Improving Complex Adaptations in Process-Oriented Case-Based Reasoning by Applying Rule-Based Adaptation*

Lukas Malburg[1,2], Maxim Hotz[1,2], and Ralph Bergmann[1,2]

[1] Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany, https://www.wi2.uni-trier.de

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

{malburgl,s4mahotz,bergmann}@uni-trier.de

{lukas.malburg,maxim.hotz,ralph.bergmann}@dfki.de

Abstract Adaptation is a complex and error-prone task in Case-Based Reasoning (CBR), including the adaptation knowledge acquisition and modeling efforts required for performing adaptations. This is also evident for the subfield of Process-Oriented Case-Based Reasoning (POCBR) in which cases represent procedural experiential knowledge, making creation and maintaining adaptation knowledge even for domain experts exceedingly challenging. Current adaptation methods in POCBR address the adaptation knowledge bottleneck by learning adaptation knowledge based on cases in the case base. However, these approaches are based on proprietary representation formats, resulting in low usability and maintainability. Therefore, we present an approach of using adaptation rules and rule engines for complex adaptations in POCBR in this paper. The results of an experimental evaluation indicate that the rule-based adaptation approach leads to significantly better results during runtime than an already available POCBR adaptation method.

Keywords: Process-Oriented Case-Based Reasoning · Adaptive Workflow Management · Rule-Based Adaptation · Drools Rule Engine · Adaptation Operators

1 Introduction

Although Case-Based Reasoning (CBR) [1] systems have been well explored in past research [4], the acquisition of adaptation knowledge still imposes a primary challenge in CBR [12, 16, 34, 35], also known as *Adaptation Knowledge Bottleneck* [17]. For this reason, performing adaptations often requires in-depth domain knowledge by experts [12, 18, 24, 25]. This is particularly evident for synthetic tasks in Process-Oriented Case-Based Reasoning (POCBR) [5, 26], wherein

* The final authenticated publication is available online at https://doi.org/10.1007/978-3-031-63646-2_4

cases represent procedural experiential knowledge in the form of semantic workflow graphs. Consequently, adaptations performed during the *Reuse* phase in POCBR require profound and detailed knowledge about possible and valid structural graph modifications. For example, adaptation knowledge in POCBR is represented by complex graph fragments that are inserted in or deleted from a retrieved workflow. Creating such comprehensive adaptation knowledge is potentially laborious and error-prone for domain experts [24, 25]. In previous work, adaptation methods in POCBR [24,25,28–30] have been developed that learn this needed adaptation knowledge automatically. However, most adaptation methods in CBR and POCBR rely on inherent and proprietary formats, without making the adaptation knowledge intuitively accessible to domain experts. In addition, CBR and POCBR frameworks often do not provide an easy mean for engineering adaptation knowledge, resulting in difficulties for verification and maintaining the learned knowledge or for manually creating new adaptation knowledge [32].

In this paper, we present an approach based on a previously developed adaptation method, in which the state-of-the-art rule engine *Drools*[1] [3] is used for performing workflow graph adaptations in POCBR. For this purpose, we utilize the learning step of the operator-based adaptation [30] and encode the adaptation knowledge directly as corresponding rules. Consequently, instead of encoding adaptations as workflow graph fragments in POCBR, they are encoded as adaptation rules, hiding the complexity of adaptations in suitable predicates in the rules. For adaptation, the rules in the rule base are applied to the retrieved workflow case to increase the similarity to the given query. In an experimental evaluation, we assess the suitability of the rule-based adaptation compared to the operator-based adaptation [30]. By the proposed approach, the adaptation knowledge is more intuitively encoded for domain experts and, thus, enables verification and maintenance of it more easily. Moreover, the approach is domain-independent and also usable for other case representations [4, 7] in CBR or for other process-oriented domains that go beyond the cooking domain used in the evaluation. Finally, by using a state-of-the-art rule engine, the rule-based adaptation approach benefits from new functionalities implemented in the future. For example, advanced search and optimization techniques (e. g., OptaPlanner[2]) can be utilized that are especially tailored for the representation of adaptation rules.

The paper is structured as follows: Section 2 describes the basics covering adaptation in CBR and POCBR and discusses related approaches using rule engines in CBR. In Sect. 3, the concept of encoding and applying adaptation rules is introduced. Section 4 describes the experimental evaluation of the proposed approach. Finally, Sect. 5 concludes the paper and discusses future work.

2 Foundations and Related Work

The foundations for this work consist of the *NEST* graph representation and similarity computation introduced by Bergmann and Gil [5] which are necessary

[1] https://www.drools.org
[2] https://www.optaplanner.org/

to represent and assess cases in POCBR. Further, knowledge representations and adaptation approaches in transformational adaptation, i. e., adaptation rules and operators, constitute a basis for this work. Hence, we introduce the relevant concepts in the following.

2.1 Semantic Workflow Representation and Similarity Assessment

In POCBR, cases represent procedural experiential knowledge [5, 26]. In this work, we use semantically annotated graphs, named *NEST* graphs [5], to represent cases. A *NEST* graph is a directed graph and represented by the quadruple $W = (N, E, S, T)$: N is a set of nodes and $E \subseteq N \times N$ a set of edges. $T : N \cup E \rightarrow \Omega$ assigns a concrete type to each node and each edge. Furthermore, $S : N \cup E \rightarrow \Sigma$ specifies a semantic description from a semantic metadata language Σ to each node and edge. These semantic descriptions can be used to describe a node or edge in more detail by semantic knowledge. A case base consists of several such *NEST* graphs that can be utilized in a POCBR system. In Fig. 1, an exemplary *NEST* graph is shown that represents a simple cooking recipe with different node and edge types and exemplary semantic descriptions. In this context, task nodes describe the cooking steps performed,

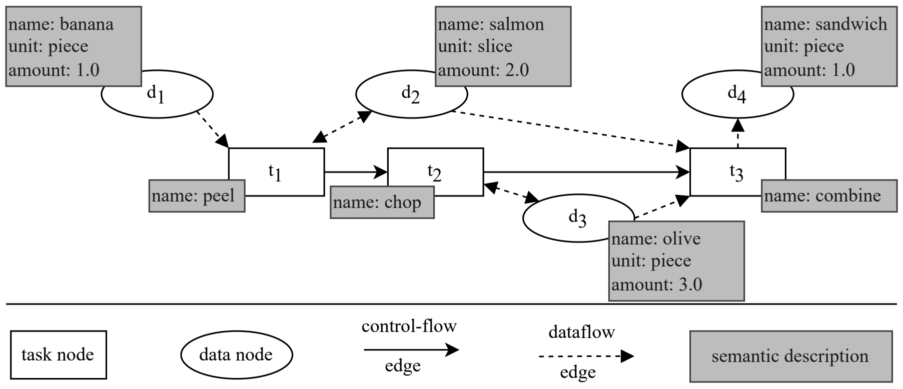

Fig. 1. *NEST* Graph with Semantic Descriptions.

and data nodes the corresponding ingredients used during preparation of the dish. The semantic descriptions used specify the performed task in more detail, i. e., which task should be executed. For data nodes, the semantic descriptions can be used to specify the type of ingredient, the amount, and unit of it. Task nodes are connected by control-flow nodes that define the execution order of the cooking procedure. Dataflow edges are used to define which task node consumes and produces which concrete data node, i. e., which task node needs a certain ingredient for performing the cooking step and what is the result afterwards.

To calculate the similarity between two *NEST* graphs, suitable similarity measures are required. Bergmann and Gil [5] have developed a semantic similarity measure that computes this similarity based on the local-global principle [4]. During this procedure, the global similarity is composed of the local similarities computed between the individual graph elements, based on corresponding local semantic similarity measures [5]. If the types of the graph elements to be compared are not identical, i. e., a data node compared with a task node, the local similarity is always 0.0. Otherwise, the similarity is calculated based on the similarity of the semantic descriptions attached to the graph elements. A graph mapping is initiated in which the best possible mapping, i. e., the mapping that results in the highest global similarity, between the query workflow elements and the case workflow elements is determined.

2.2 Adaptation Methods

Based on the semantic similarity assessment between a given query workflow and a workflow graph from the case base, the most similar case is used as the basis for adaptation. Adaptation is often required for synthetic tasks in Process-Oriented Case-Based Reasoning (POCBR) to better fit the retrieved solution to the needs of the user. In general, adaptation methods can be divided into two main categories: Generative adaptation and transformational adaptation [4]. Generative adaptation is aiming to solve a problem from scratch, even without using a case with experiential knowledge from the case base [4, 34]. Typically, a knowledge-based problem solver is needed that can solve the problem from scratch. CBR can be used to accelerate the adaptation process by reusing already available solutions from the case base. In contrast, transformational adaptation aims to modify a retrieved case from the case base to better fulfill the requirements that a user specifies in a given query. Therefore, adaptation operators or adaptation rules are used that specify the context in which a case can be modified. Depending on the degree of modification, a distinction is made between substitutional adaptation for modifying values at the attribute level and structural adaptation for larger and more complex, structural changes of the retrieved case [4]. In the following, the application of transformational adaptation with adaptation operators and adaptation rules is described in more detail.

Adaptation Operators: Adaptation operators sequentially transform a retrieved case into a successor case if the successor case remains valid, i. e., they describe generally applicable transformations of a retrieved solution. By chaining several adaptation operators together, a retrieved case can be transformed into an adapted case that better fits the user's requirements. The sequence in which operators are applied to a case is determined by search techniques, such as local or global search algorithms. Müller and Bergmann [30] present an approach in previous work in which they learn adaptation operators based on a case base in POCBR. The proposed adaptation operators are inspired by *STRIPS* operators from AI planning [14] and, thus, consist of an add part and a delete part. In the terminology of workflows used in POCBR this means that the add part is handled as an *insertion* into the graph and the delete part as a *deletion*. Both, insertion and deletion part, are comprehensive workflow graph fragments called *streamlets* that are inserted or deleted in a retrieved workflow that should be adapted. Based on these definitions, three kinds of adaptation operators can be created: 1) an *insert operator* only consisting of an insertion part, 2) a *delete operator* only consisting of a deletion part, and 3) an *exchange operator* with both an insertion and a deletion part. During learning of the adaptation operators, workflow pairs from the case base are considered. In this context, one of the workflows is transformed into the other one by deleting and adding parts in the workflow or by exchanging workflow fragments. Since creating, verifying, and maintaining complex workflow graph fragments such as used for adaptation operators are demanding and challenging tasks, domain experts are required to have in-depth knowledge about possible graph adaptations. In addition, state-of-the-art CBR and POCBR frameworks lack the ability to adequately represent such adaptation knowledge for domain experts and knowledge modelers [32].

Figure 2 depicts the *NEST* graph from Fig. 1 with a marked streamlet (dashed line). According to the definition provided by Müller [27], a streamlet consists of a partial workflow that is constructed based on a particular data node. This data node is referred to as the head data node of the streamlet. Starting from this head data node, a streamlet also includes all tasks that are directly connected to the head data node by a dataflow edge. Based on the essential concept of a head data node, the task node that only consumes the head data node is called the anchor task. The anchor task determines the position of modification in the graph. Every other node, e. g., t_2, is a normal part of the streamlet. Looking at the example from Fig. 2, the head data node d_3, the anchor task t_3, and the task node t_2 are part of the streamlet created during learning. Assuming that this learned streamlet represents a delete streamlet means that 1) the head data node is deleted from the workflow and 2) each unproductive task, i. e., task nodes that do not have any incoming or outgoing dataflow edge, are deleted. t_2 is after the deletion of d_3 such an unproductive task that must be deleted. In the example, d_3 and also t_2 are deleted. If there exists an insert streamlet in the learned adaptation knowledge (see left side of Fig. 2), it can subsequently be inserted into the graph. In the example, it is determined whether the anchor

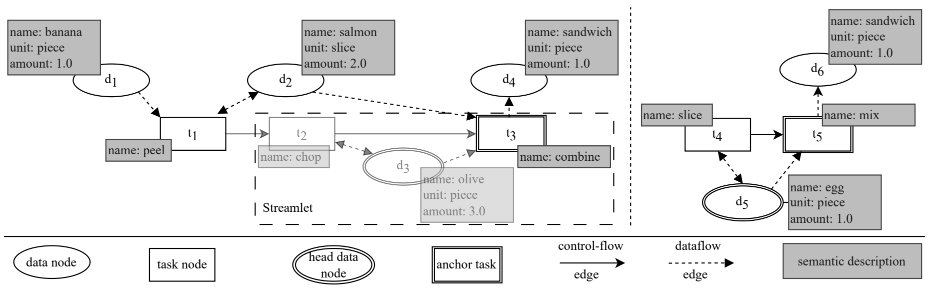

Fig. 2. *NEST* Graph with Learned Streamlet (Left Side) and Substitute Streamlet (Right Side).

tasks t_3 and t_5 from the insert streamlet are sufficiently similar to each other and produce the same output, i. e., d_4 and d_6. Assuming that both conditions are satisfied, the head data node of the streamlet d_5 and the task node t_4 are inserted into the graph. Finally, a cleanup function restores the syntactical correctness of the graph by inserting control-flow edges between t_1 and the newly inserted node t_4 and between t_4 and the anchor task t_3. In addition, a dataflow edge between the new head data node d_5 and the anchor task t_3 is inserted. In this context, it is important to note that every time an adaptation operator with either an insert or delete streamlet is applied, only the plain streamlet, i. e., all workflow components except of the anchor task are inserted or deleted to the corresponding graph. During adaptation, each data node in the retrieved graph is considered as head data node and a corresponding streamlet is constructed. Based on this streamlet, suitable and applicable adaptation operators are retrieved and executed. This procedure is repeated until all data nodes have been used once to construct a streamlet.
Adaptation Rules: In contrast to adaptation operators, adaptation rules [4] transform the solution of a case into another solution by exploiting the differences between the given query and the problem description of the retrieved case. Instead of modifying problem description and solution as for adaptation operators, only the solution part of the case is modified. Adaptation rules have a *precondition part* that must be fulfilled to apply the corresponding transformation of the *conclusion part*. For this purpose, the query case, the retrieved case, and the adapted case should be checked for their compatibility. Each adaptation rule whose precondition is fulfilled is then executed on the case [4, 27].

2.3 Rule Engines in CBR

In this section, we discuss relevant related approaches for using rule-based adaptation and rule engines in CBR. However, most of the existing work utilizes rule engines in the context of classic CBR, targeting attribute-value based cases, and not procedural cases in POCBR.

Bergmann et al. [8] present an approach of using adaptation and completion rules from general knowledge in CBR. For rule application, they propose the usage of a forward chaining rule interpreter based on a Rete-Network. For this, they use the *NéOpus* system [31], which allows the organization of rules by the means of sub-rule bases and dedicated Rete-Networks according to the structure of the case representation [8]. Bach and Althoff [2] introduce an approach in which they integrate the *Drools* rule engine into the open-source CBR framework *myCBR*. By this, it is possible to use completion and adaptation rules in a wide variety of *myCBR* application domains. Hanft et al. [15] describe the integration of *Drools* within the *SeMantic Information Logistics Architecture (SMILA)*[3]. In their work, the authors link the rule-based adaptation of CBR systems with business-oriented workflow systems by introducing the *Rule-based Adaptation of Case-based Knowledge (RACK)* for integrating CBR func-

[3] https://projects.eclipse.org/projects/rt.smila

tionalities to *SMILA* [15]. Beyond this, there have been some general approaches that investigate the application of rule engines for the purpose of context-aware workflow adaptation: Döhring et al. [13] develop several adaptation patterns for dynamically changing *BPMN 2.0*[4] workflows along with a prototypical implementation using *Drools*. Similarly, the *CBRflow* system [33] can be used for adaptive workflow management, combining rule-based adaptation representing general knowledge and cases specifying already experienced adaptations in problem situations. Besides the exclusive application of a rule engine within CBR systems, there are also some works that employ a hybrid approach of linking CBR systems with rule-based expert systems for workflow modeling and decision support. In healthcare, for instance, rule engines are also applied within *Clinical Decision Support Systems* [9] for medical diagnosis, as discussed in the work of Cabrera and Edye [11]. In their work, they develop a *Medical Diagnostic System* by combining a rule-based expert system with classical CBR methods.

Although there are related approaches that are used in CBR, they predominantly make use of substitutional adaptation concepts or employ adaptation rules for attribute-value cases. Moreover, only the approach by Bach and Althoff [2] is directly integrated into a CBR framework, although adaptation rules are universally better understandable than own, proprietary adaptation knowledge formats, and, thus, promote the broader usability of CBR frameworks. The presented approaches that incorporate a rule engine for workflow modeling support (e. g., [13, 33]) oftentimes deal with event-driven application scenarios and are not necessarily utilizing directly the CBR or POCBR methodology. Thus, it is currently not investigated how rule engines can be used in the context of POCBR to perform complex structural and transformational adaptations of semantic workflow graphs in the *Reuse* phase. In addition, the integration of a rule-based adaptation approach into CBR frameworks is only rarely investigated or not available at all in the context of POCBR. For this reason, this work aims to address this gap and, thus, to provide a first step towards the integration of rule-based adaptation into a POCBR framework.

3 Encoding and Applying Adaptation Rules in POCBR

In this section, we present a rule-based approach for using adaptation rules and a rule engine to perform complex adaptations in POCBR. First, in Sect. 3.1, we describe the predicates used to represent the adaptation operators as adaptation rules. After encoding of the adaptation rules, we discuss in Sect. 3.2 the application of the rules during the *Reuse* phase.

3.1 Encoding Adaptation Operators as Adaptation Rules

As already described in Sect. 2.2, the operator-based adaptation by Müller and Bergmann [30] consists of STRIPS-like adaptation operators. Each operator

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

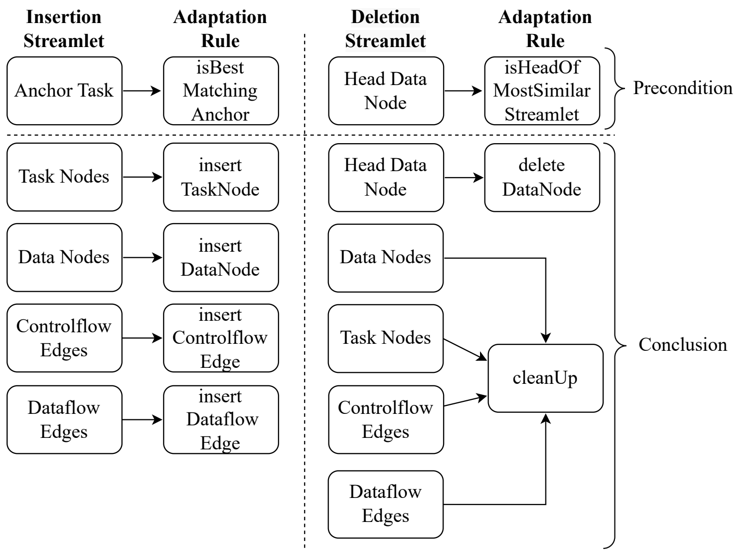

Fig. 3. Mapping Adaptation Operators to Adaptation Rules.

contains either an insertion or a deletion streamlet, or both of them. During adaptation, suitable adaptation operators are selected, performing an insertion, deletion, or an exchange in the retrieved workflow graph.

Based on the learning process in which adaptation operators are created, we inspect either the insertion streamlet or the deletion streamlet or, in the case of an exchange operator, both. To generate suitable adaptation rules, the concepts that are currently encoded in the operator-based adaptation algorithm itself (see [30]) must be converted into suitable predicates. This is required to enable the rule engine to perform validation checks or to find best matching positions in the graph. Figure 3 depicts the mapping of the concepts used for adaptation operators to suitable predicates in the adaptation rules. During the mapping process, the predicates are divided into a set of predicates in the form of preconditions that are inevitably required for the execution of a rule, and corresponding effects that specify the modifications of the adapted case after execution of a rule. To satisfy the property of identifying the best matching anchor task in the graph for the insertion streamlet, we introduce the predicate $isBestMatchingAnchor$. The remaining preconditions of the streamlet insertion, i. e., the condition that the head data node should not exist in the graph yet, can be realized by regular pattern matching and do not require special predicates (see Line 3 in Lst. 1). The insertion of the streamlet elements themselves, i. e., the conclusion of the rule, is represented by straightforward predicates, each assigned to a corresponding item type. For deletion, we specify the predicate $isHeadOfMostSimilarStreamlet$ that is used within the precondition. According to the principles outlined in Sect. 2.2, this predicate allows the identification of a data node residing in the graph that acts as the head data node of the most similar streamlet that is con-

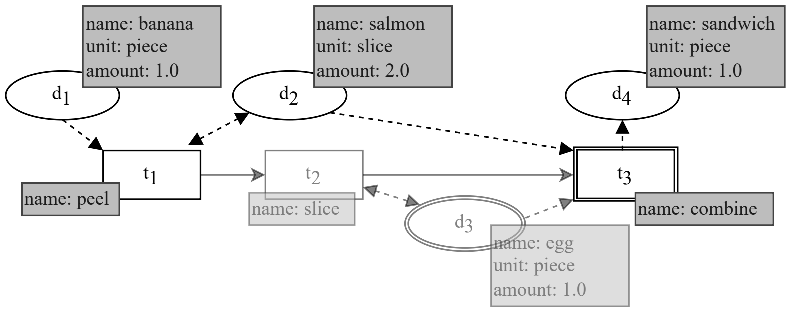

Fig. 4. Adaptation of a *NEST* Graph by Insertion.

structable out of all data nodes present. Since it is not clear which streamlet is to be identified as the most similar one to the deletion streamlet, the conclusion of a deletion rule makes use of a generic predicate *cleanUp* responsible for the removal of left over streamlet elements as proposed by Müller [27].

Figure 4 illustrates the insertion of a streamlet into a *NEST* graph according to the principles outlined previously. We can represent this insertion procedure by the means of a corresponding adaptation rule. Listing 1 depicts this equivalent rule according to the syntax of Drools Rule Language (DRL) after the mapping process. The *when* part describes the preconditions that need to be satisfied for execution, and the *then* part represents the modifications to be performed on the retrieved workflow graph. DRL provides multiple keywords, e. g., *not*, as well as the definition of variables, e. g., *$aC*, which can be matched to objects/-facts during the activation of the precondition. In particular, these objects/facts, e. g., nodes and edges, are the items of the *NEST* graph that are inserted into the working memory of the engine. This working memory, also called *Fact Base*, acts as a storage for all facts known to the engine [3].

Listing 1. Adaptation Rule for an Insert Adaptation Operator.

```
rule "INSERT === Streamlet 7818 (Egg)"
  when
     not DataNode("egg")
     $aC: TaskNode(isBestMatchingAnchor("mix", $aC, 1.0))
  then
     insertDataNode("egg", "piece", 1.0);
     insertTaskNode("slice");
     insertDataFlowEdge("slice", "egg");
     insertDataFlowEdge("egg", "slice");
     insertDataFlowEdge("egg", $aC);
     insertControlFlowEdge("slice", $aC);
end
```

As already described, a streamlet consists of a head data node, an anchor task, and all other nodes that are linked to the anchor task. To represent adaptation operators that consist of these concepts, we apply the predicates presented in Fig. 3. In the example depicted in Fig. 4, the streamlet on the right side of

Fig. 2 is encoded as an adaptation rule. In this example, we use the predicate $isBestMatchingAnchor$ to determine the best possible anchor which has a similarity of 1.0 to the streamlet anchor *mix* (see Fig. 2). Assuming that the graph task node *combine* satisfies this condition, it is stored in the variable $\$aC$. Subsequently, the plain streamlet consisting of the task *slice* and the data node one piece *egg* is inserted into the graph and connected to the other graph elements, including the newly identified anchor $\$aC$. While rules as the one depicted in Lst. 1 are generated automatically, they can also be edited or created manually within the bounds of the domain vocabulary and the DRL syntax. This aspect also elicits explainability of adaptation knowledge, as knowledge encoded in rules is perceived to be more intuitive [20] as well as it provides insights into the adaptation process and its properties.

3.2 Applying Adaptation Rules

After encoding the adaptation rules, they can be applied to a retrieved workflow graph for adaptation. Figure 5 illustrates the general inference mechanism of the *Drools* rule engine for searching and triggering adaptation rules to adapt a retrieved workflow graph. Each available and encoded rule is part of the *Fact Base* of *Drools*. In addition, the retrieved workflow graph is also part of the fact base, i. e., each graph element can be accessed to check or modify it during adaptation. To perform adaptations, *Drools* internally initiates an inference procedure that consists of several required steps for each adaptation rule contained in the fact base: 1) the preconditions encoded as predicates in a rule are accessed by the rule engine; 2) the graph elements of the retrieved workflow graph are checked whether they satisfy the preconditions of the corresponding rule; 3) if the preconditions of a rule are not satisfied, the next rule is checked. But if the preconditions are satisfied by the retrieved workflow, the rule is triggered by the rule engine; 4) the effects of the adaptation rule are executed, leading to modifications performed on the graph elements of the retrieved workflow. The

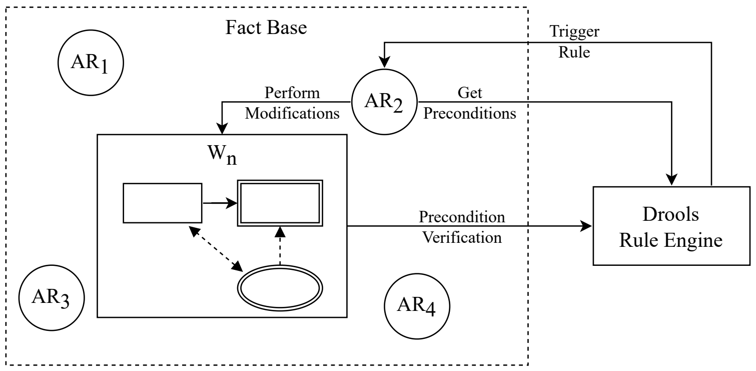

Fig. 5. General Inference Mechanism of the Rule Engine for Applying Adaptation Rules.

described search process is repeated by the rule engine until no further rule can be triggered, or the process is stopped externally, e. g., if the adaptation goal is already reached or the similarity as approximating function for the utility cannot be increased further. In general, it is difficult to find the best possible sequence of adaptation rules, i. e., the sequence that increases the similarity the most. Therefore, the best possible rules are determined step by step, similar to the application of adaptation operators (see Sect. 2.1). However, the application of adaptation operators considers each fragment of the retrieved graph only once, i. e., streamlets are sequentially created based on the data nodes of the workflow graph. In contrast, during adaptation with adaptation rules, the same fragment in the graph can be adapted more than once, depending on the availability of executable rules in the fact base. Finally, the rule engine needs to be aware of 1) the current state of the workflow (see Fig. 5), this involves all elements of the respective *NEST* workflow graph and, thus, all nodes and edges have to be inserted into the fact base; 2) every adaptation rule contained in the fact base; 3) all applicable adaptation rules for a given workflow state W_n by evaluating the preconditions of the rules; 4) the similarity improvement of the application of a rule w. r. t. the retrieved graph during rule evaluation.

4 Experimental Evaluation

In this section, we evaluate the rule-based approach in the domain of cooking recipes (see Sect. 2.1 for an example). Therefore, a direct comparison between the proposed approach and the existing adaptation operators [30] is conducted. In this context, it is important to note that Müller and Bergmann propose several POCBR adaptation methods (e. g., [28–30]). As the operator-based adaptation has the highest coverage, and it is the most sophisticated adaptation method, we apply it as the basis for the proposed approach in this work. The goal of the evaluation is to determine how well adaptations are performed by the rule-based approach and whether it leads to better and qualitative higher results. Differences in time and quality may arise from the varying utilization of adaptation knowledge in the application (see Sect. 3.2). For this purpose, we conduct an experiment by comparing the runtime properties of the rule-based adaptation compared to the adaptation operators and investigate the following hypotheses:

H1 The rule-based adaptation leads to syntactically correct adapted workflows, i. e., the resulting workflows are consistent *NEST* graphs.

H2 The rule-based adaptation results in equal or better adapted workflows w. r. t. the semantic similarity than only using the retrieved workflow without adaptation.

H3 The rule-based adaptation requires less adaptation knowledge to obtain similar improvements compared to the operator-based adaptation.

H4 Related to the operator-based adaptation, rule-based adaptation requires more time for adaptation, but achieves in relation much higher improvements, i. e., the improvement per time is higher.

Hypothesis H1 investigates whether the rule-based adaptation maintains syntactically correctness of the workflow graphs. This property is retained by the operator-based adaptation, and, thus, should also be satisfied by the rule approach. Similarly to this, Hypothesis H2 examines whether the rule-based approach leads to an improvement w. r. t. the similarity and not to a degradation. As the application of adaptation rules is not restricted to certain graph regions, we assume that less adaptation knowledge is required to obtain similar results compared to the adaptation operators (see Hypothesis H3). In addition, due to the different utilization of adaptation knowledge, and, thus, more executed rules in total than adaptation operators, we expect that the rule-based adaptation requires significant more adaptation time compared to the adaptation operators (see Hypothesis H4). However, it is expected that the improvement is higher in relation to the additional time needed, i. e., the improvement per time is higher.

4.1 Experimental Setup

In the following, we describe the experimental setup to validate the hypotheses. We perform a Leave One Out Cross Validation with a case base of 40 distinct cooking workflows. In this context, the cases in the case base contain on average 21.15 graph nodes and 55.63 graph edges. For the experiment, the rule-based adaptation approach is implemented in the POCBR framework ProCAKE[5] [6] using the *Drools* rule engine[6]. For each of the 40 workflows, we perform a retrieval on the remaining 39 recipes in the case base. Based on that, the retrieved workflows serve as a starting point for adaptation. The required adaptation knowledge is learned automatically based on the remaining case base by using the learning approach for the adaptation operators [30]. In this context, it is important to note that we ensure the separation of training and test cases by constructing a separate operator repository for every evaluation run. More precisely, the particular query workflow is omitted during learning of the respective repository. On average, the resulting repositories contain 2622.85 operators in total, out of which 1840.7 are exchange operators, 740.4 insert operators, and 41.75 delete operators. Based on these operator repositories, adaptation rules are encoded according to the concepts described in Sect. 3.1. To ensure comparable results, the same parameters that have been used during learning are also used for applying the rules and operators. This means that the *Anchor Candidate Threshold* Δ_T and the *Streamlet Similarity Threshold* Δ_S are set to be 0.5 for learning and applying the operators afterwards. Furthermore, the experiment is performed on growing fractions of the learned adaptation knowledge to observe effects related to the existing amount of adaptation knowledge regarding adaptation quality and time. Therefore, 20, 40, 60, and 80 percent of the operator repository are extracted randomly, and the experiment is conducted on each of those fractions independently. Moreover, we perform three runs for each fraction with different

[5] https://procake.uni-trier.de
[6] The implementation can be found at: https://gitlab.rlp.net/procake/procake-rule-engine

random seeds. For each of those runs, the minimum, maximum, and average as well as the median of the similarity gain and the adaptation time are recorded.

4.2 Experimental Results

Table 1 illustrates the results of the experiment[7]. All differences are statistically significant with $p < 0.05$ for runs conducted with 20% and $p < 0.01$ for the other fractions. We measured the improvement gained by adaptation in percent (see Column *Imp*) based on the semantic similarity measure (see Sect. 2.1), the time needed for adaptation in seconds (see Column *Time*), and the improvement per time, i. e., the improvement per second (see Column *Imp/s*). Furthermore, we evaluate several fractions of potential usable adaptation knowledge, i. e., 20 %, 40 %, 60 %, 80 %, and 100 %. For every run with 40 adaptations, we calculate the values for all adaptations and represent them as minimum, maximum, average, and median. All performed adaptations either by the rule engine or by the adaptation operators lead to syntactically correct workflows, showing that the knowledge is valid and the application of it is correct. Thus, we accept Hypothesis H1. In Hypothesis H2, we investigate whether the application of the rule-based adaptation does not lead to a degradation of the retrieval result, i. e., the semantic similarity remains at least the same or increases after adaptation. As can be determined in the first row of Tab. 1, the minimum improvement is 0, indicating that at worst the adaptation leads to the retrieved workflow without changing anything. The same also holds for the operator-based adaptation. Consequently, we accept Hypothesis H2. Considering Hypothesis H3 which addresses the amount of potential adaptation knowledge, it can be determined that if less adaptation knowledge is available, the rule-based adaptation results in higher improvements regarding the semantic similarity (see Column *Avg* and

Table 1. Results of the Experimental Evaluation.

		Amount of Potential Adaptation Knowledge														
			20%			**40%**			**60%**			**80%**			**100%**	
		Imp	**Time**	**Imp/s**	**Imp**	**Time**	**Imp/s**	**Imp**	**Time**	**Imp/s**	**Imp**	**Time**	**Imp/s**	**Imp**	**Time**	**Imp/s**
Rules	Min	0	0.3	0	0	0.6	0	0	2.1	0	0.02	2.6	0	0.02	3.4	0
	Max	29.28	208.3	14.59	31.52	493.6	4.21	30.67	1036.3	2.85	32.25	1497.4	2.12	31.52	1466	1.68
	Avg	9.78	29.1	1.04	11.99	74.6	0.62	13.02	111.7	0.45	13.69	162.4	0.35	14.27	208.4	0.28
	Mdn	9.42	11.47	0.43	11.91	23.93	0.24	13.74	42.3	0.17	14.44	53.2	0.14	14.95	85	0.12
Operators	Min	0	0.3	0	0	0.6	0	0	0.8	0	0	1.1	0	0	1.5	0
	Max	13.9	4.5	20.68	21.82	5.6	10.17	11.53	6.7	6.11	13.12	5.8	4.63	13.58	5.9	2.75
	Avg	2.38	1.1	2.26	2.59	1.6	1.35	2.59	2.2	1.07	2.78	2.6	0.91	2.83	3.2	0.76
	Mdn	0.58	0.8	0.49	0.81	1.3	0.58	1.18	2	0.53	1.51	2.5	0.65	1.62	2.9	0.55

[7] All evaluation results and descriptions of how to reproduce the experiment with the proposed implementation can be found at: https://gitlab.rlp.net/procake/publications/procake-rule-engine-iccbr-2024

Mdn for 20 %). In general, the results also indicate that the quality of adaptation increases with more potential usable adaptation knowledge, but more for the rules and only slightly for the operators. Consequently, the adaptation operators only lead to a small increase in improvement, even if the amount of adaptation knowledge increases. As the improvements are much higher for the rule-based adaptation compared to the adaptation operators with 20 % of knowledge, we accept Hypothesis H3. However, the results also reveal that the rule-based adaptation requires much more time for adaptation than the adaptation operators. Examining the calculated improvements per second, it can be determined that the rule-based adaptation has mostly lower improvements per second than the adaptation operators. As a conclusion, the rule-based adaptation leads to significantly better results, but also requires considerably more time for doing that than the adaptation operators. For this reason, we reject Hypothesis H4.

5 Conclusion and Future Work

In this paper, we propose an approach for using adaptation rules and a rule engine for performing complex adaptations in Process-Oriented Case-Based Reasoning (POCBR). Based on previous adaptation methods for POCBR, we develop suitable predicates that are used for encoding adaptation knowledge as adaptation rules. This prevents the modeling and definition of adaptation knowledge in the form of complex graph fragments. Thus, it is now more feasible for domain experts to model workflow adaptation knowledge in POCBR as adaptation rules[8]. In addition, the experimental evaluation results demonstrate that the rule-based adaptation leads to syntactically correct workflows, and indicates that the improvement is significantly higher than for the operator-based adaptation [30]. By the proposed approach, we contribute to the complex task of workflow adaptation in POCBR and help to reduce the laborious and time-consuming task of modeling and maintaining this adaptation knowledge.

In future work, we want to implement further adaptation methods for POCBR (e. g., [28, 29]) by using adaptation rules. Moreover, the current implementation is based on the *Drools* rule engine, but the conceptual idea and the use of special predicates for representing the concepts of adaptation operators is also transferable to other available rule engines. In this context, it is also interesting to investigate how the rule-based approach can be integrated into other CBR and POCBR frameworks developed with other programming languages (e. g., Python [19]) instead of Java. Furthermore, the *Drools* rule engine also offers a graphical user interface for manually modeling and maintaining adaptation rules. Thus, we want to investigate how such a user interface can be integrated into the *ProCAKE* framework. Another topic for further research in this context is to examine how large language models can help in creating and modeling adaptation knowledge in CBR and POCBR [10]. In the experiments, we use the

[8] An initial user study provides evidence to support this claim. The results of this user study can be found at: https://gitlab.rlp.net/procake/publications/procake-rule-engine-iccbr-2024.

domain of cooking recipes as procedural semantic workflows. In future work, we intend to investigate also other workflow domains, such as for scientific workflows [36] or for cyber-physical workflows [21,22]. Both domains have in common that they often require complex, structural adaptations to ensure executability of the processes. Moreover, it should be examined how well the proposed rule-based adaptation approach can be applied to other case representations, such as time series in temporal CBR domains [23]. There are also possibilities for further improvement in the evaluation and application of the rules. For example, we are planning to use especially tailored search and optimization techniques for rule engines, which should further improve the use and application of adaptation rules, leading to even better workflow adaptation results.

Acknowledgments. This work is funded by the Federal Ministry for Economic Affairs and Climate Action under grant No. 01MD22002C *EASY*. The authors thank Veronika Kurchyna and Vincent Muljadi for an initial concept and prototypical implementation of the rule-based approach developed during a student research project.

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994)
2. Bach, K., Althoff, K.: Developing Case-Based Reasoning Applications Using myCBR 3. In: 20th ICCBR. LNCS, vol. 7466, pp. 17–31. Springer (2012)
3. Bali, M.: Drools JBoss Rules 5.0 developer's guide: Develop rules-based business logic using the Drools platform. From technologies to solutions, Packt Publ (2009)
4. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNCS, vol. 2432. Springer (2002)
5. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014)
6. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: 27th ICCBR Workshops (2019)
7. Bergmann, R., Kolodner, J.L., Plaza, E.: Representation in case-based reasoning. Knowl. Eng. Rev. **20**(3), 209–213 (2005)
8. Bergmann, R., Wilke, W., Vollrath, I., Wess, S.: Integrating General Knowledge with Object-Oriented Case Representation and Reasoning. In: 4th German Workshop: Case-Based Reasoning - System Development and Evaluation. pp. 120–126. Humboldt-Universität Berlin, Informatik-Berichte Nr. 55 (1996)
9. Berner, E.S.: Clinical Decision Support Systems: Theory and Practice. Springer, 2nd edn. (2007)
10. Brand, F., Malburg, L., Bergmann, R.: Large Language Models as Knowledge Engineers. In: Proceedings of the Workshops at the 32nd International Conference on Case-Based Reasoning (ICCBR-WS 2024) co-located with the 32nd International Conference on Case-Based Reasoning (ICCBR 2024), Merida, Mexico, July 1, 2024. CEUR Workshop Proceedings, CEUR-WS.org (2024), Accepted for Publication.
11. Cabrera, M.M., Edye, E.O.: Integration of Rule Based Expert Systems and Case Based Reasoning in an Acute Bacterial Meningitis Clinical Decision Support System. CoRR **abs/1003.1493** (2010)

12. Craw, S., Wiratunga, N., Rowe, R.: Learning adaptation knowledge to improve case-based reasoning. Artif. Intell. **170**(16-17), 1175–1192 (2006)
13. Döhring, M., Zimmermann, B., Godehardt, E.: Extended Workflow Flexibility using Rule-Based Adaptation Patterns with Eventing Semantics. In: INFORMATIK 2010. LNI, vol. P-175, pp. 195–200. GI (2010)
14. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3-4), 189–208 (1971)
15. Hanft, A., Schäfer, O., Althoff, K.D.: Integration of Drools into an OSGI-based BPM-Platform for CBR. In: 19th ICCBR Workshops (2011)
16. Hanney, K., Keane, M.T.: Learning Adaptation Rules From a Case-Base. In: 3rd EWCBR. pp. 179–192. LNCS, Springer (1996)
17. Hanney, K., Keane, M.T.: The Adaptation Knowledge Bottleneck: How to Ease it by Learning from Cases. In: 2nd ICCBR. Springer (1997)
18. Leake, D.B.: CBR in Context: The Present and Future, chap. 1, pp. 3–30. AAAI Press/MIT Press (1996)
19. Lenz, M., Malburg, L., Bergmann, R.: CBRkit: An Intuitive Case-Based Reasoning Toolkit for Python. In: 32nd ICCBR. LNCS, vol. 14775, pp. 289–304. Springer (2024)
20. Ligeza, A.: Logical Foundations for Rule-Based Systems, Studies in Computational Intelligence, vol. 11. Springer (2006)
21. Malburg, L., Brand, F., Bergmann, R.: Adaptive Management of Cyber-Physical Workflows by Means of Case-Based Reasoning and Automated Planning. In: 26th EDOC Workshops. LNBIP, vol. 466, pp. 79–95. Springer (2023)
22. Malburg, L., Hoffmann, M., Bergmann, R.: Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. pp. 1–29 (2023)
23. Malburg, L., Schultheis, A., Bergmann, R.: Modeling and Using Complex IoT Time Series Data in Case-Based Reasoning: From Application Scenarios to Implementations. In: 31st ICCBR Workshops. vol. 3438, pp. 81–96. CEUR-WS.org (2023)
24. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards Case-Based Adaptation of Workflows. In: 18th ICCBR. LNCS, vol. 6176, pp. 421–435. Springer (2010)
25. Minor, M., Bergmann, R., Görg, S.: Case-based adaptation of workflows. Inf. Syst. **40**, 142–152 (2014)
26. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014)
27. Müller, G.: Workflow Modeling Assistance by Case-based Reasoning. Springer (2018)
28. Müller, G., Bergmann, R.: Workflow Streams: A Means for Compositional Adaptation in Process-Oriented CBR. In: 22nd ICCBR. LNCS, vol. 8765, pp. 315–329. Springer (2014)
29. Müller, G., Bergmann, R.: Generalization of Workflows in Process-Oriented Case-Based Reasoning. In: 28th FLAIRS. pp. 391–396. AAAI Press (2015)
30. Müller, G., Bergmann, R.: Learning and Applying Adaptation Operators in Process-Oriented Case-Based Reasoning. In: 23rd ICCBR. LNCS, vol. 9343, pp. 259–274. Springer (2015)
31. Pachet, F.: Reasoning with objects: the néopus environment. In: Int. Conf. East EurOOpe (1991)
32. Schultheis, A., Zeyen, C., Bergmann, R.: An Overview and Comparison of CBR Frameworks. In: 31st ICCBR. LNCS, vol. 14141, pp. 327–343. Springer (2023)

33. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In: 7th ECCBR. LNCS, vol. 3155, pp. 434–448. Springer (2004)
34. Wilke, W., Bergmann, R.: Techniques and Knowledge Used for Adaptation During Case-Based Problem Solving. In: 11th IEA/AIE. pp. 497–506. LNCS, Springer (1998)
35. Wilke, W., Vollrath, I., Althoff, K.D., Bergmann, R.: A Framework for Learning Adaptation Knowledge Based on Knowledge Light Approaches. In: 5th GWCBR. pp. 235–242 (1997)
36. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: 27th ICCBR. LNCS, vol. 11680, pp. 388–403. Springer (2019)