From an architectural perspective, this suggests that local orchestration consumes less energy by minimizing communication between the robot and an external system.

> ***Observation.*** *Under the evaluated workload, local BPMS deployment results in lower battery consumption compared with a remote client-server setup.*

We compared the mean ***latency*** related to the communication between the BPMS invoking the Navigation Service on the robot and the service receiving the request. For the local variant we observed a mean latency of 0.379 seconds

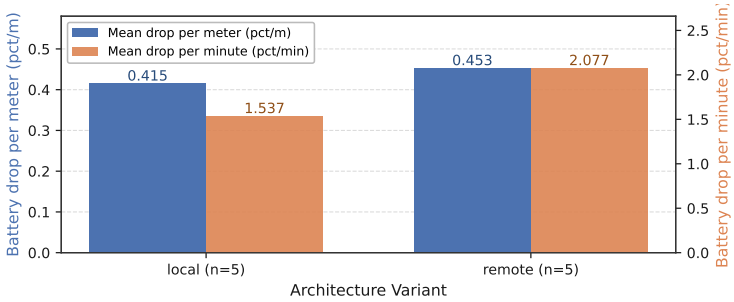



Fig. 7: Battery usage over time for the two architecture variants.

and for the remote variant a mean latency of 0.143 seconds. This seems counter-intuitive as in the local variant, the BPMS and Navigation Service run on the same machine. We attribute this difference to a slower network path resolution in the JVM. We specified in the process model the service tasks to invoke *localhost*, which results in more steps for address resolution than calling a fixed IP address. A change to call *127.0.0.1* reduced this overhead to below 100 ms and made the local deployment variant perform better. Under less optimal connectivity conditions, we expect the latency to increase for the remote deployment variant.

> ***Observation.*** *When configured correctly, running the BPMS and calling services locally outperforms remote BPMS and service calls in terms of latency.*

### 5.3 Scenario 2: Decentralized Robot Coordination

The second scenario process to be benchmarked corresponds to the decentralized robot-to-robot (r2r) coordination process depicted in Fig. 3. Each robot runs its own BPMS locally. The BPMS on TurtleBot 0 uses a BPMN service task including an API call to the BPMS of TurtleBot 1 to activate a new process instance, which invokes the local Navigation Service on TurtleBot 1. In this experiment, we focus our analysis on the communication latency between the two BPMS. We executed 15 experimental runs with a total runtime of 0.34 hours. The system logged 30 navigation events (15 pairs of start/end events). For each run, TurtleBot 1 was instructed by TurtleBot 0 to navigate to one specific location.

**Results** The communication latencies between the individual components of the robot-2-robot control scenario are depicted in Fig. 8 from left to right. We observed a mean latency in communication between the two BPMS controlling the robots individually of 518 ms, making the BPMS-based robot-robot-coordination a feasible solution in decentral edge computing scenarios where connectivity might be limited. The latency of the BPMS on TurtleBot 1 invoking its Navigation Service locally corresponds to our observations for the local

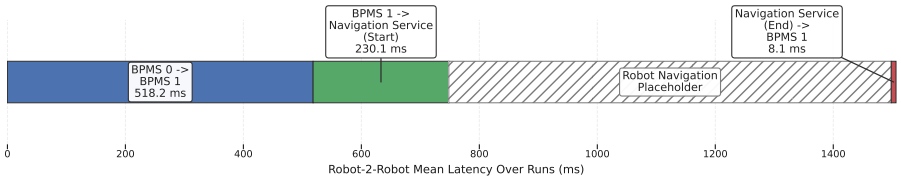



Fig. 8: Communication latency (in milliseconds) between the software components in robot-2-robot communication.

architecture variant in the scenario 1 process experiments. Note the placeholder for the actual navigation activity, which usually takes several seconds or minutes.

> ***Observation.*** *Decentral robot-robot coordination using a dedicated BPMS per robot at runtime is feasible with sub-second communication latencies.*

### 5.4 Threats to Validity and Limitations

The experimental results confirm the feasibility of using BPMS to orchestrate autonomous mobile robots acting as edge runtimes. However, the following limitations should be considered. First, we investigated a single BPMS implementation (Camunda Platform 7). While it represents a widely used, standard-compatible BPMN engine, results may vary with other BPMS platforms or runtime environments. Second, the experiments were conducted under laboratory conditions with stable network connectivity and static environments. Deployments in larger or more dynamic environments, or under unstable network conditions, may influence system behavior and performance. Third, the evaluation relies on a specific robot platform (TurtleBot with a Raspberry Pi 4). Although the platform represents a resource-constrained edge device, it still provides sufficient computing power for typical Java and Python components. Results may differ when deploying the approach to smaller or more constrained edge devices and robots that may not be capable of running an off-the-shelf BPMS. If orchestration via BPMS is not possible, more code-oriented approaches have to be followed or the orchestration has to be centralized/delegated to more capable devices.

## 6 Conclusion and Future Work

Modern mobile robots are capable of executing navigation missions autonomously to perform transportation or monitoring tasks. Despite this level of autonomy, they need to be instructed on where to move, orchestrated, and coordinated in multi-robot scenarios. In this work we propose a service-based software architecture for robot orchestration that uses business process management systems (BPMS) as central robot orchestration components. Based on the mobile robot's constraint computing resources, connectivity, and power supply, we argue that the BPMS should be directly deployed on the robots to leverage edge computing.

We conducted extensive experiments with real robots–comparing edge deployments of the BPMS with remote client-server configurations–to demonstrate the feasibility of running the BPMS locally on the robots in a more sustainable way. The results show that edge deployments of BPMS are feasible and that BPMS might become essential components of the software stack of ubiquitous systems.

In future work, we will conduct experiments under more realistic, outside conditions with limited connectivity and more dynamic environments, which will put the robot's autonomous navigation under more stress. We also plan to compare our results with using other available open-source BPMS. Furthermore, we will investigate the integration of dedicated planning components on the robots to make the process-based orchestration systems self-adaptive.

**Data Availability** The software artifacts *Navigation Service* and *Sensor Data Collector* are available at `https://github.com/ics-unisg/edge-robots`. Experimental data, analysis notebooks, and results are available on Zenodo [23].

**Acknowledgments.** This work has received funding from the Swiss National Science Foundation under Grant No. 10002384 (*TaSSAreCt* project) and from the MUR (Italy) Department of Excellence 2023 - 2027.

## References

1. Andersen, J., Rathje, P., Landsiedel, O.: Check My Flow: Distributed Conformance Checking at the Source. In: Business Process Management Workshops, LNBIP, vol. 534, pp. 101–112. Springer (2025)
2. Bourr, K., Tiezzi, F., Bettini, L.: Model-Driven Development of Multi-Robot Systems: From BPMN Models to X-Klaim Code. In: Leveraging Applications of Formal Methods, Verification and Validation. Rigorous Engineering of Collective Adaptive Systems, LNCS, vol. 15220, pp. 224–242. Springer (2025)
3. Chang, C., Srirama, S.N., Buyya, R.: Mobile Cloud Business Process Management System for the Internet of Things: A Survey. ACM Comput. Surv. **49**(4), 70:1–70:42 (2017)
4. Corradini, F., Pettinari, S., Re, B., Rossi, L., Tiezzi, F.: A BPMN-driven framework for Multi-Robot System development. Robotics Auton. Syst. **160**, 104322 (2023)
5. Corradini, F., Pettinari, S., Re, B., Rossi, L., Tiezzi, F.: A methodology for the analysis of robotic systems via process mining. In: Enterprise Design, Operations, and Computing. LNCS, vol. 14367, pp. 117–133. Springer (2023)
6. de la Croix, J.P., Lim, G.: Event-driven modeling and execution of robotic activities and contingencies in the Europa lander mission concept using BPMN. Jet Propulsion Laboratory, NASA Technical Reports Server (2020)
7. Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burattin, A., Di Ciccio, C., Fortino, G., et al.: The Internet of Things Meets Business Process Management: A Manifesto. IEEE Syst. Man Cybern. Mag. **6**(4), 34–44 (2020)
8. Kirikkayis, Y., Gallik, F., Winter, M., Reichert, M.: BPMNE4IoT: A Framework for Modeling, Executing and Monitoring IoT-Driven Processes. Future Internet **15**(3), 90 (2023)
9. Macenski, S., Foote, T., Gerkey, B.P., Lalancette, C., Woodall, W.: Robot Operating System 2: Design, architecture, and uses in the wild. Sci. Robotics **7**(66), eabm6074 (2022)

10. Mangler, J., Rinderle-Ma, S.: Cloud process execution engine: architecture and interfaces. arXiv preprint arXiv:2208.12214 (2022)
11. Marrella, A., Mecella, M., Sardiña, S.: Intelligent Process Adaptation in the SmartPM System. ACM Trans. Intell. Syst. Technol. **8**(2), 25:1–25:43 (2017)
12. Neubert, S., Roddelkopf, T., Gu, X., Göde, B., Junginger, S., Stoll, N., Thurow, K.: Architecture for a Combined Mobile Robot and Human Operator Transportation Solution for the Hierarchical Life Science Automation. In: ICINCO. pp. 41–49. SciTePress (2017)
13. Object Management Group: BPMN 2.0 Specification. https://www.omg.org/spec/BPMN/2.0/ (2011)
14. Ochoa, W., Legaristi, J., Larrinaga, F., Perez, A.: Dynamic context-aware workflow management architecture for efficient manufacturing: A ROS-based case study. Future Gener. Comput. Syst. **153**, 505–520 (2024)
15. Reiter, H., Edinger, J., Kabierski, M., et al.: ContinuumConductor : Decentralized Process Mining on the Edge-Cloud Continuum. CoRR **abs/2512.07280** (2025)
16. Rey, R., Corzetto, M., Cobano, J.A., Merino, L., Caballero, F.: Human-robot co-working system for warehouse automation. In: ETFA. pp. 578–585. IEEE (2019)
17. Roldán, J.J., Olivares-Méndez, M.A., del Cerro, J., Barrientos, A.: Analyzing and improving multi-robot missions by using process mining. Auton. Robots **42**(6), 1187–1205 (2018)
18. Sadik, A.R., Goerick, C.: Multi-Robot System Architecture Design in SysML and BPMN. Advances in Science, Technology and Engineering Systems Journal **6**(4), 176–183 (2021)
19. Schönig, S., Ackermann, L., Jablonski, S., Ermer, A.: Iot meets bpm: a bidirectional communication architecture for iot-aware process execution: S. schönig et al. Software and systems modeling **19**(6), 1443–1459 (2020)
20. Seiger, R., Herrmann, S., Aßmann, U.: Self-healing for distributed workflows in the internet of things. In: International Conference on Software Architecture Workshops. pp. 72–79. IEEE (2017)
21. Seiger, R., Huber, S., Schlegel, T.: Proteus: An integrated system for process execution in cyber-physical systems. In: International Workshop on Business Process Modeling, Development and Support. pp. 265–280. Springer (2015)
22. Seiger, R., Malburg, L., Weber, B., Bergmann, R.: Integrating process management and event processing in smart factories: A systems architecture and use cases. J. Manuf. Syst. **63**, 575–592 (2022)
23. Seiger, R., Pettinari, S., Malburg, L.: Autonomous mobile robots with business process management systems at the edge (dataset) (Mar 2026). `https://doi.org/10.5281/zenodo.18920693`
24. Seiger, R., Seidl, C., Aßmann, U., Schlegel, T.: A capability-based framework for programming small domestic service robots. In: MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering. pp. 49–54. ACM (2015)
25. Song, C., Ren, Z., Hu, R., Lu, L.: Enhancing IoT Compliance Checking with Distributed Process Mining: A Scalable Framework for Log Data Streams. In: Wireless Sensor Networks, CCIS, vol. 2341, pp. 130–140. Springer (2025)
26. Valderas, P., Torres, V., Serral, E.: Modelling and executing IoT-enhanced business processes through BPMN and microservices. J. Syst. Softw. **184**, 111139 (2022)
27. Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S.: Challenges and Opportunities in Edge Computing. In: SmartCloud. pp. 20–26. IEEE (2016)