Enriching Process Models with Relevant Process Details for Flexible Human-Robot Teaming

Myriel Fichtner(✉), Sascha Sucker, Dominik Riedelbauch, Stefan Jablonski, and Dominik Henrich

University of Bayreuth, Universitätsstrasse 30, 95447 Bayreuth, Germany
{myriel.fichtner,sascha.sucker,dominik.riedelbauch,stefan.jablonski,dominik.henrich}@uni-bayreuth.de

Abstract. Human-robot teaming is crucial for future automation in small and medium enterprises. In that context, domain-specific process models are used as an intuitive description of work to share between two agents. Process designers usually introduce a certain degree of abstraction into the models. This way, models are better to trace for humans, and a single model can moreover enable flexibility by capturing several process variations. However, abstraction can lead to unintentional omission of information (e.g., experience of skilled workers). This may impair the quality of process results. To balance the trade-off between model readability and flexibility, we contribute a novel human-robot teaming approach with incremental learning of relevant process details (RPDs). RPDs are extracted from imagery during process execution and used to enrich an integrated process model which unifies human worker instruction and robot programming. Experiments based on two use cases demonstrate the practical feasibility and scalability of our approach.

Keywords: Process Model Optimization · Task Annotation · Explanation Models · Intelligent Robots · Process Variety · Product Variety

1 Introduction

The demographic change and a trend towards small-batch production of goods with high variability pose new challenges to the future of manufacturing systems. Particularly when using domain-specific process models to describe workflows in manufacturing settings, highly varying processes require the inclusion of many specific alternatives. This can lead to large and hardly traceable process models, which can only be created with high effort. Since human-robot collaboration is considered a key enabler of partial automation in small and medium enterprises, this issue relates to process models for instructing robots and humans alike [15]. It is usually solved through abstraction: (i) In the context of robotics, we have proposed a *graphical robot programming* method based on *precedence graphs* with

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 563, pp. 249–269, 2024.
https://doi.org/10.1007/978-3-031-54531-3_14

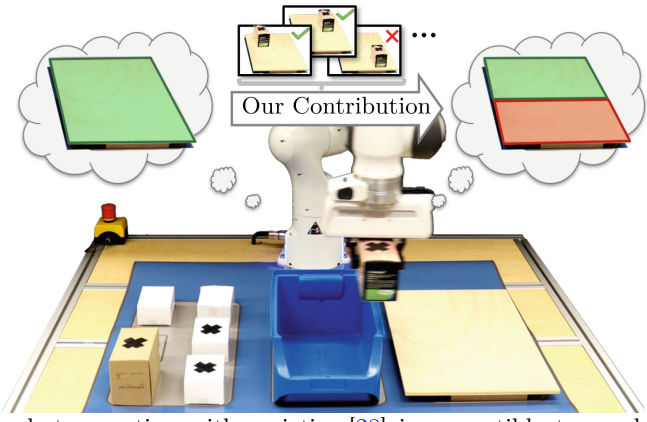


Fig. 1. The robot execution with varieties [28] is susceptible to overly coarse task modeling (upper left: no restrictions on the goal location (green)). We present an approach to specify these tasks with RPDs based on image extraction [11] resulting in increased process quality (upper right: the goal location is restricted, i.e., locations that lead to reduced process success are excluded (red)). (Color figure online)

generalized skill templates [28]. Contrasting to traditional robot programming, our precedence graph-based approach enables quick task specification and online adaptation to concrete situations in the robot workspace rather than requiring manual re-programming after each change to a new task variant. (ii) For manual labor tasks to be done by humans, several execution variants of a single process step are often aggregated into one abstract sub-task of a *business process model*. This abstraction maintains the readability of business process models by partly discarding information on process details (e.g., [5,22,23]). In both robot and human task modeling, the success of the process (e.g., in terms of product quality) can be degraded if models are designed too coarsely – this happens whenever *relevant process details* (RPDs) are omitted or inaccessible to the modeler (e.g., experience and best-practices of skilled workers). RPDs carry hidden information, significantly affecting the overall success of a process. This process knowledge must be revealed and incorporated into a model to ensure the accurate execution of a task. Thereby, RPDs can be used as task specifications for a given process model to further refine existing task instructions. For example, one RPD may prescribe a concrete position on a workbench where an object has to be placed for successful task execution. Our approach [11] therefore shows how RPDs can automatically be extracted from image data to enrich process models with task annotations, hence balancing the trade-off between model readability and preservation of necessary details.

In this paper, we bridge the gap between process model-based human-to-human and precedence graph-based human-to-robot knowledge transfer: We contribute a novel approach to human-robot collaboration with intrinsically legible task representations enriched by RPDs. This way, humans and robots can rely on domain expert knowledge encoded in RPDs for increased process quality (Fig. 1). The practical feasibility of the approach is demonstrated in two use cases.

2 Background and Related Work

Process models give an overview of the work steps to be done during the execution of a process. They serve as the basis for process execution by human and robot agents. Hence, process modeling is an essential basis of knowledge transfer in production contexts with a broad range of established techniques.

In **Business Process Management (BPM)** [34], process models are primarily intended for human workers. The focus lies on presenting process descriptions in a clear and easy-to-understand manner. To this end, *process modeling languages* as the Business Process Model and Notation (BPMN) [6], the Unified Modeling Language (UML) [30], Event-Driven Process Chains [29], etc., have been proposed. They typically define a set of *modeling elements* to map different aspects of a process. *Modeling guidelines* (e.g., [3,16]) are intended to establish standardized ways for manually composing complex processes from modeling elements. Despite these efforts to structure the model design process, mapping concrete tasks to informative but lean process models for human readers is still challenging. The trade-off between the level of detail and model complexity strongly relies on human intelligence to complete missing information omitted at model design time. Research related to automatic model optimization and improvement addresses this issue. Established approaches support model designers from *representation-related* (e.g., [5,22,23]) and *content-related* (e.g., [1,12,18]) perspectives. Representation-related techniques seek to improve the readability of process models in various ways. This involves accepting the loss of information in favor of model traceability, which is a common reason for the lack of process details. In contrast, content-related techniques refer to the modeled quantity, accuracy, relevance, and order of information. Analysis of RPDs is part of the latter category, which has hardly been addressed in research. Therefore, we have previously introduced a novel approach by employing the *Local Interpretable Model-Agnostic Explanations* method to extract the RPDs essential for process success from labeled imagery of correct/erroneous execution results [11]. The RPDs are then used to enrich process models with human-legible hints without compromising the overall model readability. Thus, process models are optimized from a content- and representation-related perspective.

In the field of **Collaborative Robotics**, process models (often also referred to as *task models*) enable robots to participate in a task. Similar to the BPM domain, complex tasks are composed of robot-executable building blocks following a pre-defined structure, e.g., UML/P state charts [33], precedence graphs [26], or AND/OR trees [8,19]. The work steps are then dispatched to human and robot agents. This leads to a collaborative process by task-sharing (e.g., [8,25]). Robot-readable task models are predominantly created manually by domain experts with *visual programming* techniques [9] (e.g., [20,26,28,31,32]) based on *robot skills* [2]. To keep the task model manageable during programming and to keep the process flexible concerning product and process variety, recent visual programming [20,31] and task sharing approaches [8] including our work [28] mimic the abstraction process as used in business process modeling: work steps are intentionally left partly under-specified by omitting information in gener-

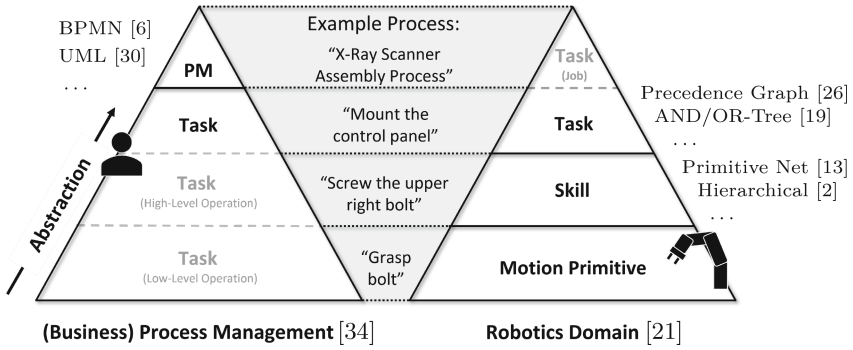


Fig. 2. Mapping of terms stemming from the domains of BPM and robotics.

alized skills (e.g., *"Put the part into the bin"* rather than *"Put the part with id 1 to cartesian position* $(0.4, 0.6, 0.01)$*"*), and the robot system resolves missing information online by reasoning on the scene perceived with cameras just as humans would do with a BPMN model. Yet the decision on how to generalize task models is left to the programmer, thus yielding a potential loss of RPDs.

In conclusion, there is a strong analogy between abstract process models and related challenges in BPM and Collaborative Robotics. We therefore hypothesize that human-robot collaboration approaches would benefit from an analogous enrichment of process models with learnt RPDs as previously enabled by our method to improve business process models [11]. In this paper, we contribute (i) a unified terminology of task modeling in the BPM and Collaborative Robotics domains, (ii) a novel approach to human-robot collaboration based on a single, hierarchical human-robot process model with RPD annotations, and (iii) an incremental learning approach which renders RPDs machine-readable for robots.

3 Terms and Definitions

Connecting the fields of robotics and (Business) Process Management requires the alignment of terminologies and the understanding of how processes and activities are defined and executed in both domains. In the robotics domain (Fig. 2; right), tasks describe actions a robot should perform. A task either refers to a single abstract activity as e.g., *"mount the control panel"*, or to a more complex job that has to be done (e.g., *"X-Ray Scanner Assembly Process"*). Tasks for robots are usually represented by task models as outlined in Sect. 2, with a clearly defined entry and exit point. Since robots require concrete program code to accomplish a task, tasks are decomposed into *skills*, which specify necessary actions on a more fine-grained level [21]. For example, *"screw the upper right bolt"* could be a skill that contributes to the task *"mount the control panel"*. Skills are further decomposed in hierarchies [2] or net structures [13] until ultimately reaching primitives. These *primitives* define the most specific set of actions that correspond e.g., to a single motion command to the robot. They include the

specification of points in the environment that the robot needs to reach as well as gripper actions (e.g., *"grasp bolt"*) [13].

In the BPM domain (Fig. 2; left), process models (PM) describe workflows or processes (e.g., "X-Ray Scanner Assembly Process") to achieve a certain goal [34]. Process models typically consist of an entry point (e.g., a start event in BPMN) defining the beginning, and an exit point (e.g., an end event in BPMN) demarcating the end of the process flow. In between, a series of tasks can be modeled, connected through directional edges to define the control flow. Tasks can be assigned to process participants such as employees or customers (e.g., through pools and swimlanes in BPMN). Furthermore, tasks contain descriptions of activities needed to complete the process. The level of detail in task descriptions is not predetermined and must be decided by the model designer. The designer also has the choice of breaking down a task into smaller sub-tasks. For instance, *"mount the control panel"* can be modeled as a single task or be divided into several sub-tasks describing the task at an operational level. An example high-level operational task is *"screw the upper right bolt"* – a low-level operational task is *"grasp bolt"*. Hence, contrasting to the skill and primitive notions in robotics, the term *"task"* is used independently of the level of detail and type of instruction.

The mapping depicted in Fig. 2 shows that, depending on whether a human or robot performs an activity, a task must be defined in more or less detail for proper execution. Humans apply context knowledge and experience subconsciously. Thus, implicitly necessary substeps of a task are automatically done, and the matching of objects mentioned in the task description with those available in the environment is done intuitively. In contrast, such reasoning cannot be assumed when assigning tasks to a robot but must be considered explicitly during task modeling. This is achieved by the aforementioned division of tasks into skills and primitives. The second aspect, the anchoring [7] of parts, is contained in the skill definition and deals with the mapping of object specifications (describing fictitious objects) to physical objects acquired by the sensors of a robot. In this paper, we call such object specifications *part templates* and physical objects *part states*.

Based on the mapping, we propose a consolidated process model to support collaborations between humans and robots effectively. To this end, *human-readable process models* (HPM) are first created by process experts with according process modeling languages from the BPM domain (e.g., BPMN). In this model, all workers (humans and robots) are considered participants of the process. They are each assigned a swimlane with a pool of tasks to work off (Fig. 3), i.e., task allocation to human or robot resources is achieved by a modeler's decision to assign process steps to swimlanes. This decision is supported by structured criteria from the field of capability-aligned process planning (e.g., [4]). At this stage of the model, tasks assigned to the robot are presented similarly to those assigned to humans. Yet, as previously explained, this description level is insufficient for a robot to perform the task. Therefore, all robot tasks are further enriched with skill- and primitive-based descriptions using visual programming

(Sect. 4.1). This leads to a unified process model composed of dedicated human and robot tasks, each being represented in a manner suitable for the corresponding agent according to Fig. 2 (see PM in Fig. 3 for a visualization). Hereinafter, we will use the term "process model" to refer to this type of unified process model which contains task descriptions for humans, skills for execution by a robot, and the allocation of tasks to humans and robots.

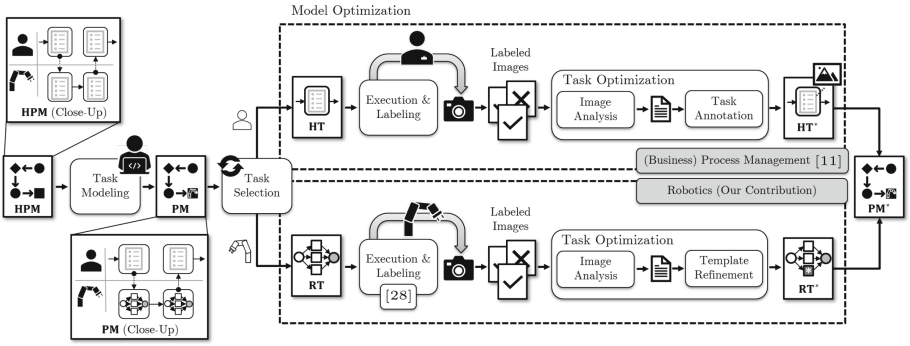
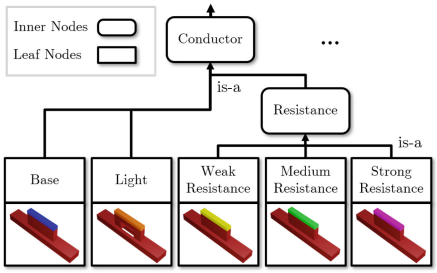


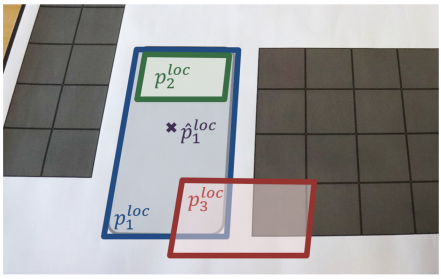
Fig. 3. Adaption of our previous concept [11] to the robotic domain.

4 Extending Unspecified Process Models

Our approach is summarized in Fig. 3. In our previous work [11], the input is a BPMN diagram designed for human workers. In contrast, our new approach additionally allows the input of joint human-robot process models according to Sect. 3. In the *Task Selection* stage, a single task is selected from the input process model. This task is the entry point to *Model Optimization*. Tasks to be executed by humans (HT) trigger the optimization procedure from our previous work [11] (top of Fig. 3). Similarly, for robot tasks (RT) that are expressed as task models, we can utilize our previous procedure by making minor adjustments to accommodate a robotic agent (bottom of Fig. 3). The input process model is executed, and after the execution of the selected task, an image of the workspace is captured. Thereby, in contrast to HTs, RTs are executed as further described in Sect. 4.2. At the end of the process model execution, the recorded image is labeled regarding process success. This is repeated several times until sufficient labeled data is collected. Then, the task is analyzed for relevant process details in the *Task Optimization* step (Sect. 4.3). The output is either a human (HT*) or robot task (RT*) enriched with relevant process details, resulting in an optimized process model (PM*). The cycle can be repeated for further optimizations by returning to the *Task Selection* step.



(a) Excerpt from the part type taxonomy.



(b) Location examples.

Fig. 4. Part types and locations contain varying degrees of ambiguity. The part type taxonomy (a) includes generic inner nodes ('conductor' and 'resistance') that comprise specific types in the leafs (e.g.,'base' or 'light'). Locations (b) encompass generic area descriptions (p_1^{loc} - p_3^{loc}) or specific poses ($\hat{p}_1^{\text{loc}}$).

4.1 Robot Task Modeling

For robot execution, steps in the process model must be mirrored in the robotics domain. We achieve this by visual robot programming of robot-executable precedence graphs with varieties following our previous work [28]. In this approach, *part states* encode physical objects in the workspace, whereas *part templates* describe partially ambiguous requirements to objects (cf. Sect. 3). Both part states and templates have the same features depending on the domain. In this paper, the features are the part type and its location. A *part type* is an entry in a tree-shaped taxonomy with 'is-a'-relations between its set P^{type} of nodes (Fig. 4a). Leaf nodes denominate *specific part types* $\hat{P}^{\text{type}} \subset P^{\text{type}}$ which parts in the physical world can be classified as. When ascending from a leaf towards the root node, inner nodes encode *generic part types*, i.e., fictitious type concepts, which enable variety in the part description. We use the predicate $\text{is\_a}(p_i^{\text{type}}, p_j^{\text{type}})$ to state whether $p_i^{\text{type}} = p_j^{\text{type}}$ or p_i^{type} is a child of p_j^{type}.

Part locations P^{loc} describe the rigid body pose of parts. We distinguish two cases: (i) A *specific location* $\hat{p}^{\text{loc}} \in \hat{P}^{\text{loc}}$ is a rigid body transform ${}^{w}T_{\text{part}} \in \mathbb{R}^{4\times4}$ indicating the part translation and rotation concerning some world frame w (with $\hat{P}^{\text{loc}} \subset P^{\text{loc}}$). (ii) An ambiguous *generic location* $p^{\text{loc}} \in P^{\text{loc}}$ specifies the 3D volume in which a part is expected. Similarly to part types, we use the predicate $\text{is\_in}(p_i^{\text{loc}}, p_j^{\text{loc}})$ to state whether p_i^{loc} is part of the volume p_j^{loc}. This predicate can be applied between specific-specific ($\hat{p}_i^{\text{loc}} \approxeq \hat{p}_j^{\text{loc}}$); specific-generic ($\hat{p}_i^{\text{loc}} \in p_j^{\text{loc}}$); and generic-generic location pairs ($p_i^{\text{loc}} \subseteq p_j^{\text{loc}}$). For example, p_2^{loc} and $\hat{p}_1^{\text{loc}}$ are in p_1^{loc}, whereas p_3^{loc} has no relation to any other location in Fig. 4b.

The *world state* is represented by a set $\hat{P} = \{\hat{p}_1, \hat{p}_2, ...\}$ of part states. Each *part state* $\hat{p}_i = (\hat{p}_i^{\text{type}}, \hat{p}_i^{\text{loc}})$ is an entity with specific part type $\hat{p}_i^{\text{type}}$ at specific location $\hat{p}_i^{\text{loc}}$. Thus, part states contain only well-defined parameters. By contrast, *part templates* $p_i = (p_i^{\text{type}}, p_i^{\text{loc}})$ are generic in their type p_i^{type} and location p_i^{loc}.

A *robot task* is a precedence graph $T = (S, \prec_S, P)$ composed of partially ordered skills $S = \{s_1, s_2, ...\}$ to manipulate a set P of part templates. The

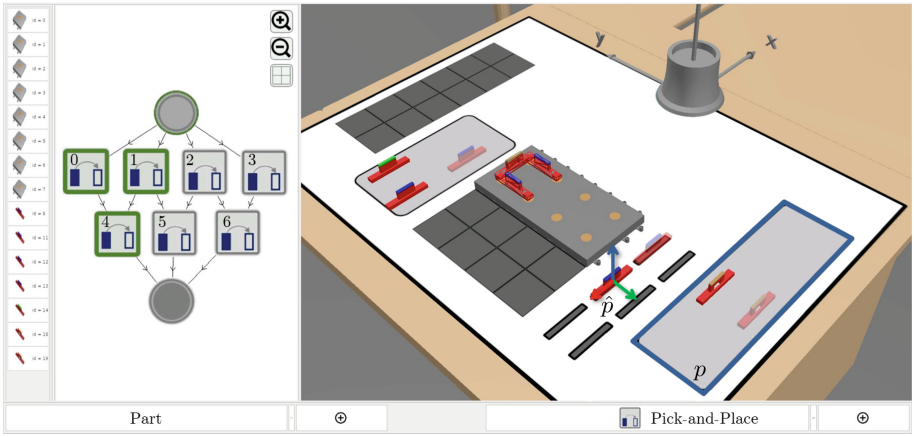


Fig. 5. Our task editor enables icon-based precedence graph modeling (left) with scene creation in a virtual workspace (right). The modeling outputs precedence graphs with inherent ambiguities (e.g., the specific $\hat{p}$ and generic p location).

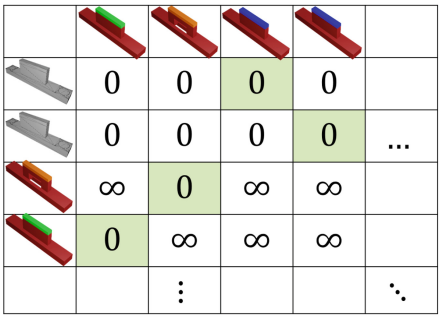
partial order $\prec_S$ defines "earlier-later" relations between skills. Skills represent any operation to change a part feature (e.g., moving a part to a new location). Thus, we define *generic skills* as a tuple $s = (p, \psi)$ of a part template $p \in P$ and a prediction function ψ. The part template p represents the skill's *precondition*, i.e., the required features of a part to be utilized. Given p, the *prediction* ψ states the required features of the part after a successful execution. Thus, ψ maps a precondition p to another part template p^* – i.e., the *postcondition* of the skill. Given the set S of skills, we call the *predicted templates*

$$P^* = \{p^* \mid \forall\, (p, \psi) \in S : p^* = \psi(p)\}. \tag{1}$$

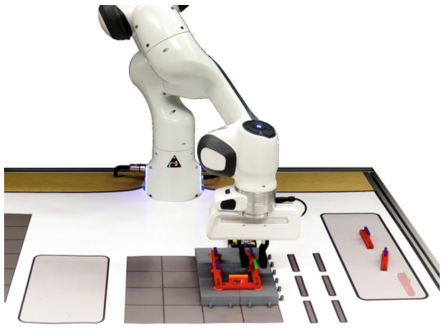
Our skill definition differs from the traditional approach of fully specific skills [21] since skills may involve ambiguous part types and locations. A skill $s \in S$ may only be applied to a part $\hat{p}_j$ iff $\hat{p}_j$ *satisfies* the part template $p_i \in s$, i.e., if it matches the precondition encoded by the input part template:

$$\text{satisfies}(\hat{p}, p) = \text{is\_a}(\hat{p}^{\text{type}}, p^{\text{type}}) \ \wedge \ \text{is\_at}(\hat{p}^{\text{loc}}, p^{\text{loc}}). \tag{2}$$

In practice, precedence graphs, as introduced above, are created by domain experts using the intuitive graphical editor shown in Fig. 5. Parts can be placed within a virtual workspace to be used as input parameters to skills, which can then be connected to precedence graphs. Please refer to our previous publications for details on this visual programming process [26,28].



(a) Matrix entries encode whether a part state (column) satisfies a template (row).



(b) The robot executes a fully specified pick-and-place skill.

Fig. 6. The underspecified part templates are anchored to the part states by solving the assignment problem utilizing a satisfaction matrix (a). Given an admissible assignment (green entries), robot skills can be executed (b). (Color figure online)

4.2 Robot Task Execution

For robots to cope with ambiguity, the underspecified skills are anchored using the approach from our prior work [28]: Initially, the robot detects all parts in the workspace with object recognition techniques and builds up a world state $\hat{P}$. The anchoring process matches each part template $p \in P$ with a part state $\hat{p} \in \hat{P}$ that satisfies p (Eq. 2). To find an admissible assignment, at least one part state must be provided per template ($|\hat{P}| \geq |P|$), yielding $\mathcal{O}(|\hat{P}|!)$ possible assignments. Even for few part states ($|\hat{P}| \leq 10$), testing each assignment is infeasible. However, we can solve this assignment problem with efficient algorithms, e.g., the Kuhn-Munkres algorithm [17] with $\mathcal{O}(|\hat{P}|^3)$ runtime complexity:

Let $\mathbf{A} = (a_{i,j})$ denote a $|P| \times |\hat{P}|$ cost matrix with a row for each part template and a column for each part state (Fig. 6a). Any correct assignment of $\hat{p}_j$ to p_i has no cost, whereas false assignments have infinite costs, i.e.,

$$a_{i,j} = \begin{cases} 0 & \text{if satisfies}(\hat{p}_j, p_i) \\ \infty & \text{otherwise} \end{cases}, i \in \{1, ..., |P|\}, j \in \{1, ..., |\hat{P}|\}. \tag{3}$$

The Kuhn-Munkres algorithm outputs an injective mapping $f : \{1, ..., |P|\} \rightarrow \{1, ..., |\hat{P}|\}$ which minimizes the cost term $\sum_i a_{i,f(i)}$ ($i \in \{1, ..., |P|\}$). By construction of $\mathbf{A}$, an admissible assignment f has 0 cost, and any solution involving a wrong assignment has an infinite overall cost. This occurs if necessary parts are missing in the current world state. In other words, f says that part template p_i must be associated with part state $\hat{p}_{f(i)}$ to incur an overall correct assignment.

Given this assignment, the precedence graph can be scheduled into a skill sequence. In addition, generic skill parameters must be specified. For example, a grid-based placement planner determines specific transformations from generic goal locations. A fully specified skill can then be executed by a typical state-of-the-art skill architecture (e.g., [2,21]) as shown in Fig. 6b.

4.3 Task Optimization

This step analyzes labeled image data by applying techniques to extract hidden process information (*Image Analysis*). This gives us RPDs attached to the considered task description and, thus, to the process model. For human tasks, this is done by creating a *Task Annotation* resulting in HT*, whereas, for RTs, the underlying template is adjusted (*Template Refinement*) resulting in RT*.

Image Analysis. Regarding this step, we follow our previous work [11]. The Image Analysis is described by a function ϕ mapping labeled images V to RPDs D, i.e., $\phi(V) = D$. Thereby, $V = \{(v_1, l_1), (v_2, l_2), ...\}$ is a set of images v_i recorded after execution of a task T and labeled as $l_i \in \{0, 1\}$. The label indicates whether the process was successful ($l_i = 1$) or failed ($l_i = 0$). The Image Analysis step has three phases: (i) A *convolutional neural network* (CNN) is trained with the labeled image data to predict whether an image shows a successful or failed execution. (ii) We employ *Local Interpretable Model-Agnostic Explanations* (LIME) [24] to generate an explanation for each positively labeled input image $\{(i, l) \in V \mid l = 1\}$. LIME highlights image regions relevant for predicting the positive class, i.e., features decisive for process success. (iii) We derive D from these local explanations in a generalization step. Thereby, we focus on image content semantically representing the same object or part across all images to derive global insights. Initially, we search for parts within the highlighted regions of each local explanation. We then analyze each identified part for further information based on a selected set of features. Thereby, the set of features is domain-specific and exchangeable. For instance, the color, shape, or position can be determined of each part identified in a local explanation. Across all local explanations, we receive a set of features and values for each part, representing relevant process details. Formally, the output of this step is a set of RPDs D. Each $d \in D$ refers to exactly one part for which relevant details were found. Thus, d comprises a set of analyzed features and values for that object. Given the features of our part templates, we define a RPD $d = (d^{\text{type}}, d^{\text{loc}})$ as a tuple of a generic type $d^{\text{type}} \in P^{\text{type}}$ and a generic location $d^{\text{loc}} \in P^{\text{loc}}$. Furthermore, D always details the task instruction of T. How T is to be adapted varies depending on the agent, as described in the subsequent sections.

Task Annotation for Human Workers. The discovered RPDs have to be integrated into the process model while preserving model readability for human workers. Therefore, intuitive *task annotations* (e.g., texts, diagrams, images) are created [10]. The task annotations are then attached to the original process model as proposed by [35]. This gives us an improved task HT*, which enriches the input process model PM by RPDs.

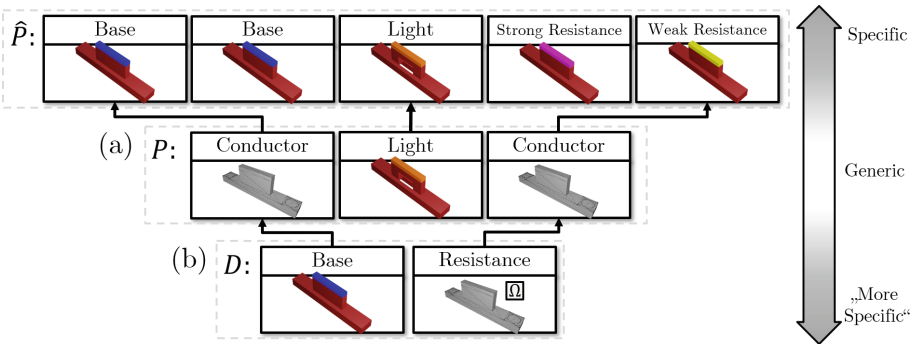


Fig. 7. For a successful execution of tasks with varieties, every part template $p \in P$ must be correctly mapped to one part state $\hat{p} \in \hat{P}$ (a). If the tasks with varieties are modeled too coarsely, the part templates must be specified by the RPDs by mapping every detail $d \in D$ to one template $p \in P$.

Template Refinement for Robot Tasks. In contrast to task annotations for humans, robots require explicit adjustments to the task description. Therefore, the skills and part templates in the task must be identified and adjusted corresponding to the RPDs. This problem is analogous to the mapping from part templates to part states (Fig. 7): RPDs D and templates P constitute generic features of parts and, therefore, are modeled equally (cf. Image Analysis). However, RPDs D refer to the world state *after* execution, whereas part templates P specify requirements of the initial world state *before* execution. We can bridge this gap due to our skill modeling, which allows us to predict expected part templates P^* after execution (Eq. 1). These predicted templates may be too generic and must, therefore, be restricted by the RPDs. To this end, a mapping from each detail $d \in D$ to exactly one template $p \in P^*$ is required. The RPDs can only arise from the execution of overly coarse skills – in consequence, (i) there are fewer RPDs than templates, and (ii) the RPDs must be equally or more specific an the part templates. For that, we define the predicate specifies: $D \times P \rightarrow \{\textsc{True}, \textsc{False}\}$ that returns whether a RPD d_i is at least as specific as the template p_j. This means that d_i^{type} is equal to, or a child of p_j^{type}, and that d_i^{loc} is equal to or lies in p_j^{loc}. Due to the general definitions of our part types and locations, the specifies function is defined analogously to satisfies:

$$\text{specifies}(d, p) = \text{is\_a}(d^{\text{type}}, p^{\text{type}}) \ \wedge \ \text{is\_at}(d^{\text{loc}}, p^{\text{loc}}). \tag{4}$$

Again, a brute force mapping between RPDs D and templates P is infeasible due to the number of feasible combinations but can be achieved with the Kuhn-Munkres algorithm (Sect. 4.2): Let $\mathbf{B} = (b_{i,j})$ denote a $|D| \times |P|$ cost matrix with a row for each RPD and a column for each part template. We model wrong assignments with the *specifies*-function:

$$b_{i,j} = \begin{cases} 0 & \text{if specifies}(d_i, p_j) \\ \infty & \text{otherwise} \end{cases}, i \in \{1, ..., |D|\}, j \in \{1, ..., |P|\}. \tag{5}$$

Solving the assignment problem gives us the minimum cost assignment $g : \{1, ..., |D|\} \rightarrow \{1, ..., |P|\}$ stating that RPD d_i must restrict part template $p_{g(i)}$.

Having identified the part templates belonging to the RPDs, they must be constrained. Transferring the values of a RPD to the assigned templates is not sufficient because the templates are already transformed by the prediction of the task. Thus, it must be differentiated whether the initial templates or the prediction of the skills must be adjusted. To this end, the prediction of a skill must be considered: If the skill adjusts the feature to be constrained during execution, the prediction must be adjusted (e.g., the target location in Pick-and-Place-Skills). If the skill does not affect the feature, the initial template (precondition) must be adjusted. For example, this is the case if part types need to be restricted and only Pick-and-Place-Skills are used.

5 Evaluation

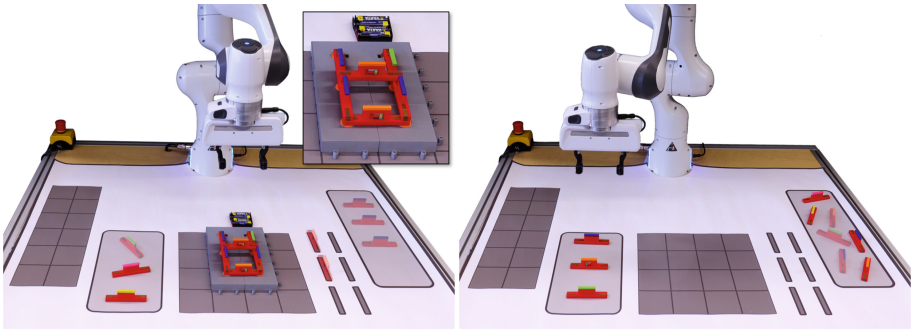
We have evaluated our approach and prototypical implementation in two experiments motivated by possible use cases from the manufacturing sector (Sect. 5.1). The results and optimization steps to improve the knowledge extraction are elaborated in Sect. 5.2. In Sect. 5.3, we finally discuss the results from different perspectives and provide recommendations for similar setups.

5.1 Experimental Validation

We designed two use cases built upon benchmark tasks of [27] and previous real-world process use cases of [11]. Both use cases address process steps that involve a robot to place a set of work pieces (conductors) in a given working environment (Fig. 8). Use Case 1 (UC1) describes an assembly and Use Case 2 (UC2) a kitting task. For our experiments, we assumed that the use cases are designed as task models with the purpose of being executed by a robot agent and that they are part of a given process model (Sect. 4). Furthermore, we made the following assumptions:

1. Labeled images are given. The data stems from executions of the respective task model, i.e., UC1 or UC2. After execution, an image of the workspace is captured and labeled according to the success of process outcomes.
2. The task models for both use cases are too generic and lead to deviating process outcomes.
3. We precisely know the missing process detail causing process failure. This allows us to validate that our approach succeeds in finding the missing RPD.

In order to have sufficient and qualitatively adequate data we used synthetically generated images corresponding to the scenes in Fig. 8. We generated and labeled images for each use case based on known, predefined rules that distinguish a successful execution from an unsuccessful one. The process domain remains consistent across both experiments and all task executions, resulting in an unchanged background image.

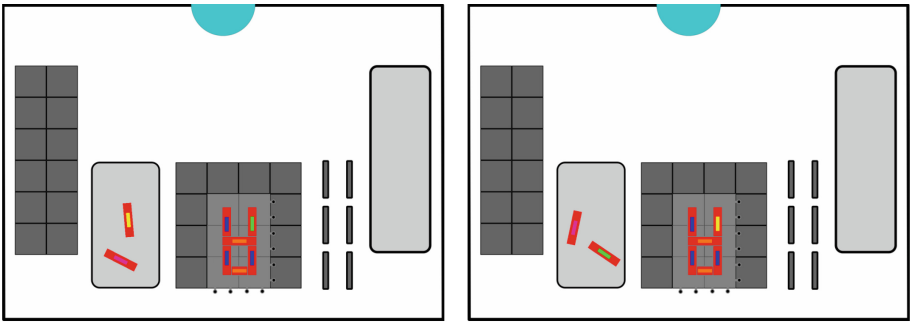


(a) Use Case 1: Assembly (b) Use Case 2: Kitting

Fig. 8. We evaluate our approach with use cases from real-world process environments. Figures (a) and (b) show the world states before (semi-transparent) and after task execution for each use case respectively.

Use Cases. The first use case (UC1) is an assembly scenario. The task is to mount conductors on a circuit board in a specific arrangement. Six conductors must be placed, each being identified by its color: base conductors are blue, resistance conductors are green, yellow, or pink, and light conductors are orange. Different colors of resistance conductors indicate different resistance values (weak = yellow, medium = green, strong = pink). The board is positioned in the scene at a fixed location. The conductors are initially located in two areas (Fig. 8a). Precise positions within the grey regions can vary. As commonly observed in practice (e.g., [14,18]), we assume that the task model was created by a non-expert who is familiar with the process flow but who does not know further details about the process. Consequently, the task model contains information that the circuit board needs to be populated by three base conductors, two light conductors, and one resistance conductor at specific positions. However, it is not specified which resistance conductor (which strength) should be placed. If a low-value resistance (yellow) is attached, the light conductor will burn out, thus rendering the process unsuccessful. On the other hand, if the resistance is too strong (pink), the light conductor glows too weakly. Therefore, placing a medium resistance (green) at the given position on the circuit board is crucial for process success. Since the task model has no further specifications, the robot picks a resistance conductor of any type that is available in the workspace during task execution. Deviating process outcomes are observed from which labels are derived for the captured images. We validate if the RPD, i.e., the right type of resistance, can be extracted from a labeled image data set. This means that the association of the feature "color" with process success has to be identified.

The second use case (UC2) covers a kitting scenario, i.e., the delivery of all components required for the assembly of a product. In the initial situation of UC2, all available conductors are located within the right region of the workspace (Fig. 8b). The final state requires three conductors (a base, light, and resistance each) to be positioned in the left region. In this scenario, we assume that a



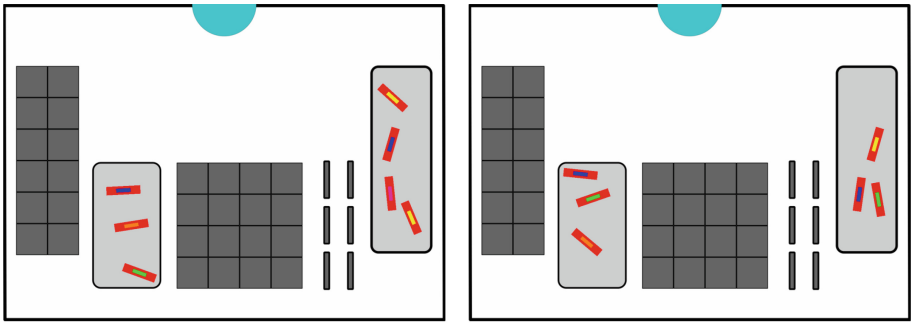
(a) Positive Example (b) Negative Example

Fig. 9. Example input data for Use Case 1.

process expert created a generic task model to move one conductor of each type from the right to arbitrary, varying positions in the left region. This generic task model was sufficient as long as the subsequent process of assembling the parts was performed by human workers who were able to deal with varying part-feeding locations. Over time, the assembly line was restructured, and the subsequent assembly process was assigned to a robot that expects parts in a specific order for grasping: The base conductor must be placed in the top, the light conductor in the middle, and the resistance conductor in the bottom third of the left region. The initial kitting task model does not contain these new RPDs regarding more specific part goal locations. Executing this model yields the observation that the process occasionally fails. With UC2, we show that the location-related RPDs can be extracted with our approach. This involves a more complex association between two features (color and position) and process outcomes.

Implementation. We trained a classification model using TensorFlow[1] for both use cases. Regarding the CNN, we followed a standard model architecture which comprised three convolution layers followed by corresponding pooling layers. Subsequently, the output was flattened and fed into a fully-connected dense layer to derive the final classification outcome. As training data, we generated image data per use case representing the respective setups. For UC1, we generated 1000 images, of which 500 showed positive (Fig. 9a) and 500 negative states (Fig. 9b) of the workspace after task execution from a top-down view. The positions of the six conductors on the circuit board were fixed across all images since they were defined precisely in the task model. The position and orientation of the two remaining conductors were determined at random – this reflects the variability of the process concerning initial part locations. For UC2, we generated 10000 images comprising 5000 positive (cf. Fig. 10a) and 5000 negative samples (cf. Fig. 10b). The position and orientation of each conductor was again determined randomly within the respective rectangular regions. The training of the CNN

[1] https://www.tensorflow.org/ (Accessed: 02 May 2023).



(a) Positive Example (b) Negative Example

Fig. 10. Example input data for Use Case 2.

per use case was performed until a sufficient high accuracy rate (> 0.95) and low loss (< 0.1) were achieved.

The first experiments were conducted with the basic version of LIME as provided in the official python package[2]. We defined important input parameters according to the process domain and experimental setup: the parameter n_{features} describes the maximum number of image segments to be considered in the explanation. This parameter is crucial for useful and interpretable results since consideration of too many features would lead to uncertainty in which segments are of real importance for the classification. In contrast, not all important segments are highlighted by selecting a too-low value. We know that, in our domain, the reason for process success or failure is always related to the parts since the image background is fixed across task executions. The number of parts is, hence, a good guide in choosing n_{features}. However, since we do not know which object or object feature is relevant, all possible occurring objects must be considered. In UC1, a maximum of 8 and UC2, a maximum of 7 conductors may occur in the scene. Therefore, we set n_{features} accordingly. The number of sample instances used by LIME to generate a local explanation is defined by n_{samples}. Higher values produce more samples and more accurate explanation at the cost of computation time. Inspired by existing code examples and experience from prior experiments, the value for both use cases was set to 1000.

We implemented the generalization step using an object recognition procedure based on color. For each local explanation, we extracted the color information from all highlighted segments and identified the parts in individual segments this way. We then identified all regions of the segment that correspond to an object, i.e., regions that differ in color from the background. We calculated the centroid of each object region, which is used to determine the object's position. The object information identified in this way is collected across all local explanations, and a list of positions is output at the end for each object type.

[2] https://github.com/marcotcr/lime (Accessed: 30 April 2023).

	Use Case 1	**Use Case 2**
Type	*Position*	*Position*
Base	(390, 382)	[(172, 290),...,(258, 341)]
Weak Resistance	[(179, 316),...,(247, 471)]	-
Medium Resistance	(434, 382)	[(176, 429),...,(258, 480)]
Strong Resistance	[(179, 302),...,(258, 458)]	-
Light	(410, 463)	[(171, 359),...,(259, 409)]

Table 1. Resulting details in the default setup for both use cases.

5.2 Results and Optimizations

The output of the generalization step, i.e., the derived RPDs, for each use case is summarized in Table 1. For UC2, the default setup provides correct results since the analyzed RPDs align with the anticipated specifications: for each of the three conductors (base, light, and medium resistance), a region for the parts to be placed was extracted. The regions are represented by a bounding box for each RPD, referring to the top, middle, and bottom third of the left area, as expected in this use case. In UC1, five RPDs were detected. They indicate that five conductors (base, light, weak resistance, medium resistance, strong resistance) must be present at the computed positions during the execution of the task for process success – this does not match our expectations since only a single conductor (a medium resistance) at a specific position is relevant for success in this case. To get to the root cause of this result, we examined the partial results of each step of our analysis process. Examples of the results of the explanation step, i.e., the local explanations, for each use case are shown in Fig. 11. The images provide evidence that the error occurred during this step. In UC1 (Fig. 11; left), eight image segments (as determined by n_{features}) are highlighted, meaning that they were essential for the image to be classified as a successful process execution. Across all explanations for UC1, the segment containing the base (blue rectangle) and medium resistance (green rectangle) conductor is always highlighted. While this

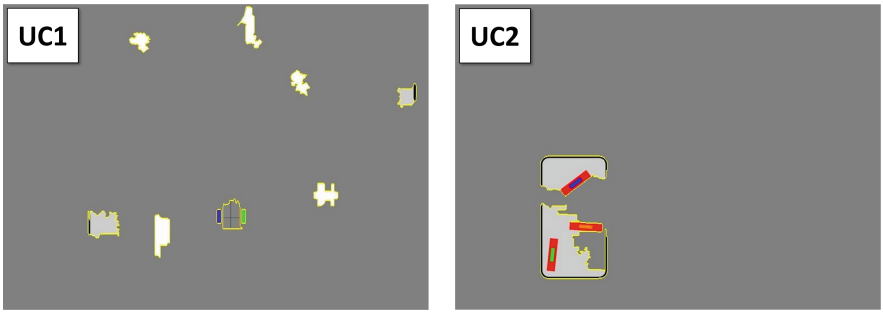


Fig. 11. Exemplary local explanations in the default setup.

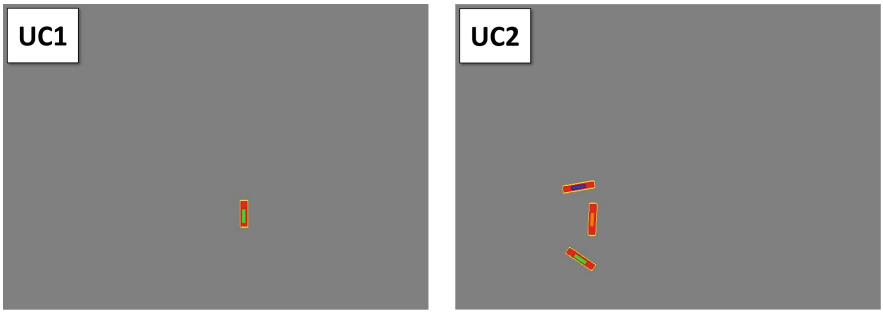


Fig. 12. Exemplary local explanations in the optimized setup.

partially aligns with the expected outcome, it is inaccurate due to the highlighting of the base conductor. This is the result of the segmentation sub-step of LIME, where both conductors are assigned to the same segment. Consequently, either both conductors or none of them can be marked relevant in the explanation. Furthermore, in a few cases (18%), some highlighted segments contain small parts of the other resistance conductors placed in the left area of the scene. Since the generalization step considers all highlighted segments in all explanations, these parts are also analyzed, and identified conductors are erroneously declared as RPDs (Table 1). The inaccurate outcomes can thus be attributed to a sub-optimal execution of intermediate steps. Therefore, we applied optimizations to enhance the knowledge extraction:

1. The segmentation process of LIME was adapted to ensure that each object is assigned to a single segment. Additionally, we replaced the quick shift algorithm, which is by default used in the LIME implementation, with the SLIC algorithm. SLIC is better suited for the shape of parts in our domain.[3]
2. Based on a segmentation that assigns an individual segment to each conductor, the optimal value for the LIME parameter $n_{\text{features}} = 1$ in UC1 (3 in UC2). This value can be obtained by determining a cut-off based on the weight-ordered sorting of segments for classification purposes. If the distance between the values exceeds a certain threshold, the segments should not be merged while we define the cut-off at this point.

These optimizations enable the adaptation of the LIME implementation to our domain. They result in a tendency to highlight fewer patches and increase the robustness of the generalization step, in turn reducing requirements for this step. The local explanations resulting from the optimized setup are shown in Fig. 12. For UC2, the results of the generalization remain unchanged in the optimized setup compared to the default version. However, the local explanations are more precise, thereby rendering them more reliable. For UC1, we are able to achieve

[3] https://scikit-image.org/docs/dev/api/skimage.segmentation.html (Accessed: 08 May 2023).

the expected RPD, resulting in the generalization step exclusively providing the medium resistance conductor with the specific positional information (434, 382).

5.3 Discussion

The experimental results demonstrate the efficacy of our novel approach for concretizing underspecified task models. The processing pipeline in which steps are executed sequentially leads to potential issues with error propagation. Regarding this aspect, we have determined that the results of image classification and explanation generation with LIME are the most crucial. We, therefore, want to discuss these two steps regarding their scalability to other process environments:

(i) In the initial step of training a CNN predictor for process outcomes, we particularly encountered the challenge of acquiring a sufficient amount of training data. In our experiments, we found that a data set with at least 10000 images was indispensable for favorable outcomes in UC2 – insufficient sample size or imbalanced distribution of positive and negative samples led to inadequate outcomes in the explanation step. Small and medium-sized enterprises may have difficulties with obtaining this amount of data since manufacturing processes may here not occur sufficiently frequently. Therefore, alternative strategies for the training process in real-world settings are needed. Data augmentation approaches that generate additional samples from a smaller data set might mitigate this issue. Additionally, pre-trained CNNs from similar domains in which large-scale data sets are available could be a viable alternative.

(ii) Our results regarding the local explanation generation step have shown that LIME requires modifications to ensure efficacy for our objectives. Despite a well-trained CNN, an appropriate value of the parameter n_{features} had to be tuned meticulously, and the segmentation approach had to be adapted for more precise outcomes. However, we do not consider this a major issue. On the one hand, the results of UC2 demonstrate that satisfactory outcomes can be obtained in principle with the default settings of LIME in some scenarios. On the other hand, domain knowledge, such as the number and features of parts, is usually available for individual process environments. Our experiments show that knowing the number of parts in the process provides guidance for an initial estimate and consecutive improvement of n_{features}. Furthermore, due to other methods employed during the design, engineering, and production stages of manufacturing environments (e.g., quality control processes), appropriate segmentation techniques adapted to the process might be readily available. Based on the low level of effort required and the significant results optimization, we highly recommend adapting the LIME segmentation step to increase the probability of successfully identifying RPDs.

6 Conclusion and Future Work

We presented a novel approach to improve human and robot process models by deriving specifications from relevant process details. Our approach unlocks new potentials regarding human-robot collaboration through enriching process models for both agents. It extends previous work in the context of BPM and proves its applicability for further domains. We evaluated our work with two use cases inspired by the manufacturing industry. The experimental results provide strong evidence of its effectiveness in identifying relevant process details.

Future research should focus on the improvement of the robustness of the approach. This includes examining data augmentation techniques to enable more straightforward applicability in real process environments that come with limitations in execution data quantity and quality. Furthermore, we aim to conduct experiments with more complex parts and settings to explore opportunities for enhancement and provide recommendations for implementing the approach in sophisticated process settings.

Acknowledgements. We thank Philipp Jahn and Carsten Scholle for their valuable work supporting the implementation and evaluation of our approach.

References

1. Ahmadikatouli, A., Aboutalebi, M.: New evolutionary approach to business process model optimization. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 2 (2011)
2. Andersen, R.H., Solund, T., Hallam, J.: Definition and initial case-based evaluation of hardware-independent robot skills for industrial robotic co-workers. In: 41st International Symposium on Robotics (ISR), pp. 1–7. VDE (2014)
3. Becker, J., Rosemann, M., Von Uthmann, C.: Guidelines of business process modeling. In: BPM: Models, Techniques, and Empirical Studies, pp. 30–49 (2002)
4. Beumelburg, K.: Fähigkeitsorientierte Montageablaufplanung in der direkten Mensch-Roboter-Kooperation (2005)
5. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_7
6. Chinosi, M., Trombetta, A.: BPMN: an introduction to the standard. Comput. Stand. Interfaces **34**(1), 124–134 (2012)
7. Coradeschi, S., Saffiotti, A.: An introduction to the anchoring problem. Robot. Auton. Syst. **43**(2–3), 85–96 (2003)
8. Darvish, K., et al.: A hierarchical architecture for human-robot cooperation processes. IEEE Trans. Rob. **37**(2), 567–586 (2021)
9. Dietz, T., et al.: Programming system for efficient use of industrial robots for deburring in SME environments. In: 7th German Conference on Robotics, pp. 1–6 (2012)
10. Fichtner, M., Fichtner, U.A., Jablonski, S.: An experimental study of intuitive representations of process task annotations. In: Sellami, M., Ceravolo, P., Reijers, H.A., Gaaloul, W., Panetto, H. (eds.) CoopIS 2022. LNCS, vol. 13591, pp. 311–321. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17834-4_19

11. Fichtner., M., Schönig., S., Jablonski., S.: How lime explanation models can be used to extend business process models by relevant process details. In: Proceedings of the 24th International Conference on Enterprise Information Systems. vol. 2, pp. 527–534 (2022)
12. Gounaris, A.: Towards automated performance optimization of BPMN business processes. In: New Trends in Database and Information Systems (ADBIS), pp. 19–28 (2016)
13. Hasegawa, T., Suehiro, T., Takase, K.: A model-based manipulation system with skill-based execution. IEEE Trans. Robot. Autom. **8**(5), 535–544 (1992)
14. Koschmider, A., et al.: Business process modeling support by depictive and descriptive diagrams. In: Enterprise Modelling and Information Systems Architectures, pp. 31–44 (2015)
15. Matheson, E., Minto, R., Zampieri, E.G., Faccio, M., Rosati, G.: Human-robot collaboration in manufacturing applications: a review. Robotics **8**(4), 100 (2019)
16. Mendling, J., Reijers, H.A., van der Aalst, W.M.: Seven process modeling guidelines (7pmg). Inf. Softw. Technol. **52**(2), 127–136 (2010)
17. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957)
18. Niedermann, F., Radeschütz, S., Mitschang, B.: Deep business optimization: a platform for automated process optimization. INFORMATIK - Business Process and Service Science - Proceedings of ISSS and BPSC (2010)
19. Nottensteiner, K., et al.: A complete automated chain for flexible assembly using recognition, planning and sensor-based execution. In: Proceedings of 47st International Symposium on Robotics (ISR), pp. 1–8 (2016)
20. Paxton, C., et al.: Costar: instructing collaborative robots with behavior trees and vision. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 564–571 (2017)
21. Pedersen, M.R., et al.: Robot skills for manufacturing: from concept to industrial deployment. Robot. Comput. Integr. Manuf. **37**, 282–291 (2016)
22. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: a slider approach. In: 12th International IEEE Enterprise Distributed Object Computing Conference (2008)
23. Reichert, M., et al.: Enabling personalized visualization of large business processes through parameterizable views. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1653–1660 (2012)
24. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?" explaining the predictions of any classifier. In: Proceedings of 22nd ACM SIGKDD, pp. 1135–1144 (2016)
25. Riedelbauch, D.: Dynamic Task Sharing for Flexible Human-Robot Teaming under Partial Workspace Observability. Ph.D. thesis, University of Bayreuth (2020)
26. Riedelbauch, D., Henrich, D.: Fast graphical task modelling for flexible human-robot teaming. In: 50th International Symposium on Robotics (ISR), pp. 1–6. VDE (2018)
27. Riedelbauch, D., Hümmer, J.: A benchmark toolkit for collaborative human-robot interaction. In: 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), pp. 806–813. IEEE (2022)
28. Riedelbauch, D., Sucker., S.: Visual programming of robot tasks with product and process variety. In: Annals of Scientific Society for Assembly, Handling and Industrial Robotics (to appear) (2022)

29. Scheer, A.W., Thomas, O., Adam, O.: Process modeling using event-driven process chains. In: Process-Aware Information Systems: Bridging People and Software through Process Technology, pp. 119–145 (2005)
30. Selic, B., et al.: Omg unified modeling language (version 2.5). Technical Report (2015)
31. Senft, E., et al.: Situated live programming for human-robot collaboration. In: ACM Symposium on User Interface Software and Technology, pp. 613–625 (2021)
32. Steinmetz, F., Wollschläger, A., Weitschat, R.: Razer - a HRI for visual task-level programming and intuitive skill parameterization. IEEE Robot. Autom. Lett. **3**(3), 1362–1369 (2018)
33. Thomas, U., et al.: A new skill based robot programming language using UML/P statecharts. In: IEEE International Conference on Robotics and Automation (2013)
34. Van Der Aalst, W.M., Ter Hofstede, A.H., Weske, M.: Business process management: a survey. Bus. Process Manage. **2678**(1019), 1–12 (2003)
35. Wiedmann, P.C.K.: Agiles Geschäftsprozessmanagement auf Basis gebrauchssprachlicher Modellierung. Universitaet Bayreuth (Germany) (2017)