Journal of Intelligent Information Systems
https://doi.org/10.1007/s10844-022-00766-w

Applying MAPE-K control loops for adaptive workflow management in smart factories

Lukas Malburg[1,2] · **Maximilian Hoffmann**[1,2] · **Ralph Bergmann**[1,2]

Received: 9 March 2022 / Revised: 28 July 2022 / Accepted: 10 November 2022
© The Author(s) 2023

Abstract
Monitoring the state of currently running processes and reacting to ad-hoc situations during runtime is a key challenge in *Business Process Management (BPM)*. This is especially the case in cyber-physical environments that are characterized by high context sensitivity. MAPE-K control loops are widely used for self-management in these environments and describe four phases for approaching this challenge: Monitor, Analyze, Plan, and Execute. In this paper, we present an architectural solution as well as implementation proposals for using MAPE-K control loops for adaptive workflow management in smart factories. We use *Complex Event Processing (CEP)* techniques and the process execution states of a *Workflow Management System (WfMS)* in the monitoring phase. In addition, we apply automated planning techniques to resolve detected exceptional situations and to continue process execution. The experimental evaluation with a physical smart factory shows the potential of the developed approach that is able to detect failures by using IoT sensor data and to resolve them autonomously in near real time with considerable results.

Keywords Complex event processing · Automated planning · Cyber-physical environments · Smart factories · Adaptive workflow management · Process adaptation

✉ Lukas Malburg
malburgl@uni-trier.de; lukas.malburg@dfki.de

Maximilian Hoffmann
hoffmannm@uni-trier.de; maximilian.hoffmann@dfki.de

Ralph Bergmann
bergmann@uni-trier.de; ralph.bergmann@dfki.de

[1] Artificial Intelligence and Intelligent Information Systems, University of Trier, Behringstraße 21, Trier, 54296, Germany

[2] German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, Behringstraße 21, Trier, 54296, Germany

Published online: 25 January 2023

1 Introduction

Business Process Management (BPM) (Dumas et al., 2018) is a research field of high interest with different areas of usage, e.g., in companies, in public administration, or in smart environments (Seiger et al., 2019) such as smart homes or smart factories (e.g., Schönig et al., 2020; Marrella et al., 2017; Malburg et al., 2020a, b; Seiger et al., 2020; 2022). Smart IoT environments that are characterized by high context sensitivity and huge amounts of real-time data can particularly benefit from the advantages of applying BPM solutions (Janiesch et al., 2020). Although BPM research for smart environments has been attracting more and more attention recently, it is still in its infancy. To control smart spaces on a higher level in a more process-oriented way (Malburg et al., 2020b; Seiger et al., 2022), state-of-the-art *Workflow Management Systems (WfMSs)* can be used. However, current WfMSs and imperative process models are rather limited w.r.t. flexibility and often only provide means to handle simple, mostly expected situations that must be fully specified within the model (Reichert & Weber, 2012; Weber et al., 2004; Wieland et al., 2015). One main drawback of creating fully specified process models is the large effort in modeling and that not all occurring situations are known in advance (Marrella et al., 2017). Thus, flexibility is inevitably required in smart environments to react to expected and also unexpected situations by adapting faulty processes (Marrella et al., 2017; Malburg et al., 2020b; Wieland et al., 2015).

There already are approaches that address these issues and use more advanced methods (e.g., *Case-Based Reasoning (CBR)* (Weber et al., 2004; Grumbach & Bergmann, 2019) or Automated Planning (Marrella et al., 2017; Marrella, 2019)) for adaptive workflow management but they have either only been used to a limited extent in cyber-physical environments, or they are only conceptual approaches that have not been evaluated in real-world application scenarios. Additionally, related approaches mostly only address adapting processes while not dealing with the concrete cause that implies the need for adaptation (Dadam & Reichert, 2009; Pesic & van der Aalst, 2006). Simple triggers such as human intervention are often not feasible for real-world scenarios and reveal the need for autonomous detection and advanced adaptation techniques of failures in a holistic framework that is demonstrated in a real-world application context.

In this paper, we present an approach for adaptive workflow management in smart factories by using the architectural blueprint of MAPE-K (*Monitor*, *Analyze*, *Plan*, and *Execute* phases as well as the knowledge *K* (IBM, 2006)) control loops, which builds the basis to develop self-managing software systems in cyber-physical environments (IBM, 2006; Muccini et al., 2016) and which is also used in the context of BPM (Seiger & Aßmann, 2019; Seiger et al., 2019). The approach combines *Complex Event Processing (CEP)* methods for monitoring smart environments to derive high-level error events from low-level IoT sensor data and AI planning techniques for BPM (Marrella, 2019) to adapt processes for recovery and further execution. We enhance the current research by introducing a holistic framework that is implemented, demonstrated, and evaluated with a physical smart factory from Fischertechnik that simulates two independent shop floors. Incorporating this factory for demonstration and evaluation of the approach strengthens the results and validates the findings in a scenario that is closer to real-world production lines and, thus, facilitates the transfer to them (Malburg et al., 2020b; Seiger et al., 2022).

The paper is structured following the *Design Science Research (DSR)* methodology for information systems research as proposed by Hevner et al. (2004): Section 2 describes

fundamental concepts covering the used physical smart factory, automated planning in combination with BPM as well as related work (*Rigor–Knowledge Base*). Further, Section 3 presents the proposed research artifact as a framework for implementing a MAPE-K control loop for self-management of manufacturing processes to increase the flexibility and adaptivity in failure situations (*Develop/Build*). Section 4 examines the suitability of the developed research artifact in an experimental evaluation where it is used to automatically monitor and analyze manufacturing processes in a smart factory and to plan adaptations with AI planning techniques in case of failures (*Justify/Evaluate*). By using the experimental evaluation, it is also shown that the created artifact is potentially useful for real production lines (*Relevance–Environment*). Finally, Section 5 concludes the paper and shows areas of future work.

2 Foundations and related work

As defined in the *Design Science Research (DSR)* (Hevner et al., 2004) methodology, the rigor of the proposed approach is based on a knowledge base consisting of relevant foundations, methods, and models that are presented in the following sections. More adaptivity in managing production processes can be facilitated with MAPE-K control loops if the technology for implementing each of the individual phases is well-matched. The proposed approach combines *Complex Event Processing (CEP)* for monitoring with automated planning for (re-)planning production processes. We use a physical smart factory from Fischertechnik that allows us to conduct close to reality research. The model is controlled in a process-oriented way by using a service-based architecture and state-of-the-art *Workflow Management Systems (WfMSs)* (see Section 2.1). We also introduce the basics for autonomic MAPE-K control loops in Section 2.2 and for automated planning in BPM in Section 2.3. In addition, we present related work for adaptive workflow management and for implementing MAPE-K control loops in BPM research in Section 2.4.

2.1 Physical smart factories for business process management research

Learning Factories (Abele et al., 2017) are gaining importance in education, training, and Industry 4.0 research (cf. Simons et al., 2017; Prinz et al., 2016; Seiger et al., 2020). They are used for developing and evaluating research artifacts in a laboratory and experimental environment before moving to real production settings. Instead of relying entirely on simulated data for developing and evaluating research artifacts, physical models provide much more realistic data and run-time behavior such as ad-hoc interventions (Malburg et al., 2020b; Seiger et al., 2022). For this reason, the developed artifacts are more easily transferable to real-world production lines and, thus, relevant for these environments (*Relevance–Environment* in DSR (Hevner et al., 2004)). In our research (Klein & Bergmann, 2019; Malburg et al., 2020a, b, 2021; Hoffmann et al., 2022), we use a physical smart factory from Fischertechnik.[1] The custom model simulates two independently working production lines consisting of two shop floors that are linked for the exchange of workpieces. Figure 1 illustrates the used smart factory. Each shop floor consists of 5 identical resources: a sorting

[1]Fischertechnik is a company that produces modules for simulating factories on a small scale. More general information can be found at https://www.fischertechnik.de/en/simulating/industry-4-0 and information regarding our custom model at https://iot.uni-trier.de.

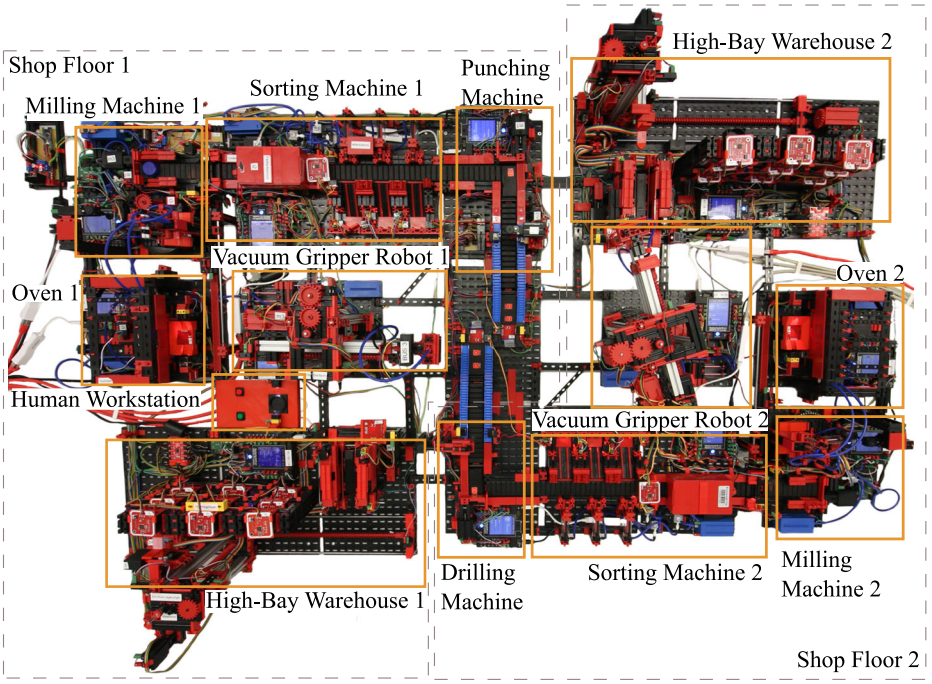

Fig. 1 The physical smart factory. (Source: Malburg et al. (2020b))

machine with color detection, a multi-processing station with an *Oven (OV)* and a *Milling Machine (MM)* connected by a *Workstation Transport (WT)*, a *High-Bay Warehouse (HBW)* for unfinished and finished workpieces, and a *Vacuum Gripper Robot (VGR)*. In addition, the first floor has a *Punching Machine (PM)* and a *Human Workstation (HW)*, and the second floor a *Drilling Machine (DM)*. The HW is a special resource that allows simulating human interaction during production, e.g., conducting a manual quality inspection of the product. It is equipped with multiple, programmable buttons that control and report the result of the human activity, e.g., the workpiece tolerances are insufficient. Each shop floor is further equipped with 13 light barriers, 16 switches, and 3 capacitive sensors for control of the actuators consisting of 16 motors, 4 compressors, and 8 valves. The resources are enhanced with sensors mounted on moving parts, motors, and compressors for condition and pressure monitoring for predictive maintenance (Klein & Bergmann, 2019). Moreover, NFC readers/writers are integrated into the stations, resulting in 28 communication points. This allows each workpiece to be tracked and required manufacturing operations and parameters to be retrieved and adjusted during production. A camera module is placed above the two shop floors to track the workpieces in the shop floor by object detection techniques (Malburg et al., 2021). The workpieces used for simulating the production are small cylindrical blocks (height $= \sim 1.4$ cm, diameter $= \sim 2.6$ cm) of varying colors, each equipped with an NFC tag, which contains information regarding the individual workpiece such as an identifier, the type (i.e., color), the current production state, and timestamped production history. The sensors and actuators of the processing stations are connected to Fischertechnik TXT controllers; 6 Raspberry PIs and 2 Arduinos are used for managing the additional sensors and

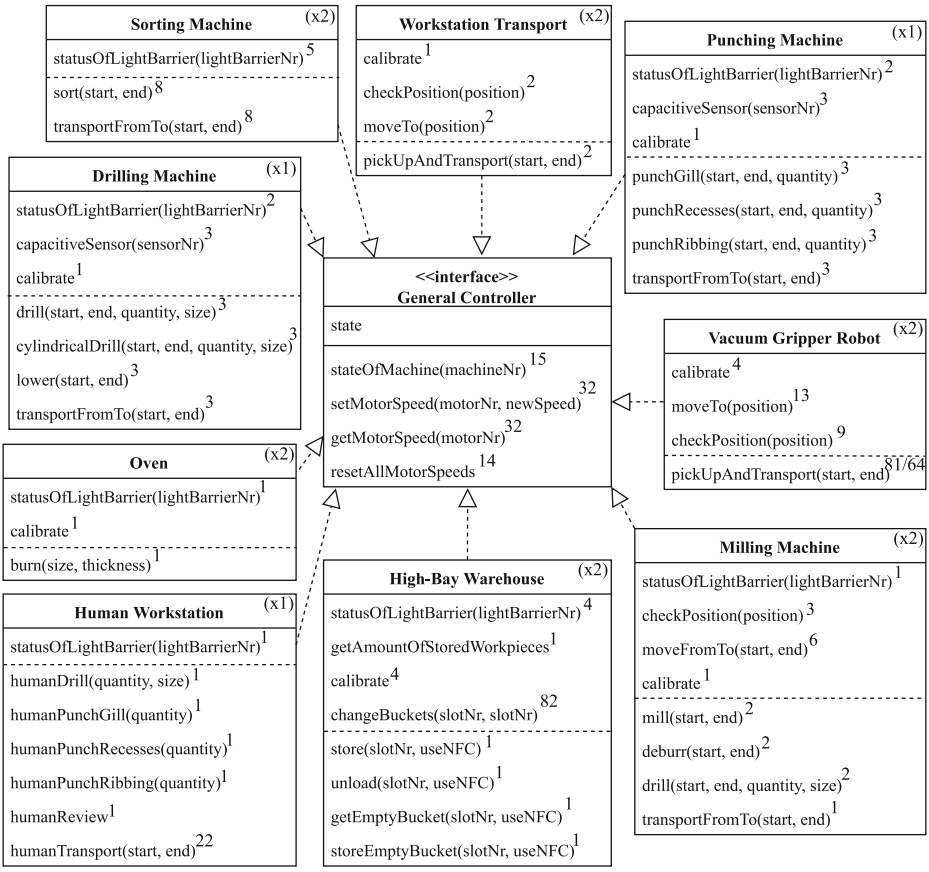

Fig. 2 Class diagram of provided capabilities for each resource. (Based on: Malburg et al. (2020a))

the camera, which are all linked via Ethernet to a central network switch. Apache Kafka is used as a database to store the smart factory IoT sensor data over time.

In order to control the physical smart factory in a process-oriented way, previous work (Malburg et al., 2020a, b; Seiger et al., 2022) presents a service-based architecture with semantic annotations that abstracts from low-level resource functionalities and allows other systems such as WfMSs to invoke the smart factory services remotely. Figure 2 illustrates a class diagram as an overview of the capabilities provided by the resources. The capabilities can be divided into 1) simple ("getter-and-setter") functionalities that are used to retrieve IoT sensor data or to specify general equipment parameters and 2) manufacturing functionalities that perform a physical activity in the smart factory. Both types of capabilities are separated with a dashed line in Fig. 2. For example, the manufacturing functionality for executing a transport of a workpiece with the VGR is called `pickUpAndTransport` and has two parameters (*start* and *end*) for receiving variable start and end positions. Due to the variety of pickup and drop off positions, there are 81 possible configurations[2] for this capability in the first shop floor and 64 in the second one. Each of these configurations has

[2]The superscript number above the individual methods in Fig. 2 illustrates this.

different preconditions and effects, such as light barriers at specific physical positions that must be interrupted. For this reason, each parameter configuration must be implemented by an individual web service with its specific semantic annotations. All in all, this results in 256 web services for the manufacturing functionalities in the smart factory.

By using semantic annotations for each resource functionality, it is possible to describe what preconditions must be satisfied to execute the activity and the effects that must hold after successfully executing the service in the physical world, i.e., the smart factory. In Fig. 3, a part of the semantic annotations of a web service with its preconditions and effects is illustrated. Instances of classes from the domain ontology are represented by violet rectangles, and the classes themselves by orange ellipses. Green rectangles with rounded corners depict data properties and edges denote relations, i.e., object properties by a solid line and data properties by a dashed line. During execution of the depicted service, all preconditions are checked before the execution of the service is started in the smart factory. For example, it is checked whether the VGR, the HBW, and the oven are ready. This is required to ensure that a check of the corresponding light barriers is possible and that the corresponding resource is ready for the transport. In addition, it is checked whether the light barrier at the HBW is interrupted, which indicates that the bucket with the workpiece is located at this position, and whether the light barrier at the storage place of the oven is not interrupted, which in turn indicates that the place can be used. Effects are checked after each service invocation. For example, after the successful transport, the workpiece is now located at the storage place of the oven and its corresponding light barrier must be interrupted.

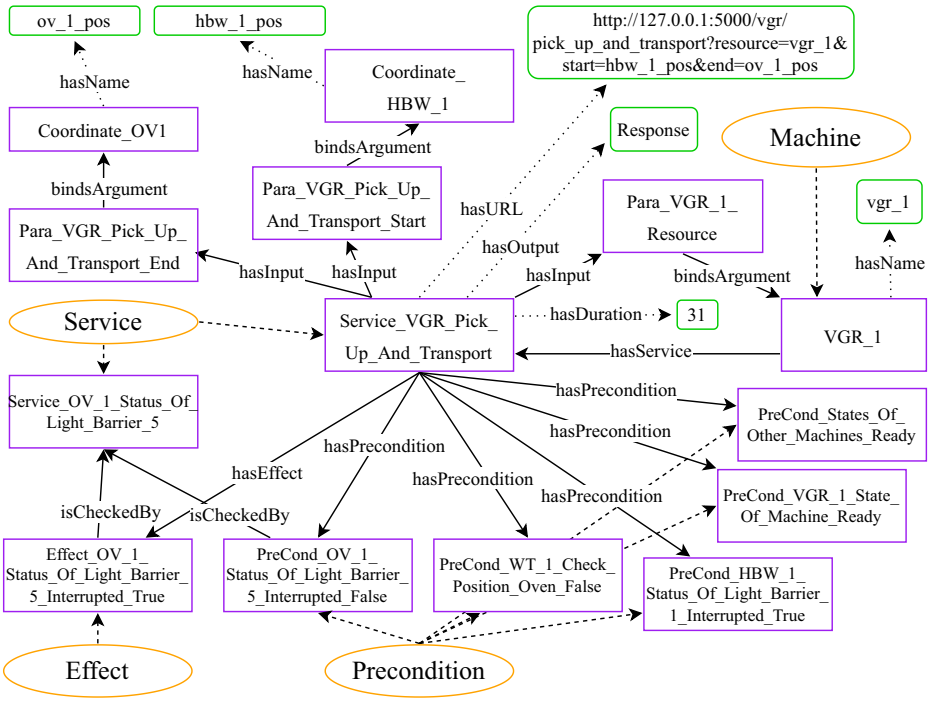

Fig. 3 Semantic annotations of the *Pick Up and Transport* service from the Vacuum Gripper Robot (VGR) as a graph

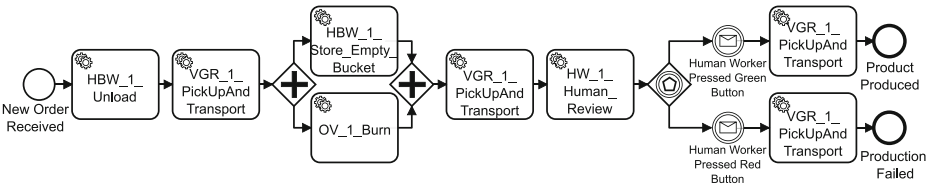

Fig. 4 Sheet metal manufacturing process as BPMN 2.0 model

By using the described service-based architecture, it is possible to model and execute BPMN 2.0[3] processes that consist of BPMN-compliant service tasks.[4]

The BPMN 2.0 process in Fig. 4 illustrates a sheet metal manufacturing process. Sheet metal production serves as a placeholder for industrial manufacturing processes. It is characterized by a high degree of individualization, since each piece of sheet metal is manufactured individually for a customer according to the corresponding requirements. This leads to processes that are not completely standardized. Although a similar sequential control-flow is often followed, the processes sometimes vary greatly in terms of parameter settings. In addition, some machines in the factory can be used for several tasks during sheet metal production: For example, the milling machines can be used to mill sheet metals but also for drilling holes with certain sizes and numbers of holes. Thus, it is possible to use several machines beside the respective machine in the other production line for certain activities, e.g., in cases of failures during production that in turn require more adaptive workflow management for a flexible execution.

2.2 MAPE-K control loops

When executing processes in smart environments that are characterized by high dynamics, the execution is always at risk of failures without already known solutions. MAPE-K control loops (IBM, 2006) provide a blueprint for self-managing information systems that is applicable in process execution scenarios (Seiger et al., 2019; Seiger & Aßmann, 2019). The entity of interest is the managed element, which can generally be any part of an information system. We focus on managed elements in the form of processes, as visualized by the example process in Fig. 5 and discussed in BPM-related literature (cf. Seiger et al., 2019; Seiger & Aßmann, 2019). The following explanations and its examples are based on IBM (2006), Seiger et al. (2019), and Seiger & Aßmann (2019) and especially target cyber-physical environments in which MAPE-K control loops are frequently applied (Muccini et al., 2016).

The MAPE-K cycle consists of four phases that are designed as a continuous loop which can be iteratively instantiated several times:

1. *Monitor*: The environment in which the managed element is executed is continuously monitored by using IoT sensor data. The observed data can be manifold, e.g., topology information, smart home data, and process execution metrics. Throughout the monitoring process, the data is usually continuously processed, e.g., by aggregation or filtering.

[3] https://www.omg.org/spec/BPMN/2.0.2/

[4] To illustrate the mapping between BPMN 2.0 service tasks and the corresponding web services, we use as a naming scheme the executing resource followed by the capability it actuates. Parameter settings are for the sake of readability not illustrated.

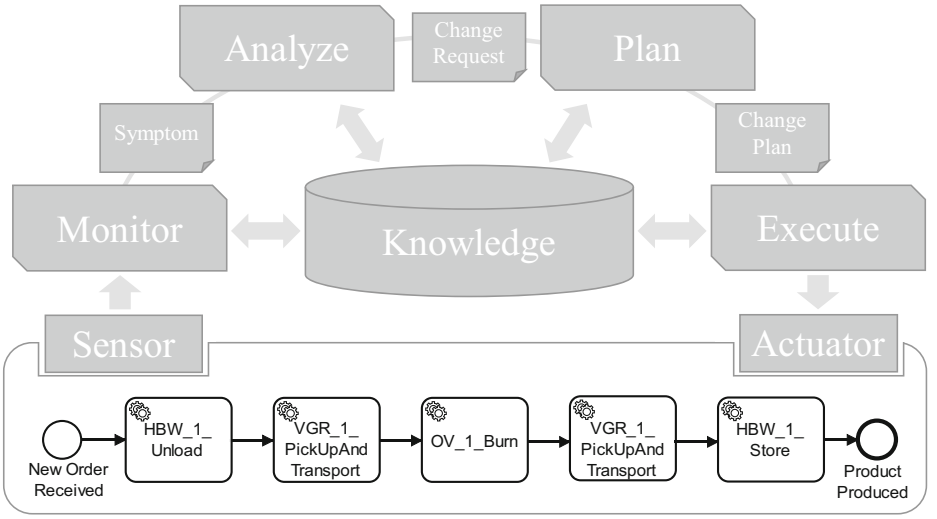

Fig. 5 MAPE-K control loop for processes. (Based on: IBM (2006) and Seiger et al. (2019))

In case the process is not fulfilling the specified goals and objectives, *Symptoms* are described and the situation is further analyzed in the subsequent phase.

2. *Analyze*: In this phase, the current situation and the monitored symptoms are checked in more detail. For instance, reasoning can be performed with the symptoms to infer higher-level information. The result of this phase is a *Change Request* that is passed on to the *Plan* phase.
3. *Plan*: Based on the *Change Request* from the preceding phase, a *Change Plan* with corresponding actions is created in this phase. The goal is to create actions that lead to the managed element fulfilling the specified goals and objectives again. The request can have various forms, ranging from simple, practical changes, e.g., repetition of the process execution, to more complex and advanced strategies such as the structural modification and migration of the process model.
4. *Execute*: Eventually, the *Change Plan* is executed by actuators[5] in the environment in which the managed element operates. This usually involves a WfMS as the main execution engine if the managed element is a process. To lower the risk of the changes applied to the managed element not restoring the desired state, it is possible to re-run the complete MAPE-K loop and, especially, to monitor and analyze for remaining or new upcoming problems. In addition, it is also possible to use appropriate validation tests or to execute the process in a simulation before execution in the native environment.

Knowledge as part of the MAPE-K control loop is shared between all phases. Relevant knowledge for autonomic systems can be topology information, historical logs, metrics, symptoms, and policies. The knowledge can also be updated by the phases to share new information with all phases. Semantic annotations for web services or the used

[5]The original publication (IBM, 2006) uses the term "effector", which is not as common in the IoT field as the term actuator that we use here.

domain ontology FTOnto[6] (Klein et al., 2019) describing the physical smart factory (see Section 2.1) can also be part of the MAPE-K knowledge.

2.3 Automated planning for business process management in industry 4.0

Implementing a blueprint such as MAPE-K in the context of *Cyber-Physical Production Systems (CPPS)* (Monostori, 2014) can be realized by sophisticated methods based on *Artificial Intelligence (AI)* that can quickly respond to situations, e.g., failures of machines, occurring during production. In our research, we use state-of-the-art *Workflow Management Systems (WfMSs)* to execute the production processes in the smart factory. However, these state-of-the-art WfMSs are very limited w.r.t. flexibility and often only provide means to handle simple, mostly expected situations that must be fully specified in the workflow model before execution (Reichert & Weber, 2012; Marrella et al., 2017). Since several situations, including unexpected events, can occur in smart environments (Marrella et al., 2017), more sophisticated methods for process adaptations are needed that also consider the environmental context of process execution for enabling more adaptive workflow management and handling failure situations appropriately. A promising approach to remedy these problems is the use of *Automated Planning*[7] in BPM (Marrella 2017, 2019; Rodríguez-Moreno et al. 2007), since "BPM is in need of techniques that go beyond hard-coded solutions" (Marrella, 2019). The work of Marrella (2019) presents three use cases based on the BPM life cycle (Dumas et al., 2018) in which AI planning can be applied in BPM. In the *Design* phase of the life cycle, AI planning can be used to automatically generate process models. More precisely, planning methods can be applied to generate an imperative process model, since a plan with its sequential and partial-ordered actions is rather similar to a process (Marrella, 2019). A second application area for combining BPM and AI planning is the *Implementation & Execution* phase of the life cycle. During this phase, AI planning is used in cases in which processes cannot be further executed as planned and adaptations are required. This is especially the case in dynamic domains such as in smart manufacturing (Wieland et al., 2015; Malburg et al., 2020b), in smart homes (Seiger et al., 2019), or for knowledge-intensive processes in emergency management (Marrella et al., 2017). To combine BPM and AI planning for autonomous process adaptation, real-time monitoring during the process execution is required to capture situations in which an adaptation is needed (Marrella, 2019). The third phase of the BPM life cycle is the *Diagnosis & Optimization* phase, in which planning techniques can be used for conformance checking (e.g., de Masellis et al., 2022), i.e., trace alignment. All in all, there are several application scenarios for using AI techniques, such as automated planning, in BPM. However, all have in common that a suitable representation of the problem must be specified by using a specification language. For automated planning, the *Planning Domain Definition Language (PDDL)* (McDermott et al., 1998) in its several versions (see Haslum et al. (2019) for a comprehensive overview) is the de facto standard to express planning problems and enables the use of state-of-the-art planners. A classical planning problem expressed in PDDL consists of an *Initial State* in which predicates are used to describe the current world state, i.e., the current process execution state as well as the state of the environment. Similarly, the *Goal State* is defined with predicates that should hold after the planning process is finished. Moreover, it is possible to use *Objects* in the planning

[6] https://gitlab.rlp.net/iot-lab-uni-trier/ftonto

[7] Since *Automated Planning* is a technique from the field of *Artificial Intelligence (AI)*, it is also often called *AI Planning*.

problem to express possible parameter values for actions (see Haslum et al. (2019) for more details and Section 3.3 for a concrete example). For describing the planning domain, instantaneous actions also called planning operators must be defined. These actions are applied to the initial state during problem-solving, which in turn leads to a state transition of one world state to another world state. The planning process terminates after applying several planning actions if the goal state is reached (Haslum et al., 2019; McDermott et al., 1998). In general, planning actions are very similar to individual tasks in processes, as they also cause a change of state. Consequently, generated plans of classical planning problems are simply sequential chains of tasks as in control-flow oriented processes.

2.4 Related work

In this section, we present as part of the foundations some selected related approaches that deal with adaptive workflow management. The included publications give a broad overview of various topics and techniques applied for more adaptive workflows. We divide the approaches into two groups based on their main contribution: The first group consists of approaches that deal with adaptive workflow management in general. These works are not applied in cyber-physical environments or deal with autonomic MAPE-K control loops. In the second group, we present approaches that also deal with adaptive workflow management but have a special focus on smart environments. Some of them use autonomic MAPE-K control loops or advanced AI methods for process adaptation, such as automated planning for BPM.

The *ADEPT* framework by Dadam and Reichert (2009) is one of the earliest and most prominent approaches for adaptive workflow management. The *ADEPT* system focuses on process flexibility via process model evolution and ad-hoc changes of process instances. Thereby, monitoring for process errors and adapting a process is mainly performed manually. If an error occurs, the user selects a proper ad-hoc change such as inserting or deleting tasks as well as changing their execution order. Weber et al. (2004) present the *CBRFlow* system for adaptive workflow management by using conversational *Case-Based Reasoning (CBR)* (Aamodt & Plaza, 1994). If changes to a workflow become necessary due to exceptions or environmental changes and the deviation is not defined in the workflow model, the user adds a case to the case base by answering corresponding questions. The case from the case base can be retrieved in the future and describes how the situation can be handled, e.g., by skipping a task, if this situation occurs again. To prevent the case base from getting too large, frequently reused cases are abstracted to rules that can be applied to update the underlying workflow model. Pesic and van der Aalst (2006) present a declarative way of modeling workflows by defining constraints between process tasks. Their approach to self-management is to restrict the model to detect more errors, and to loosen or redefine restrictions to make the process model more flexible overall. A similar work is presented by Grumbach and Bergmann (2019) who use CBR for deviation management to enable flexible workflows. In their approach, users can flexibly deviate from predefined workflows enabled by declarative constraints but still get guidance in their work through already experienced situations stored in a CBR system.

The *SitOPT* approach presented by Wieland et al. (2015) applies situation-aware adaptive workflows in the manufacturing domain. So called *Situational-Aware Workflows* can be used in failure situations and describe which actions should be performed to resolve the exception. The situational-aware workflows are stored in a corresponding repository and must be manually modeled by users, which is a time-consuming and knowledge-intensive task. In addition, the approach is similar to other approaches that fully specify the process

model to resolve expected, not unexpected, situations. Similar to this approach, the *Case Management Model and Notation (CMMN)*[8] can be used to specify cases that are applied in certain scenarios. By using the proposed metamodel and the notation to create such cases, it is possible to describe them as a set of actions that should be applied in a particular situation to achieve a desired outcome. Although this is a proper way to describe expected situations, unexpected situations cannot be handled appropriately. In addition, the manual effort for creating cases is a time-consuming and laborious task. Seiger et al. (2019) present the *PROtEUS* system that enables the execution and adaptation of cyber-physical workflows in smart home environments. The approach is similar to our work and applies autonomic MAPE-K control loops to determine if there is a convergence between the assumed cyber process state and the actual physical process state (*Cyber-Physical Consistency*). If this is not the case, resource-based adaptation techniques are used that substitute the acting resource with another functionally equivalent one and, thus, enable self-healing processes. The *SmartPM* system by Marrella et al. (2017) uses automated planning techniques to adapt processes in case of exceptional situations. The approach is applied to knowledge-intensive emergency management processes with ad-hoc exceptions. To determine exceptions during process execution, the expected reality of the process is compared with the physical reality, similar to the concept of *Cyber-Physical Consistency* by Seiger et al. (2018, 2019) and Seiger & Aßmann (2019). If a gap is detected, AI planning is used to create a recovery plan for the problem situation. Rodríguez-Moreno et al. (2007) also present an approach in which planning techniques are used in BPM, i.e., for semi-automatically generating workflows for business process reengineering. However, the approach rather aims at enabling the user to annotate the workflow model in such a way that it can be used for planning, since creating suitable planning domain descriptions is a time-consuming and knowledge-intensive task for a domain expert. The user can then, in turn, be supported in the reengineering task. Another branch of work uses multi-agent techniques for autonomous workflow adaptation. One approach is introduced by Richly et al. (2010) in which the task of monitoring and adapting processes is conducted by using the *Belief–Desire–Intention (BDI)* software model in combination with CBR. Multiple BDI agents continuously monitor the execution of processes and provide feedback to the current state based on already experienced cases of executions.

Compared to the proposed approach, some approaches also apply advanced AI techniques for adaptive workflow management but only a few evaluate their work in cyber-physical environments with real-world run-time behavior. In addition, we present a holistic framework based on using self-managing MAPE-K control loops (Muccini et al., 2016) in combination with BPM, which has been only rarely discussed in past research (Seiger et al., 2019; Seiger & Aßmann, 2019).

3 Implementing MAPE-K control loops for adaptive workflow management

In this section, we present, as part of the *Develop/Build* phase of the DSR (Hevner et al., 2004) methodology, the proposed approach for using MAPE-K control loops in smart environments. The approach is implemented in a physical Fischertechnik smart factory (see Section 2.1) that demonstrates an exemplary application scenario for smart environments.

[8] https://www.omg.org/spec/CMMN/1.1/

However, the approach can also be used in other smart environments in which the presented components are applicable. The architecture contains several components which for their part are independent of each other, i.e., the design of the architecture is similar to a microservice architectural style (Richards & Ford, 2021). An overview of the architecture is given in Section 3.1. Afterwards, we present in Sections 3.2–3.5 the individual phases of MAPE-K control loops (see Section 2.2). For each corresponding phase, we describe how it can be applied in smart environments in general and how the approach is implemented for the examined smart factory setup in detail.

3.1 Architectural overview

The proposed architecture for applying MAPE-K control loops in smart environments consists of several components that are independent of each other and can be called remotely from other outside systems. Figure 6 illustrates the architecture of our MAPE-K approach for cyber-physical environments. As the approach considers a process as managed element for MAPE-K control loops, the architecture consists of a *Workflow Management System* on top of it. The WfMS is one of the central components in the architecture, as it executes the managed element and the *Change Plan* that is created after an iteration of the MAPE-K control loop in the smart environment. Processes are orchestrated in the WfMS by individual web service invocations. For this reason, a *Web Server* with (semantic) web services must be available for remotely accessing the components, i.e., sensors and actuators, of the *Smart Environment*. The IoT sensor data that is generated by the smart environment during process execution is streamed to a *Database*. Based on this database, a *Stream Processing* engine can be used to derive complex events such as situations that influence the execution of the process, e.g., a pressed button during production that decides which branch in the process is further executed (see Fig. 4). Another central component in the architecture is the *Workflow Monitor* that is responsible for checking if the current states correspond to the desired states of the processes that are currently executed in the smart environment. For this purpose, the *Workflow Monitor* receives the states from the WfMS and combines them with complex events from the *Stream Processing* engine. If the *Workflow Monitor* detects mismatches between the current and the desired process state, a request for adaptation is sent to the *Adaptation Engine*. The adaptation engine, in turn, adapts the process and returns

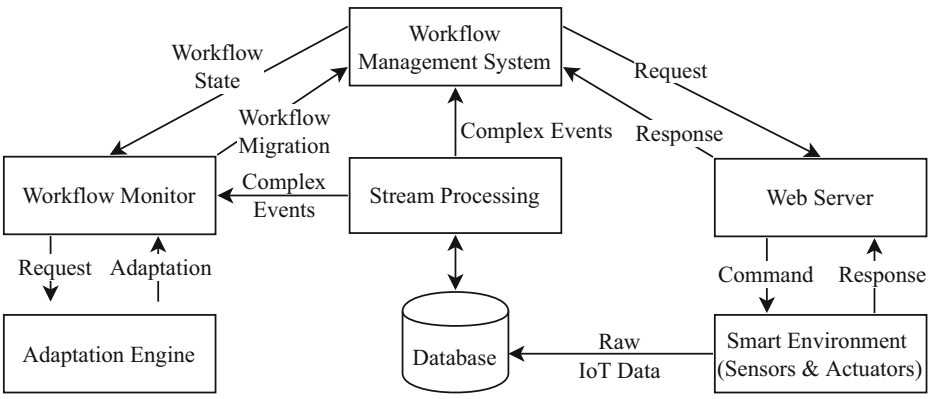

Fig. 6 Architecture of proposed MAPE-K approach

it to the *Workflow Monitor*. The *Workflow Monitor* decides if the adaptation is suitable to handle the current problem situation or not, and if the adaptation is compatible with other currently running processes. If the adaptation is appropriate, the currently running process is finally migrated to a new process model and further executed by the WfMS. In the following, the components and single phases of MAPE-K control loops for smart environments are described in more detail. In addition, we describe how the single phases can be implemented by presenting a specific implementation for MAPE-K control loops in the examined context of a physical smart factory (see Section 2.1).

3.2 Monitor: stream processing

In order to detect changing context that can occur in smart environments, the IoT sensor data must be analyzed in near real-time and high-level events must be derived. These high-level events can be used to guide the execution of the process in the WfMS. In addition, stream processing in smart environments builds the basis for more advanced business process analytics, such as to evaluate the state and performance of currently running process instances (zur Muehlen & Shapiro, 2015). We use *Complex Event Processing (CEP)* methods for monitoring the smart environment in combination with business process management (Soffer et al., 2019). By using CEP, it is possible to detect matching patterns in IoT sensor streams and to derive previously specified events for higher-level systems. Thus, it is possible to react to suddenly occurring misleading situations in smart environments. The applicability of CEP methods ranges from simple events such as changes in resource states to much more complex events such as value changes or other complex patterns in multiple IoT sensor streams (Seiger et al., 2020; 2022).

Example Listing 1 illustrates an exemplary CEP app for the *Siddhi*[9] platform that is used in our implementation to monitor the context of the smart factory, i.e., the states of the machine resources, based on the data handled by Apache Kafka. After the definition of the machine state streams for all machine resources (Lines 3–5), we consider a time window of one second of each machine stream (Line 11 and Line 17) and check if the state in the corresponding time window is unique. If this is the case, the state of the resource has not changed. In contrast, if the resource state has changed, an event is generated and inserted into the *FactoryStatesStream* (Lines 12–13 and Lines 18–19). Finally, this stream consisting of the events with resource state changes is sent to the *Workflow Monitor* or another system specified as publisher in the header (Lines 21–26). For example, if the state changed from *ready* to *inactive*, which indicates a resource breakdown, an event is sent as a *Symptom* to the subsequent *Analyze* phase to check whether the current process execution is affected. The example from Listing 1 could be easily extended and customized to much more complex events with the event query language of *Siddhi*. However, developing suitable CEP apps and identifying relevant events is a complex and knowledge-intensive task and depends strongly on the particular domain. For a more comprehensive overview of different CEP methods for BPM tasks including AI-empowered ones we refer to Soffer et al. (2019).

[9] https://siddhi.io/

```
@App:name("MonitorResourceStates")

define stream DM_2_LastStateStream(state string);
...
define stream OV_1_LastStateStream(state string);

define stream FactoryStatesStream(resource string,
    state string);

@info(name='DM_2_Query')
from DM_2_LastStateStream#unique:
deduplicate(state, 1 sec)
select "DM_2" as resource, state as state
insert into FactoryStatesStream;

@info(name='OV_1_Query')
from OV_1_LastStateStream#unique:
deduplicate(state, 1 sec)
select "OV_1" as resource, state as state
insert into FactoryStatesStream;

@info(name="Message Stream to Workflow Monitor")
@sink(type="http",
publisher.url="http://localhost:8081/
monitoring/factory", method="POST",
    @map(type="json",
    @payload("""{"resource": "{{resource}}", "state":
        "{{state}}"}""")))
```

Listing 1 Exemplary CEP query for monitoring the resource states

3.3 Analyze: workflow monitor

During the *Analyze* phase in the MAPE-K control loop, the current environmental context, e.g., the changed resource states, is further examined and a *Change Request*, if necessary, is created for the following *Plan* phase. For this purpose, the *Workflow Monitor* component is part of the proposed architecture. It has two main sources of information: 1) the WfMS that delivers the process states of the currently executed processes and 2) the *Stream Processing* component that provides information about the context in the smart environment. One main task of the *Workflow Monitor* is to combine both information sources and to analyze in more detail whether the contextual change in the smart environment affects the current process execution. The component then decides whether simple fixes such as repeating a task (*Redo*) are sufficient, or whether more advanced adaptations of the process are required.

To detect deviations during process execution, *Cyber-Physical Consistency* (Seiger et al., 2018, 2019; Seiger & Aßmann, 2019), i.e., whether the expected reality corresponds to the physical reality (Marrella et al., 2017), must be checked. We validate real-world consistency by using semantic annotations from web services (see Section 2.1) used to control the smart factory. Each web service is annotated with the preconditions that must be satisfied for executing the activity and with the effects that must hold after a successful execution. If a precondition or an effect is not satisfied, deviant behavior will occur or has occurred during process execution. The *Workflow Monitor* checks whether the resource state change affects the current process executions by using the data from the WfMS and the semantic annotations of the services. In this context, it can be distinguished between *reactive* and *proactive*

situations: In the first case, a current activity is executed and an error occurs during execution so that the corresponding effects are not fully satisfied. Thus, the *Workflow Monitor* must *react* to this deviant situation. In the second case, the *Workflow Monitor* recognizes an unsatisfied precondition for an activity that is planned to be executed soon. For this purpose, the *Workflow Monitor* specifies a *Change Request* to proactively handle this situation. The approach is generic regarding how to determine deviations between the expected and the physical process state. For example, also other techniques such as online conformance checking methods (e.g., van Zelst et al., 2019) or predictive process monitoring approaches (see Rama-Maneiro et al., 2021 for an overview) can be applied instead of using a checker for semantic annotations.

Based on the determined situation, the *Workflow Monitor* decides whether a simple *Redo* is sufficient or whether more advanced adaptations are required. In the first scenario, the *Plan* phase is executed directly in the workflow monitor component and it communicates the corresponding commands to the WfMS. In the latter case in which more advanced adaptations are necessary, the workflow monitor uses the information provided by the WfMS and the stream processing component and converts it into a corresponding planning problem consisting of the current initial state and a goal state that should be reached by creating a *Change Plan*. More precisely, the *Workflow Monitor* follows three successive steps for doing this: First, the successfully executed tasks of the current process are examined to gather information on the state of the product that is produced. Second, the available resources are determined to get information on the functionalities that can be used for production, e.g., web services that rely on non-responding resources cannot be used during planning. Finally, the currently executed process is further analyzed to determine all production steps that have not been executed successfully or that cannot be executed due to the occurred situation. Based on this, the planning goal state is generated. More precisely, it is determined which activities of the production workflow are affected by the occurred error. These activities and the properties they add to the product, e.g., the drilling of holes, build the basis for creating the planning predicates contained in the planning goal state.

Example As an example from the smart factory, we assume that the *Oven (OV)* in the first shop floor is broken. Figure 7 depicts the process instance created from the process model shown in Fig. 4. Since the oven is defective, the task `OV_1_Burn` cannot be executed (marked with a red cross). However, due to the defect of the oven, the transport from the *High-Bay Warehouse (HBW)* to the oven (see first `VGR_1_PickUpAndTransport` task) as well as the transport from the oven to the human review station (see second `VGR_1_PickUpAndTransport` task) cannot be executed too. In contrast to the `OV_1_Burn` task, where the resource performing the task is not functional, here, it is because a precondition is not fulfilled: More precisely, no transport to or from the oven can be executed, since the oven must be functional for this purpose (see Fig. 3). Both tasks are

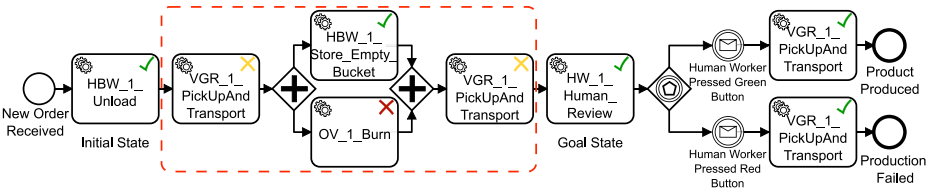

Fig. 7 Running process instance of Fig. 4

thus marked with a yellow cross, since they could generally be executed if the conditions are satisfied. The goal of the process adaptation, i.e., the change plan, is to continue process execution by replacing the part from the first failed tasks to the last failed task with other, suitable tasks that can currently be executed. Suitable tasks are those tasks that lead to the same output, e.g., a burn task can only be replaced by another burn task. In Fig. 7, we mark the part that should be replaced with a dashed red frame. In the following, we describe how the planning problem is automatically generated. Therefore, Listing 2 shows a part of the planning problem expressed in PDDL (see Section 2.3) based on the described running example. To describe the planning problem, the currently executed process and the state of the factory are converted into the corresponding PDDL constructs: First, a key for the workflow (*workflow_1*) is defined as a PDDL object in Line 2. Afterwards, general settings of the factory are initialized, e.g., possible transport routes of the *Vacuum Gripper Robot (VGR)* in the factory. For the sake of simplicity, they are not shown in Listing 2. Please consider that these general settings are not dependent on the planning problem itself, but rather on the smart environment. Lines 6–14 define the current state of the factory. In the example above, all resources except for the oven in the first shop floor (*ov_1*) can be used for production. The state of the executed process, captured by the WfMS, is translated into predicates that describe the current production state (Lines 17–20).

In order to capture this initial state, the task before the part that should be replaced is important (see Fig. 7). Consequently, the `HBW_1_Unload` task is used to derive the initial

```
(define (problem factory-problem) (:domain factory)
(:objects workflow_1 - workflow)
(:init
    ...

    ; resource factory states
    (isReady vgr_1)     (isReady vgr_2)
    (isReady hbw_1)     (isReady hbw_2)
    (isInactive ov_1)   (isReady ov_2)
    (isReady mm_1)      (isReady mm_2)
    (isReady wt_1)      (isReady wt_2)
    (isReady sm_1)      (isReady sm_2)
    (isReady pm_1)      (isReady dm_2)
    (isReady hw_1)

    ; workflow_1 execution state
    (bucketAt hbw_1_pos)
    (bucketAtIsIntendedFor hbw_1_pos workflow_1)
    (isSteelSlab workflow_1)
    (at workflow_1 hbw_1_pos)

    (=(total-cost) 0)

  )
  (:goal (and (at workflow_1 hw_1_pos)
  (bucketAt hbw_1_automatic_pos)
  (isSheetMetal workflow_1)
  (isSheetMetalWithSize middle workflow_1)
  (isSheetMetalWithThickness thick workflow_1))

  (:metric minimize (total-cost))
)
```

Listing 2 Exemplary planning problem for occured failures during manufacturing

state of the planning problem. This results in the predicates that specify that the bucket with the workpiece is currently located at the *High-Bay Warehouse (HBW)* in the first shop floor (*hbw_1_pos*) and that the workpiece is currently not burned. Please note that unprocessed workpieces are automatically considered as steel slabs. For generating the goal state of the planning problem, each failed task is examined, and it is checked which properties it will add to the workpiece if it is executed. The `OV_1_Burn` task burns the workpiece so that it is a sheet metal with a corresponding size and thickness. The other tasks do not add a special property to the workpiece, since they only move it from one position to another or are executed to store the empty bucket in the warehouse (see Fig. 7). For this reason, one part of the goal state is derived from the `OV_1_Burn` task (see Line 27–29).[10] To ensure a semantically correct process after adaptation and, thus, to continue process execution, the first task (`HW_1_Human_Review`) after the part that should be replaced is also used to derive the goal state, i.e., the correct position of the workpiece (see Line 25). In general, it is also possible to generate a planning problem in which the completion of the whole process is captured. In this case, the goal state is created in the same way as described previously, and the part to be replaced is assumed to reach the last task in the process. Although this is a further possibility to solve the problem, it increases the planning effort significantly, since more tasks need to be used during planning. The given metric in Line 31 is used to tell the planner that we expect a plan in which the total cost of all actions is as low as possible.

In addition to the conversion into a planning problem, the planning domain must be specified. Please note that the planning domain represents the general knowledge of the application domain and is therefore not directly dependent on the problem itself. As an example, Listing 3 shows the planning action that corresponds to the semantic service illustrated in Fig. 3. As already mentioned in Section 2.4, it is generally laborious to apply AI planning to real-world problems, since the knowledge acquisition and modeling effort to generate the required fully observable planning domain description is high and, thus, sometimes only incomplete planning domain models can be used (Rodríguez-Moreno et al., 2007; Marrella et al., 2017; Nguyen et al., 2017). For this reason, we use a converter that translates the semantic service architecture into a planning domain description and, thus, significantly reduces the high effort required to generate suitable planning domains by reusing existing knowledge. In Section 2.1, we describe that 256 different parameter configurations exist for the manufacturing capabilities in the smart factory. Since their preconditions and effects change significantly due to the selected parameters, we implement for each possible parameter configuration one web service with its semantic annotations. The resulting 256 services are then converted into corresponding planning actions by a one-to-one mapping, i.e., each web service is mapped to one corresponding planning action.

The conversion to planning actions can either be done in an offline phase before processes are executed or ad-hoc when the *Workflow Monitor* detects an error during process execution. Creating the planning actions ad-hoc has the advantage that some services do not need to be converted, e.g., if the resource executing the service is not available or if they are not relevant to the current situation, which reduces the complexity of the planning problem. On the other hand, an offline conversion reduces the workload for the workflow monitor in failure situations and is suitable for simple domains. The planning action

[10] Please note that we assume the `OV_1_Burn` task is parametrized to produce a medium-sized, thick sheet metal.

```
(:action                                                                    1
    VGR_Transport_Resource_VGR_1_Start_HBW_1_End_OV_1
    :parameters (?workflowID - workflowID)                                  2
    :precondition (and (at ?workflowID hbw_1_pos)                           3
      (isReady vgr_1) (isReady hbw_1)                                       4
      (isReady ov_1)                                                        5
      (bucketAtIsIntendedFor hbw_1_pos ?workflowID))                        6
    :effect (and (at ?workflowID ov_1_pos)                                  7
      (not (at ?workflowID hbw_1_pos))                                      8
      (increase (total-cost) 31))                                           9
)                                                                           10
```

Listing 3 Planning action for the pick up and transport service

illustrated in Listing 3 represents the relevant part of parameters, preconditions, and effects from the semantic service. Please note that the service input parameters (*start*, *end*, and *resource*) that are modeled as instances in the domain ontology FTOnto (Klein et al., 2019) are not converted into planning parameters, since they can be specified more efficiently as constants, e.g., *hbw_1_pos*, and directly used in predicates, e.g., *(at ?workflowID hbw_1_pos)*. However, the reference to the workflow is automatically added to the planning action as the parameter *workflowID*. The time for service execution is considered as the total cost for executing the action. All in all, we convert all 256 services into corresponding planning actions in an offline phase and the *Workflow Monitor* transfers them to the subsequent *Plan* phase together with the planning problem as a *Change Request*.

3.4 Plan: adaptation engine

As stated in Section 2.3, AI planning can be used in BPM to adapt processes. During the *Plan* phase in the MAPE-K control loop, an *Adaptation Engine* is used to generate a *Change Plan* that solves the current problem situation w.r.t. the received *Change Request*. In the architecture, we propose to use AI planning techniques to implement the adaptation engine component. For this purpose, we use a REST endpoint to control automated planners remotely. After receiving the change request with the corresponding planning files from the workflow monitor, the automated planner tries to solve the problem. The generated plans are not checked for validity or further processed in the adaptation engine component, but sent to the workflow monitor that performs these actions. Since the PDDL specification language is used, and it is supported by many state-of-the-art planners, we do not suggest a specific planner.

Example For the exemplary problem described in Section 3.3, two generated plans are illustrated in Fig. 8. The plan on the right side is generated by using a greedy search algorithm and determines a plan with a total cost of 532. It means that the execution of this change plan requires a time of 532 seconds in the smart factory. The plan on the left side is created by using an A* search algorithm with total cost of 447 seconds. In general, the plans differ in their length as well as their parameter configuration. The plan on the right side transports and picks up the workpiece at the drilling machine, whereas the left side plan uses the punching machine, which is closer to the starting position and therefore faster to reach. In addition, the plan on the right side requires four transport actions to reach the oven, whereas the plan on the left side needs only two actions.

(vgr_transport_resource_vgr_1_
start_hbw_end_pm_1_sink workflow_1)

(hbw_store_empty_bucket_resource_
hbw_1 workflow_1)

(vgr_transport_resource_vgr_2_
start_pm_1_sink_end_oven workflow_1)

(ov_burn_resource_ov_2_size_parameter_
thickness_parameter middle thick workflow_1)

(vgr_transport_resource_vgr_2_
start_oven_end_dm_2_sink workflow_1)

(vgr_transport_resource_vgr_1_start_dm_2_
sink_end_human_workstation workflow_1)

; cost = 447 (general cost)

(vgr_transport_resource_vgr_1_
start_hbw_end_dm_2_sink workflow_1)

(hbw_store_empty_bucket_resource_
hbw_1 workflow_1)

(vgr_transport_resource_vgr_1_start_dm_2_
sink_end_human_workstation workflow_1)

(vgr_transport_resource_vgr_1_start_human_
workstation_end_pm_1_sink workflow_1)

(vgr_transport_resource_vgr_2_start_pm_1_
sink_end_oven workflow_1)

(ov_burn_resource_ov_2_size_parameter_
thickness_parameter middle thick workflow_1)

(vgr_transport_resource_vgr_2_
start_oven_end_dm_2_sink workflow_1)

(vgr_transport_resource_vgr_1_start_dm_2_
sink_end_human_workstation workflow_1)

; cost = 532 (general cost)

Fig. 8 Plans generated by A* search (left) and Greedy search (right) for an exemplary planning problem

3.5 Execute: workflow monitor & workflow management system

After receiving the generated plans from the *Adaptation Engine*, the *Execute* phase of the MAPE-K control loop is applied to execute the *Change Plan* in the real-world. For this purpose, the *Workflow Monitor* selects one of the created plans based on certain specified criteria. These could include the costs for executing the plan in the smart environment, the availability of actuators since the state of the real world might have changed again during planning, or the impact of process adaptation on other running processes that can thereafter no longer be executed as planned. Afterwards, the *Workflow Monitor* converts the *Change Plan* into a corresponding sequence of tasks, and adds this sequence to the native BPMN process model by using an *Error Boundary Event*. The *Error Boundary Event* is attached to the failed task in the process.

Example As a running example, the process depicted in Fig. 4 is currently executed in the smart factory. We assume, as in previous sections, that the oven on the first shop floor is defective, which leads to a termination of the process execution at the first `VGR_1_PickUpAndTransport` task. This is because the corresponding transport precondition is not satisfied, which requires that the oven must be functional. For the same reason, the second `VGR_1_PickUpAndTransport` task that transports the workpiece from the oven to the human workstation is also not executable. Both tasks are thus marked with a yellow cross that indicates that the performing resource, e.g., the VGR, is active but the preconditions are not satisfied. In contrast to this, the task `OV_1_Burn` cannot be executed since the performing resource is not functional. Thus, we mark this task with a red cross. Figure 9 depicts a part of the process from Fig. 4 and the left side optimal *Change Plan* (see Fig. 8) with a modeled BPMN error event. Since the task `VGR_1_PickUpAndTransport` that should transport the workpiece to the oven in the first shop floor failed during process execution due to the not satisfied semantic service precondition, the *Error Boundary Event*

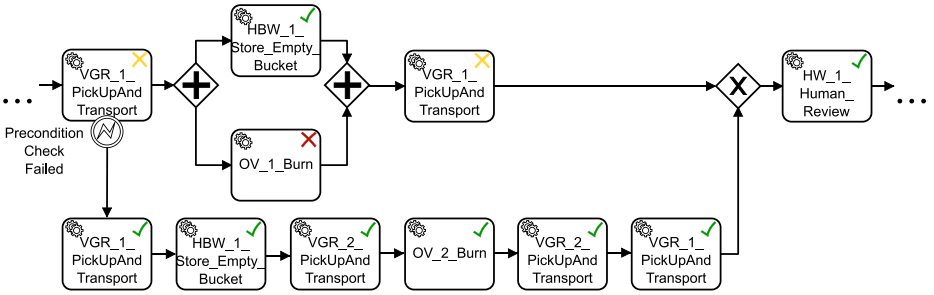

Fig. 9 Adapted sheet metal manufacturing process

is added to this task. The adapted process hence contains the original and planned process sequence (upper part of the visualization) and the inserted change plan (lower part). We do not remove the currently non-executable sequence from the process model to retain the knowledge of this modeled process along with the sequence that can be executed if a failure occurs at the task `VGR_1_PickUpAndTransport`. If the same failure occurs in the future, the knowledge of the performed adaptation and the alternative sequence can be reused.

Before the adapted process model is sent to the WfMS and a migration of the currently running process instance to the updated model is performed, the process is checked for syntactic and semantic correctness as well as for executability. More precisely, it is checked whether the task sequence and parameterization of tasks for execution in the smart factory are correct. In addition, it is also checked whether all required resources are available for execution. During the execution of the *Change Plan*, a further exceptional situation can occur, which leads to an additional run of the MAPE-K control loop. Consequently, a further *Error Boundary Event* is added to the failed task in the native *Change Plan*. All in all, this procedure is analogous to the procedure already described and illustrated in Fig. 9.

4 Experimental evaluation

In this section, we describe as part (*Justify/Evaluate*) of the DSR methodology (Hevner et al., 2004) the experimental evaluation of the proposed approach, which enables self-management in the physical Fischertechnik smart factory (see Section 2.1). We evaluate all phases of the MAPE-K control loop by manufacturing processes that are executed in the smart factory. The considered scenario is a production component, i.e., a machine resource, that is suddenly out of order and the MAPE-K implementation reacts to this by monitoring the changed environmental state and, in turn, the consequences for currently running processes (see Section 3.2). Based on this, a planning problem is generated (see Section 3.3), a suitable plan to adapt the process (see Section 3.4) is searched for, and the execution with the changed, migrated process in the WfMS is resumed (see Section 3.5). In the following, the experimental setup is introduced (see Section 4.1). Section 4.2 examines the results of the evaluation w.r.t. the self-management capabilities as well as the practical applicability and usefulness of the proposed approach. Finally, we discuss concluding remarks in Section 4.3.

4.1 Experimental setup

We assume that each of the *two shop floors* in the previously introduced Fischertechnik smart factory (see Section 2.1) simulates an individual and partly independent production

line of a real-world production environment in the experiments. The used processes deal with sheet metal production, as already introduced in Section 2.1. Thereby, each shop floor executes a single production process at a time, which resembles mass production with repeating processes. All resources continuously push their state to the database such that it can be used by the stream processing engine. The production in the two shop floors remains independent of each other until a problem is encountered in the IoT sensor streams. In this case, the shop floor where no problem is present can be used to execute the replanned process from the other shop floor. To obtain real world (re-)planning problems, a sudden failure during the production (out of order) is injected in a manufacturing resource of the smart factory. We use a *failure generation engine* for this task that can be customized to produce failures according to certain parameters. The output of the engine is the shop floor which is affected by the failure, the affected manufacturing resource on this floor,[11] and the duration that this resource is turned off.[12] We parameterized the engine to have at most two failures simultaneously, where each individual failure lasts at least 25 seconds and at most 45 seconds. After this time period, other randomly chosen resources are determined for failure generation, i.e., during the entire runtime of the processes (approx. 6 to 9 minutes for each run) several machine resource failures are simulated. Hence, we only experiment with reactive failure recovery and do not regard the proactive variant.

After a failure occurs in the smart factory, we stop all processes and capture their current state to use them as the initial planning state. The further planned production sequence and the final workpiece properties are also captured to be used as the planning goal state, analogous to the example in Section 3.3 and Listing 2. We utilize *four different production processes*[13] throughout the experiment that are executed in pairs of two. The processes P 1.1 and P 1.2 are executed on the first and second shop floor, respectively, and both contain 12 tasks. The processes P 2.1 and P 2.2 are also executed on both shop floors but contain 16 and 19 tasks, respectively, to increase the complexity of the planning problems. Thereby, P 1.1 and P 1.2 use 6 out of 7 different resources and P 2.1 and P 2.2 use all 7 different resources in the respective shop floor. We generate *20 random problems* given these processes (10 for each pair of processes) that should be solved by the proposed MAPE-K approach to continue production based on the current production state of the processes and by considering the environmental context, i.e., the failed machine resources and the machine resources currently still functional. The defective components are marked in the planning problem as *inactive* accordingly so that they could not be used during change plan generation. Thus, other components that are able to perform the required activities to reach the desired goal must be used by adapting the production processes, possibly by using production capacities of the other production line. As the adaptation engine (see Sections 3.1 and 3.4) for the experiments, we use Fast Downward (FD) (Helmert, 2006), which is a state-of-the-art planner. FD can be configured to use different search algorithms with different search heuristics during planning. We use two different configurations: A* search with the landmark-cut heuristic (*lmcut*) (Helmert & Domshlak, 2009) and a lazy greedy best-first search with a combination of the fast-forward (*hff*) (Hoffmann, 2001) and the context-enhanced additive heuristic (*hcea*) (Helmert & Geffner, 2008). The utilized planning domain is generated in

[11] We have excluded the vacuum gripper robots and the high-bay warehouses as central components since their failure could never be recovered and would lead to a production stop.

[12] The problems generated also occur regularly in the smart factory even without simulation, e.g., a defect caused by the light barrier not working correctly.

[13] See Appendix A for an example of one of these processes.

an offline phase before the experiments by a converter based on the 256 semantic services and the domain ontology FTOnto (see Section 3.3). Moreover, the domain is annotated with 27 relational predicates and one functional numeric fluent, i.e., the total-cost function[14] (see Listing 4 in Appendix A for a list of all used predicates). A virtual machine with an Intel Xeon Gold 6130 CPU (8 virtual cores) with 2.10 GHz (turboboost 3.70 GHz) with 8 GB RAM, running Ubuntu 16.04 is used for the experiments.

4.2 Experimental results

The experimental results are focused on the outcome of the plan phase of the MAPE-K control loop, since the result of this phase requires the proper detection and analysis of the failure beforehand. We do not present in-depth results of the monitor phase, i.e., failure detection, as the approach was able to detect all generated random failures.

Similar to Marrella et al. (2017), we examine the generated plans (see Table 1) regarding the total cost, i.e., the time to execute the change plan in the smart factory, and the length of the created plans as well as the time for planning. The plan length is the number of actions, and the total cost is the sum of the individual actions' cost (in seconds). The time is further split into the plan time, i.e., solving the specific planning problem with FD, and the total time, i.e., reporting the planning problem until receiving the *Change Plan* via REST (both in milliseconds). The results for the 20 planning problems are grouped by the A* plan length and averaged per group, resulting in eight groups. Additionally, we show the number of cases in each group (see leftmost column), e.g., four problems with an A* plan length of five. Note that four detected problems could not be solved at all, since no other machine resources are available for recovery. This is the reason why only 16 problems are assessed in Table 1. We compare planning solutions created by A* and greedy search. The results of A* are the optimal results w.r.t. plan length and total costs, where a faster but not optimal greedy search is compared to.

The total times of both approaches are very similar. On average, there is no difference in time, but individual examples show a difference of up to 15%. These variations are most likely caused by the overhead that is introduced by using the proposed REST-based setup and the underlying network communication. When inspecting the plain planning times, differences are more noticeable: Greedy search has shorter plan times across all problems, with an avg. decrease of 43% (range of 25% to 67%). This shows that greedy is significantly outperforming A* search in plan time. The reduced plan time, in turn, has the drawback of suboptimal plans. The produced plans of greedy search are 11% longer than the optimal plans, on average. However, the total cost of the produced plans is more meaningful since it ultimately determines the runtime and, thus, the overall efficiency of the process in the smart factory. The average cost of generated plans by greedy is 13% higher than the optimal cost. These values range from no cost increase for six problems to an increase of 105% in one case. The overall results underline the individual strengths of greedy and A* search, with greedy being faster but not guaranteeing optimal results as A* does. The response delays of both approaches to failures can be considered as real-time, with autonomously recovered processes almost immediately. This shows that the approach allows self-management of cyber-physical processes in smart factories with low time efforts and with no human intervention.

[14]The PDDL 2.1 planning domain, all randomly generated problems, and the used example in Section 3 are available at https://gitlab.rlp.net/iot-lab-uni-trier/jiis-2022-journal.

Table 1 Experimental results of A* search (optimal) and Greedy search (suboptimal)

	A* Search				Greedy Search			
Problems in Group	Plan Length	Total Cost [s]	Plan Time [ms]	Total Time [ms]	Plan Length	Total Cost [s]	Plan Time [ms]	Total Time [ms]
4	5	416.00	8.43	355.00	5.00	416.00	4.43	354.00
4	6	453.50	14.73	373.75	7.00	500.00	6.08	361.25
1	8	319.00	7.60	341.00	9.00	353.00	3.60	389.00
3	9	315.67	8.77	347.33	10.33	330.67	3.57	349.67
1	10	685.00	24.00	352.00	12.00	821.00	4.00	357.00
1	11	381.00	10.40	360.00	19.00	782.00	7.60	348.00
1	13	674.00	28.00	385.00	18.00	921.00	14.40	390.00
1	14	699.00	28.00	355.00	20.00	1004.00	8.00	381.00

In addition to the quantitative evaluation results, we also want to discuss the practical applicability and usefulness of the proposed approach as part of a qualitative evaluation. For using the approach in other smart environments or in practice, a service-oriented architecture is inevitably required. Current research already investigates this topic, for example, with implementations of asset administration shells (Perzylo et al., 2019) that provide similar functionalities as the used service-oriented architecture in the proposed approach. However, it is difficult to assess whether such architectures have already paved the way to corporate practice. The service-oriented architecture that enables the control in a more process-oriented way is combined with a WfMS, a *Workflow Monitor*, a *Stream Processing* engine, and an *Adaptation Engine*. Many companies already use WfMSs for their operational processes, which can be reused for the proposed approach. Depending on the configuration of the smart environment, effort is required to define suitable CEP queries to detect deviant behavior during process execution. As stated in Section 3.2, this depends heavily on the domain, which is why it is difficult to make a concrete estimation of the costs and effort here. However, the effort required to implement a corresponding *Workflow Monitor* and an *Adaptation Engine* is manageable: The *Adaptation Engine* can be realized by established and openly available planners. To integrate a *Workflow Monitor*, data from the WfMS must be captured, which is typically possible by using REST-APIs. This data is combined with the events stemming from the stream processing engine and further aggregated and analyzed. In addition, the adaptation must be initiated in the *Adaptation Engine* based on the data provided by the workflow monitor. All in all, the use of the proposed approach is possible in practice with a manageable effort, since existing approaches and implementations can be reused. However, knowledge of domain experts is inevitably required to be able to use the approach appropriately.

4.3 Concluding remarks

We want to particularly discuss *scalability* and *process scheduling* in the following: The presented approach is generic and generally scalable w.r.t. cyber-physical environments with more demanding characteristics such as a higher number of concurrent process executions or resources. These factors increase the size of the planning problems since more actions are incorporated during planning. This inevitably increases the overall complexity of the problem-solving procedure, and planning can become a performance bottleneck (Marrella

et al., 2017; Bylander, 1991; Borrajo et al., 2014). A possible solution is the use of non-optimal planning methods such as greedy search. Another possibility is to reuse knowledge from existing (executed) processes or plans as solution candidates for solving upcoming process adaptations by using *Case-Based Reasoning (CBR)* (Weber et al., 2004; Aamodt & Plaza, 1994) or *Case-Based Planning* (Borrajo et al., 2014). Instead of generating a change plan for recovering faulty processes from scratch, best-practice fragments can be reused by the adaptation engine (Malburg & Bergmann, 2022), which can drastically reduce adaptation time and enhance the quality of adaptation results (cf. Malburg et al., 2023). These fragments could, for instance, stem from shop floor workers with deep knowledge about the production and process adaptations that need to be performed if certain situations occur (Malburg & Bergmann, 2022). Another aspect to discuss is process scheduling, which has not been considered in the experiments to reduce complexity. Scheduling is needed to decide whether and when a replanned process can be executed in an environment where other processes are already running. To enable planning that considers these circumstances, a temporal planner (e.g., Eyerich et al., 2009) is needed. Temporal planning would increase the problem of computational complexity even more, as first experiments have already shown. However, we assume in the conducted experiments that higher-level systems such as manufacturing execution systems or enterprise resource planning systems take care of scheduling the tasks within production, which is inevitable in cyber-physical environments to manage proper access to constrained resources (Seiger et al., 2022; Malburg et al., 2020a).

5 Conclusion and future work

In this paper, we present an approach for self-management of processes in cyber-physical environments. The proposed approach is based on MAPE-K control loops that are widely used in cyber-physical systems (Muccini et al., 2016). For monitoring the environment, we apply *Complex Event Processing* methods to derive higher-level events from IoT sensor data and combine them with the process execution states from the *Workflow Management System*. Based on this information, failures can be detected during process execution, which in turn are resolved by applying automated planning techniques. We conduct an experimental evaluation with a physical smart factory (see Section 2.1) to improve on current research that often does not address cyber-physical environments and, instead, is mainly based on artificially generated IoT data. The evaluation shows that the approach can be used in a real-world smart factory and is able to detect failures and to solve them autonomously in near real-time with considerable results. Therefore, the presented work also contributes to more advanced and autonomous process analytics in smart environments.

In future work, we want to address the aspects discussed in Section 4.3 in more detail. Particularly, the combination of AI planning with *Case-Based Reasoning* (Aamodt & Plaza, 1994; Borrajo et al., 2014) promises further potential for improvements (cf. Malburg et al., 2023; Malburg & Bergmann, 2022). Such improvements are especially needed if the planning problems increase in complexity, for example, in real production lines with even more complex domains. In this context, we examine possibilities to artificially increase the complexity of the planning domain used for research. This can be achieved, for instance, by expanding the smart factory to include additional components or by integrating the smart factory with a digital twin of itself which combines the benefits of the physical and virtual worlds.

Appendix A: Additional information on the experimental evaluation

A.1 Exemplary process

Figure 10 shows P 2.2 as one of the four manufacturing processes that are used in the experimental evaluation (see Section 4). The process contains 19 tasks and has the goal of producing a piece of sheet metal with several properties, e.g., burned, milled, deburred, etc. For simplification reasons these processes do not contain control-flow blocks (*XOR*, *OR*, and *AND* gateways) which means that all tasks are executed in a single sequence. The other three processes used in the experiments are similar to this one.

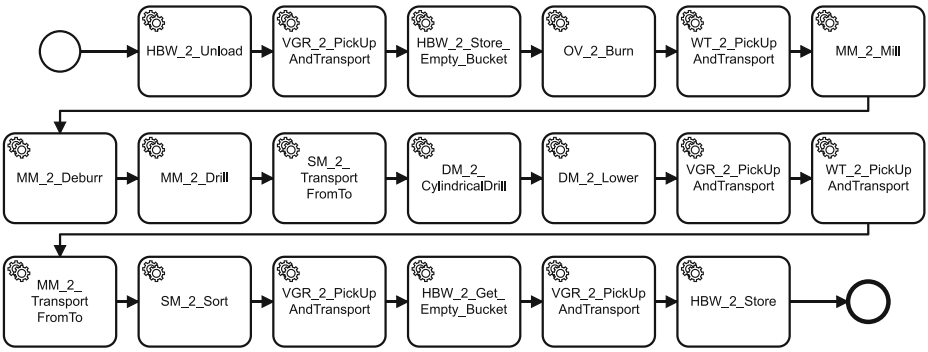

Fig. 10 Manufacturing process p 2.2 used in the experimental evaluation

A.2 Failure generation

As introduced in Section 4.1, we use a *failure generation engine* that simulates resource failures throughout process execution on the shop floor. This engine simply deactivates resources such that they cannot be used anymore. Figure 11 illustrates an example of generated failures over the time span of 10 minutes for a process execution of P 2.2 (see Fig. 10). Specifically, it displays the number of machines with failures (vertical axis) within blocks of 60 seconds (horizontal axis). The depiction indicates that the number of failed resources within a time frame of 60 seconds is always between 3 and 5. The average number of failures per block is 3.7 and the average duration of each failure is 35.81 seconds. The average time between the appearance of two consecutive failures is 20.47 seconds.

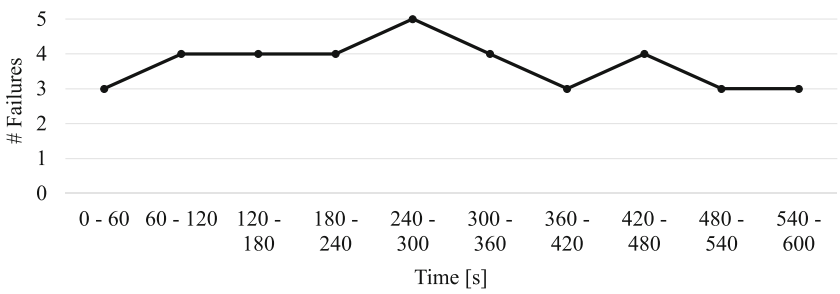

Fig. 11 Output of the failure generation engine

A.3 Planning domain description

In the experiments, we use an automatically generated planning domain description consisting of 27 relational predicates and one functional numeric fluent. To illustrate the individual

```
(define (domain factory)
(:requirements :typing :action-costs
    :negative-preconditions)

 (:types
  ...
 )

 (:predicates
  (at ?workflowID - workflowID ?p - position)
  (isReady ?resource - resource)
  (isInactive ?resource - resource)
  (isSteelSlab ?workflowID - workflowID)
  (isMilled ?workflowID - workflowID)
  (hasGills ?workflowID - workflowID)
  (hasGillsWithQuantity ?quantity - gills_quantity
      ?workflowID - workflowID)
  (hasRecesses ?workflowID - workflowID)
  (hasRecessesWithQuantity ?quantity -
      recesses_quantity ?workflowID - workflowID)
  (hasRibbings ?workflowID - workflowID)
  (hasRibbingsWithQuantity ?quantity -
      ribbings_quantity ?workflowID - workflowID)
  (isSheetMetalWithSize ?size - sheet_size
     ?workflowID - workflowID)
  (isSheetMetalWithThickness ?thickness -
      sheet_thickness ?workflowID - workflowID)
  (isSheetMetal ?workflowID - workflowID)
  (isSorted ?workflowID - workflowID)
  (isDeburred ?workflowID - workflowID)
  (isDrilledWithSize ?size - holes_size ?workflowID -
      workflowID)
  (isDrilledWithQuantity ?quantity - holes_quantity
      ?workflowID - workflowID)
  (isCylindricalDrilledWithSize ?size - holes_size
      ?workflowID - workflowID)
  (isCylindricalDrilledWithQuantity ?quantity -
      holes_quantity ?workflowID - workflowID)
  (isDrilled ?workflowID - workflowID)
  (bucketAt ?p - position)
  (bucketAtIsIntendedFor ?p - position ?workflowID
     - workflowID)
  (isCylindricalDrilled ?workflowID - workflowID)
  (isLowered ?workflowID - workflowID)
  (isReviewed ?workflowID - workflowID)
  (isApplicable ?value - object ?resource -
      resource)
 )

 (:functions
  (total-cost)
 )
 ...
)
```

Listing 4 Part of the planning domain descirption used in the experimental evaluation

relational predicates, Listing 4 depicts a part of the planning domain description in which the predicates and the used function are specified. The relational predicate `isApplicable` is used to specify whether a resource is able to produce a certain workpiece property, e.g., `(isApplicable holes_quantity_8 dm_2)` specifies that the drilling machine in the second shop floor can drill a workpiece with 8 holes.

Funding This work is funded by the Federal Ministry for Economic Affairs and Climate Action under grant No. 01MD22002C *EASY*. Open Access funding enabled and organized by Projekt DEAL.

Availability of data and materials The data and results of the experiment are available from the corresponding author on request. The experimental evaluation can be reproduced by using the PDDL files that are available at https://gitlab.rlp.net/iot-lab-uni-trier/jiis-2022-journal.

Declarations

Conflict of interest/Competing interests The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, *7*(1), 39–59.

Abele, E., Chryssolouris, G., Sihn, W., & et al. (2017). Learning factories for future oriented research and education in manufacturing. *CIRP Annals*, *66*(2), 803–826.

Borrajo, D., Roubícková, A., & Serina, I. (2014). Progress in case-based planning. *ACM Computing Surveys*, *47*(2), 35:1–35:39.

Bylander, T. (1991). Complexity results for planning. In *12th IJCAI* (pp. 274–279). Morgan Kaufmann.

Dadam, P., & Reichert, M. (2009). The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - Research and Development*, *23*(2), 81–97.

de Masellis, R., Di Francescomarino, C., Ghidini, C., & et al. (2022). Solving reachability problems on data-aware workflows. *Expert Systems with Applications*, *189*, 116059.

Dumas, M., La Rosa, M., Mendling, J., & et al. (2018). *Fundamentals of business process management*. Berlin: Springer.

Eyerich, P., Mattmüller, R., & Röger, G. (2009). Using the context-enhanced additive heuristic for temporal and numeric planning. In *19th ICAPS*. AAAI.

Grumbach, L., & Bergmann, R. (2019). Towards case-based deviation management for flexible workflows. In *21st LWDA*, (Vol. 2454 pp. 241–252). CEUR-WS.org.

Haslum, P., Lipovetzky, N., Magazzeni, D., & et al. (2019). An introduction to the planning domain definition language. Synth. Lect. on Artif. Intell and Mach Learn.. Morgan & Claypool.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: what's the difference anyway? In *19th ICAPS* (pp. 162–169). AAAI.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *18th ICAPS* (pp. 140–147). AAAI.

Hevner, A. R., March, S. T., Park, J., & et al. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105.

Hoffmann, J. (2001). FF: the fast-forward planning system. *AI Magazine*, *22*, 57–62.

Hoffmann, M., Malburg, L., & Bergmann, R. (2022). ProGAN: toward a framework for process monitoring and flexibility by change via generative adversarial networks. In *BPM Workshops, LNBIP* (vol. 436 pp. 43–55). Springer.

IBM (2006). An architectural blueprint for autonomic computing: Autonomic Computing White Paper.

Janiesch, C., Koschmider, A., Mecella, M., & et al. (2020). The internet of things meets business process management: a manifesto. *IEEE Transactions on Systems, Man, and Cybernetics*, *6*(4), 34–44.

Klein, P., & Bergmann, R. (2019). Generation of complex data for AI-based predictive maintenance research with a physical factory model. In *16th ICINCO* (pp. 40–50). SciTePress.

Klein, P., Malburg, L., & Bergmann, R. (2019). FTOnto: a domain ontology for a fischertechnik simulation production factory by reusing existing ontologies. In *21st LWDA*, (Vol. 2454 pp. 253–264). CEUR-WS.org.

Malburg, L., & Bergmann, R. (2022). Towards adaptive workflow management by case-based reasoning and automated planning. In *30th ICCBR Workshops. CEUR-WS.org*. Accepted for Publication.

Malburg, L., Klein, P., & Bergmann, R. (2020a). Semantic web services for AI-research with physical factory simulation models in industry 4.0. In *1st IN4PL* (pp. 32–43). SciTePress.

Malburg, L., Seiger, R., Bergmann, R., & et al. (2020b). Using physical factory simulation models for business process management research. In *BPM workshops, LNBIP,* (vol. 397 pp. 95–107). Springer.

Malburg, L., Rieder, M. P., Seiger, R., & et al. (2021). Object detection for smart factory processes by machine learning. *Procedia Computer Science*, *184*, 581–588.

Malburg, L., Brand, F., & Bergmann, R. (2023). Adaptive management of cyber-physical workflows by means of case-based reasoning and automated planning. In *26th EDOC workshops*. Springer, LNBIP, Accepted for Publication.

Marrella, A. (2017). What automated planning can do for business process management. In *BPM workshops, LNBIP,* (vol. 308 pp. 7–19). Springer.

Marrella, A. (2019). Automated planning for business process management. *Journal on Data Semantics*, *8*(2), 79–98.

Marrella, A., Mecella, M., & Sardiña, S. (2017). Intelligent process adaptation in the smartPM system. *ACM Transactions on Intelligent Systems and Technology*, *8*(2), 25:1–25:43.

McDermott, D. V., Ghallab, M., Howe, A., & et al. (1998). PDDL - the planning domain definition language: Technical Report CVC TR-98-003/DCS TR-1165.

Monostori, L. (2014). Cyber-physical production systems: roots, expectations and R&D challenges. *Procedia CIRP*, *17*, 9–13.

Muccini, H., Sharaf, M., & Weyns, D. (2016). Self-adaptation for cyber-physical systems: a systematic literature review. In *11th SEAMS* (pp. 75–81). ACM Press.

Nguyen, T. A., Sreedharan, S., & Kambhampati, S. (2017). Robust planning with incomplete domain models. *Artificial Intelligence*, *245*, 134–161.

Perzylo, A., Grothoff, J., Lucio, L., & et al. (2019). Capability-based semantic interoperability of manufacturing resources: A BaSys 4.0 perspective. *IFAC-PapersOnLine*, *52*(13), 1590–1596. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM.

Pesic, M., & van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. In *BPM workshops, LNCS,* (vol. 4103 pp. 169–180). Springer.

Prinz, C., Morlock, F., Freith, S., & et al. (2016). Learning factory modules for smart factories in industrie 4.0. *Procedia CIRP*, *54*, 113–118.

Rama-Maneiro, E., Vidal, J., & Lama, M. (2021). Deep learning for predictive business process monitoring: Review and Benchmark. *IEEE Transactions on Services Computing*.

Reichert, M., & Weber, B. (2012). *Enabling flexibility in process-aware information systems - challenges, methods, technologies*. Berlin: Springer.

Richards, M., & Ford, N. (2021). *Fundamentals of software architecture*. 1st edn. O'Reilly Media Company.

Richly, S., Schmidt, S., & Aßmann, U. (2010). A semantic-BDI-based approach to realize cooperative, reflexive workflows. In *8th WCICA* (pp. 1680–1685). IEEE.

Rodríguez-Moreno, M. D., Borrajo, D., Cesta, A., & et al. (2007). Integrating planning and scheduling in workflow domains. *Expert Systems with Applications*, *33*(2), 389–406.

Schönig, S., Ackermann, L., Jablonski, S., & et al. (2020). IoT meets BPM: a bidirectional communication architecture for IoT-aware process execution. *Software and Systems Modeling*, *19*(6), 1443–1459.

Seiger, R., & Aßmann, U. (2019). Consistency and synchronization for workflows in cyber-physical systems. In *10th ICCPS* (pp. 312–313). ACM.

Seiger, R., Huber, S., & Schlegel, T. (2018). Toward an execution system for self-healing workflows in cyber-physical systems. *Software and Systems Modeling*, *17*(2), 551–572.

Seiger, R., Huber, S., Heisig, P., & et al. (2019). Toward a framework for self-adaptive workflows in cyber-physical systems. *Software and Systems Modeling*, *18*(2), 1117–1134.

Seiger, R., Zerbato, F., Burattin, A., & et al. (2020). Towards IoT-driven process event log generation for conformance checking in smart factories. In *24th EDOC workshops* (pp. 20–26). IEEE.

Seiger, R., Malburg, L., Weber, B., & et al. (2022). Integrating process management and event processing in smart factories: A systems architecture and use cases. *Journal of Manufacturing Systems*, *63*, 575–592.

Simons, S., Abé, P., & Neser, S. (2017). Learning in the AutFab – the fully automated industrie 4.0 learning factory of the university of applied sciences Darmstadt. *Procedia Manufacturing*, *9*, 81–88.

Soffer, P., Hinze, A., Koschmider, A., & et al. (2019). From event streams to process models and back: Challenges and opportunities. *Information Systems*, *81*, 181–200.

van Zelst, S. J., Bolt, A., Hassani, M., & et al. (2019). Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, *8*(3), 269–284.

Weber, B., Wild, W., & Breu, R. (2004). CBRFlow: enabling adaptive workflow management through conversational case-based reasoning. In *7th ECCBR, LNCS,* (vol. 3155 pp. 434–448). Springer.

Wieland, M., Schwarz, H., Breitenbucher, U., & et al. (2015). Towards situation-aware adaptive workflows: SitOPT - A general purpose situation-aware workflow management system. In *PerCom workshops* (pp. 32–37). IEEE.

zur Muehlen, M., & Shapiro, R. (2015). Business process analytics. In *Handbook on business process management 2* (pp. 243–263). Springer.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.