Learning and Applying Adaptation Operators in Process-Oriented Case-Based Reasoning

Gilbert Müller(✉) and Ralph Bergmann

Business Information Systems II, University of Trier, 54286 Trier, Germany
{muellerg,bergmann}@uni-trier.de
http://www.wi2.uni-trier.de

Abstract. This paper presents a novel approach to the operator-based adaptation of workflows, which is a specific type of transformational adaptation. We introduce the notion of workflow adaptation operators which are partial functions transforming a workflow into a successor workflow, specified by workflow fractions to be inserted and/or deleted. The adaptation process itself chains adaptation operators during a local search process aiming at fulfilling the query as best as possible. Further, the paper presents an algorithm that learns workflow adaptation operators from the case base automatically, thereby addressing the common problem of adaptation knowledge acquisition. An empirical evaluation in the domain of cooking workflows was conducted which demonstrates convincing adaptation capabilities without a significant reduction of the workflows' quality.

Keywords: Process-oriented case-based reasoning · Operator-based adaptation · Workflows

1 Introduction

Process-aware information systems (PAISs) [7] support the operational business of an organization based on models of their processes. PAISs include traditional workflow management systems as well as modern business process management systems. In the recent years, the use of workflows has significantly expanded from the original business area towards new application fields such as e-science, medical healthcare, information integration, and even cooking [10,24,25]. Process-oriented case-based reasoning (POCBR) [20] covers research on case-based reasoning (CBR) for addressing problems in PAISs. Recent research deals with approaches to support modeling, composition, adaptation, analysis, monitoring and optimization of business processes or workflows [2,12,13,18,21,22,26]. Workflow adaptation addresses the adaptation of a retrieved workflow from a repository (case base) to fulfill the specific needs of a new situation (query). In POCBR, adaptation methods that originate from case adaptation in CBR are proposed for this purpose. In our previous work we have investigated case-based adaptation [17], compositional adaptation [22], as well as the use of generalized cases [23] for adaptation.

E. Hüllermeier and M. Minor (Eds.): ICCBR 2015, LNAI 9343, pp. 259–274, 2015.
DOI: 10.1007/978-3-319-24586-7_18

In this paper, we present a novel operator-based approach [3] for adapting workflow cases represented as graphs. The *workflow adaptation operators* we propose in Sect. 3 are partial functions specifying ways of adapting a workflow towards a successor workflow. Like in STRIPS, the operators are specified by two workflow sub-graphs, one representing a workflow fraction to be deleted and one representing a workflow fraction to be added. The adaptation process (see Sect. 5) transforms a retrieved workflow into an adapted workflow by chaining various adaptation operators. This process can be considered a search process towards an optimal solution w.r.t. the query. Most importantly, we also propose an algorithm to learn such workflow adaptation operators automatically from the case base, thereby extending previous work on learning adaptation knowledge [4,9,27] towards POCBR (see Sect. 4). Thus, the knowledge acquisition bottleneck for adaptation knowledge is avoided. Finally, we experimentally evaluate our approach in the domain of cooking (see Sect. 6). We can show that with the learned workflow adaptation operators, a high percentage of the changes requested for a retrieved workflow can be fulfilled without significantly reducing the quality of the adapted workflows.

2 Foundations

We now briefly introduce relevant previous work in the field of POCBR.

2.1 Workflows

Broadly speaking, a *workflow* consists of a set of *activities* (also called *tasks*) combined with *control-flow structures* like sequences, parallel (AND) or alternative (XOR) branches, as well as repeated execution (LOOPs). In addition, tasks exchange certain *data items*, which can also be of physical matter, depending on the workflow domain. Tasks, data items, and relationships between the two of them form the *data flow*.

We illustrate our approach in the domain of cooking recipes (see example workflow in Fig. 1). A cooking recipe is represented as a workflow describing the instructions for cooking a particular dish [24]. Here, the tasks represent the cooking steps and the data items refer to the ingredients being processed by the cooking steps. An example cooking workflow for a pasta recipe is illustrated in Fig. 1. Based on our previous work [2,22,23] we now introduce the relevant formal workflow terminology.

Definition 1. *A* workflow *is a directed graph* $W = (N, E)$ *where* N *is a set of nodes and* $E \subseteq N \times N$ *is a set of edges. Nodes* $N = N^D \cup N^T \cup N^C$ *can be data nodes* N^D*, task nodes* N^T*, or control-flow nodes* N^C*. In addition, we call* $N^S = N^T \cup N^C$ *the set of sequence nodes. Edges* $E = E^C \cup E^D$ *can be control-flow edges* $E^C \subseteq N^S \times N^S$*, which define the order of the sequence nodes or data-flow edges* $E^D \subseteq (N^D \times N^S) \cup (N^S \times N^D)$*, which define how the data is shared between the tasks.*

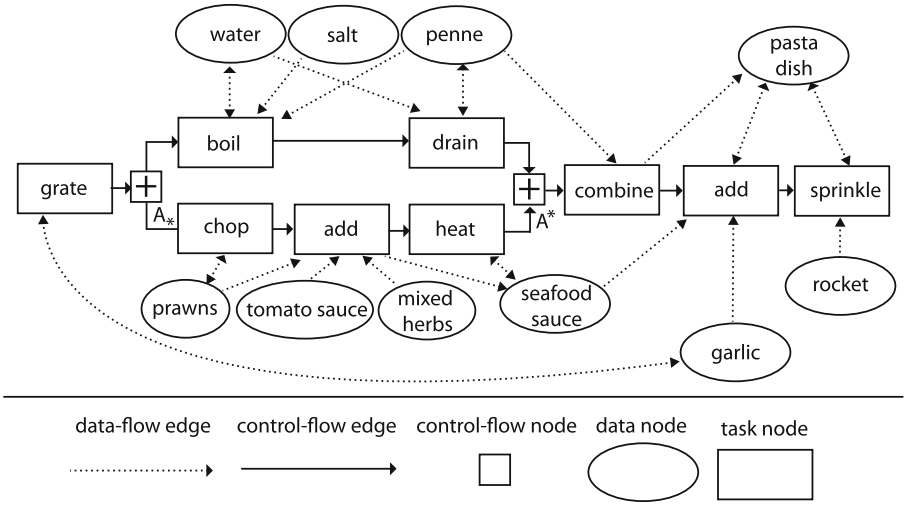

Fig. 1. Example of a block-oriented cooking workflow

The control-flow edges E^C of a workflow induce a strict partial order on the sequence nodes S. Thus, we define $s_1 < s_2$ for two sequence nodes $s_1, s_2 \in S$ as a transitive relation that expresses that s_1 is executed prior to s_2 in W. We further define $n \in]x_1, x_2[$ iff $x_1 < n < x_2$, describing that node n is located between x_1 and x_2 in W w.r.t. the control-flow edges.

We further denote that two data nodes $d_1, d_2 \in N^D$ are data-flow connected $d_1 \ltimes d_2$ if there exists a task that consumes data node d_1 and produces data node d_2. Moreover, $d_1 \ltimes^+ d_2$ denotes that $d_1, d_2 \in N^D$ are transitively data-flow connected:

$$d_1 \ltimes d_2, \text{iff } \exists t \in N^T : ((d_1, t) \in E^D \wedge (t, d_2) \in E^D) \tag{1}$$

$$d_1 \ltimes^+ d_2, \text{iff } d_1 \ltimes d_2 \vee \exists d \in N^D : (d_1 \ltimes d \wedge d \ltimes^+ d_2) \tag{2}$$

2.2 Block-Oriented Workflows

We now restrict the workflow representation to block-oriented workflows [22], i.e., workflows in which the control-flow structures form blocks of nested workflows with an opening and closing control-flow element. These blocks must not be interleaved.

Definition 2. *A* block-oriented workflow *is a workflow in which the control-flow nodes $N^C = N^{C_*} \cup N^{C^*}$ define the control-flow blocks. Each control-flow block has an opening node from N^{C_*} and a related closing node from N^{C^*} specifying either an AND, XOR, or LOOP block. These control-flow blocks may be nested but must not be interleaved and must not be empty.*

Figure 1 shows an example block-oriented workflow, containing a control-flow block with an opening AND control-flow node A_* and a related closing AND control-flow node A^*.

Further, we introduce a terminology of *consistent block-oriented workflows*. According to Davenport, "[...] a process is simply a structured, measured set of activities designed to produce a specific output [...]" [5]. In the following, these specific workflow outputs are denoted as $W^O \subseteq N^D$. In the cooking domain, the specific output is the particular dish produced, i.e., "pasta dish" in Fig. 1. Hence, for a *consistent workflow*, we require that each ingredient must be contained in the specific output, as otherwise the ingredient as well as the related tasks would be superfluous.

Definition 3. *A block-oriented workflow is* consistent*, iff each produced ingredient is contained in the specific output of the workflow. Thus, each ingredient must be transitively data-flow connected to the specific output W^O, i.e.,* $\forall d \in N^D \exists o \in W^O d \ltimes^+ o$.

2.3 Semantic Workflow Similarity

To support retrieval and adaptation of workflows, the individual workflow elements are annotated with ontological information, thus leading to a *semantic workflow* [2]. In particular, all task and data items occurring in a domain are organized in taxonomy, which enables the assessment of similarity among them. We deploy a taxonomy of cooking ingredients and cooking steps for this purpose. In our previous work, we developed a semantic similarity measure for workflows that enables the similarity assessment of a case workflow w.r.t. a query workflow [2].

The core of the similarity model is a local similarity measure for semantic descriptions $sim_\Sigma : \Sigma^2 \to [0,1]$. In our example domain the taxonomical structure of the data and task ontology is employed to derive a similarity value that reflects the closeness in the ontology. It is combined with additional similarity measures that consider relevant attributes, such as the quantity of an ingredient used in a recipe (see [2] for more details and examples). The similarity $sim_N : N^2 \to [0,1]$ of two nodes and two edges $sim_E : E^2 \to [0,1]$ is then defined based on sim_Σ applied to their assigned semantic descriptions. The similarity $sim(QW, CW)$ between a query workflow QW and a case workflow CW is defined by means of an admissible mapping $m : N_q \cup E_q \to N_c \cup E_c$, which is a type-preserving, partial, injective mapping function of the nodes and edges of QW to those of CW. For each query node and edge x mapped by m, the similarity to the respective case node or edge $m(x)$ is computed by $sim_N(x, m(x))$ and $sim_E(x, m(x))$, respectively. The overall workflow similarity with respect to a mapping m, named $sim_m(QW, CW)$ is computed by an aggregation function (e.g. a weighted average) combining the previously computed similarity values. The overall workflow similarity is determined by the best possible mapping m

$$sim(QW, CW) = \max\{sim_m(QW, CW) \,|\, \text{admissible mapping}\, m\}.$$

This similarity measure assesses how well the query workflow is covered by the case workflow. In particular, the similarity is 1 if the query workflow is exactly included in the case workflow as a subgraph. Hence, this similarity measure is not symmetrical.

2.4 Partial Workflows and Streamlets

We aim at reusing workflow parts within the representation of adaptation operators. Therefore, we now introduce the definition of partial workflows according to Müller and Bergmann [22] and the new definition of so-called streamlets.

Definition 4. *For a subset of tasks* $N_p^T \subseteq N^T$*, a* partial workflow W_p of a block-oriented workflow $W = (N, E)$ *is a block-oriented workflow* $W_p = (N_p, E_p \cup E_p^{C+})$ *with a subset of nodes* $N_p = N_p^T \cup N_p^C \cup N_p^D \subseteq N$*.* $N_p^D \subseteq N^D$ *is defined as the set of data nodes that are linked to any task in* N_p^T*,i.e.,* $N_p^D = \{d \in N^D | \exists t \in N_p^T : ((d,t) \in E^D \vee (t,d) \in E^D)\}$*.* $N_p^C \subseteq N^C$ *is the maximum set of control-flow nodes such that* W_p *is a correct block-oriented workflow.* W_p *contains a subset of edges* $E_p = E \cap (N_p \times N_p)$ *connecting two nodes of* N_p *supplemented by a set* E_p^{C+} *of additional control-flow edges that retain the execution order of the sequence nodes, i.e.,* $E_p^{C+} = \{(n_1, n_2) \in N_p^S \times N_p^S | n_1 < n_2 \wedge \nexists n \in N_p^S : ((n_1, n) \in E_p^C \vee (n, n_2) \in E_p^C \vee n \in]n_1, n_2[)\}$*.*

In general, control-flow nodes are part of a partial workflow if they construct a workflow w.r.t. the block-oriented workflow structure. The additional edges E_p^{C+} are required, to retain the execution order $s_1 < s_3$ of two sequence nodes if for $s_1, s_2, s_3 \in S$ holds $s_2 \in]s_1, s_3[$ but $s_2 \notin N_p$. Figure 2 illustrates a partial workflow W_p of the workflow W given in Fig. 1. One additional edge is required in this example, depicted by the double-line arrow since "grate" and "add" are not linked in W.

Based on Definition 4, we now introduce *streamlets* that represent a partial workflow constructed by all tasks linked to a certain data node $d \in N^D$. Hence, a streamlet describes the partial workflow comprising the tasks processing a certain data node d. Thus, it is the smallest fraction of a workflow regarding d. Streamlets will become the smallest fraction of a workflow to be modified by workflow adaptation operators.

Definition 5. *A* streamlet $W_d = (N_d, E_d)$ *for data* $d \in N^D$ *in workflow* W *is defined as a partial workflow for the subset of tasks connected to* d*, i.e.* $\{t \in N^T | (t,d) \in E^D \vee (d,t) \in E^D\}$*. The data node* d *is referred to as the head data node of* W_d*. Further, let the tasks in* W_d *that do not produce* d *be defined as anchor tasks* A_d *for* d*, i.e.,* $A_d = \{t \in N_d^T | \nexists (t,d) \in E_d^D\}$*.*

An example streamlet is illustrated in Fig. 2. In general, anchor tasks (see double-lined rectangle, task node "add") are those tasks that consumes the head data node (see double-lined circle, data node "garlic") but that do not produce it. Hence, these tasks mark the positions where the head data node is linked into

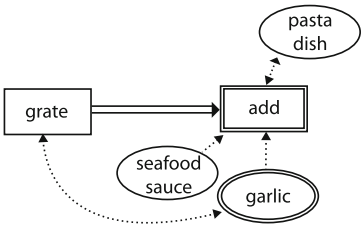

Fig. 2. Example of a streamlet W'

the overall workflow and used as an input to other tasks (e.g. after adding garlic it is used together with the seafood sauce as part of the pasta dish). Please note that a streamlet may contain more than one anchor, e.g., salt can be added to the boiled water and to the pasta sauce.

3 Workflow Adaptation Operators

In CBR, we usually distinguish between substitutional, transformational, and derivational adaptation approaches [4,27]. Substitutional and transformational adaptation approaches make use of adaptation knowledge represented as adaptation rules or adaptation operators. Adaptation rules describe how differences between the problem description in the query and the retrieved case (rule's precondition) can be compensated by certain changes of the solution in the retrieved case (rule's conclusion). Adaptation operators [3] however, do not explicitly represent differences between query and retrieved case, but they are partial functions specifying ways of adapting a case towards a successor case. The adaptation process in CBR transforms a retrieved case into an adapted case by chaining various adaptation operators. Consequently, workflow adaptation is performed by applying chains of adaptation operators $W \overset{o_1}{\rightarrow} W_1 \overset{o_2}{\rightarrow} \ldots \overset{o_n}{\rightarrow} W_n$ to the retrieved workflow W, thereby computing the adapted workflow W_n. This process can be considered a search process towards an optimal solution w.r.t. the query.

3.1 Definition of Workflow Adaptation Operators

For applying operator-based adaptation to POCBR, a notion of workflow adaptation operators is required defining them as partial functions that transform workflows. We loosely follow the representation idea of STRIPS operators and define an adaptation operator by specifying an insertion and a deletion streamlet. Operator preconditions are not explicitly specified, but result implicitly from those streamlets. In a nutshell, a workflow adaptation operator, if applicable, removes the deletion streamlet from the workflow and adds the insertion streamlet instead. In Fig. 3 an example of an operator is shown, specifying that in a cooking workflow prawns can be replaced by tuna, while at the same time the preparation step chop needs to be replaced by drain. Besides operators that

exchange workflow streamlets, they could also just insert or just delete a streamlet of a workflow. We now give a formal definition of workflow adaptation operators, specifying their representation and operational semantics.

Definition 6. *Let $\mathcal{W}$ be the set of all consistent block-oriented workflows. A workflow adaptation operator is a partial function $o : \mathcal{W} \mapsto \mathcal{W}$ transforming a workflow $W \in \mathcal{W}$ into an adapted workflow $o(W) \in \mathcal{W}$. The adaptation operator o is specified by an insertion streamlet o_I and a deletion streamlet o_D, each of which can also be empty. Based on the presence of the two streamlets, we distinguish three types of adaptation operators.*

- *An* insert operator *consists only of an insertion streamlet with the anchor tasks A_d. The application of o to W inserts o_I into W except for the anchor tasks A_d at the positions of the best matching anchor tasks A_d. The operator is only applicable (precondition A) iff in W tasks matching A_d exist and if the resulting workflow is consistent.*
- *A* delete operator *consists only of a deletion streamlet o_D with the head data node d and the anchor tasks A_d. The application of o to W deletes the streamlet W_d from W except for the anchor tasks A_d. The operator is only applicable (precondition B) iff there exists a workflow streamlet W_d in W which is sufficiently similar to o_D and if the resulting workflow after deletion is consistent.*
- *An* exchange operator *consists of an insertion and a deletion streamlet. The application of o to W deletes o_D from W (except for the anchor tasks) and subsequently inserts o_I (except for the anchor tasks) at the position of the best matching anchor tasks. The operator is only applicable iff both previously defined preconditions A and B are fulfilled and if the resulting workflow after deletion and insertion is consistent.*

The conditions of identical head node and the minimum similarity between the streamlet and the deletion streamlet serve as a precondition to check whether the streamlet W_d is similar enough to the deletion streamlet. This ensures that operators are only applied if a similar streamlet is present in the workflow. For the insertion, a matching anchor is needed to ensure that the streamlet can be added to a suitable position of the workflow merging in the right data node.

An example of a workflow exchange adaptation operator o is given in Fig. 3. The head data nodes are marked by a double-circled data object, i.e., tuna or prawns, respectively. Further, the anchor tasks are marked by double-lined rectangles. These anchor tasks are used during adaptation, to identify the position of the streamlet within the entire workflow, i.e., the position at which the insertion streamlet is inserted. Hence, the example adaptation rule describes that prawns can be exchanged by tuna, if in W a streamlet W_d similar to o_D is present. This also enforces that tasks have to be changed as well, because the chop task also has to be exchanged by a drain task.

3.2 Details of the Operator Application

We now give some more details about how operators are applied, making the previous definition more precise. The space limitation prevents us from a detailed

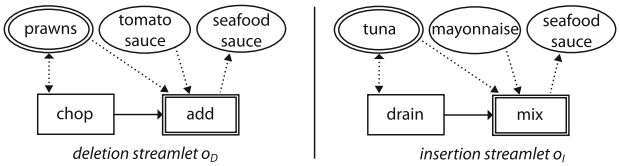

Fig. 3. Example adaptation operator

description of the algorithm. Hence, we illustrate our approach primarily by an example application of the exchange operator shown in Fig. 3 to the workflow given in Fig. 1.

To determine the applicability of a delete or exchange operator o for a workflow W, the definition requires that there exists workflow stream W_d in W for the head node d of o_D that is sufficiently similar to the deletion streamlet o_D. We implement this condition by a similarity threshold Δ_S, i.e., we require that $sim(W_d, o_D) \geq \Delta_S$. Further, we require that the output data nodes of the anchor tasks of o_D are the same as the output data nodes of the mapped tasks in W, ensuring that the data node d is removed from the same successive data nodes (e.g. see "seafood sauce" in Fig. 3). To remove the streamlet W_d from W (see Fig. 4 for an example) a partial workflow is constructed, containing all tasks of W except of the non-anchor tasks contained in W_d, i.e., $N^T \setminus (N_d^T \setminus A_d)$.

To add the insertion streamlet o_I to W, tasks in W must be identified that match the anchor tasks A_d of o_I. For this purpose, the partial workflow constructed from o_I for the anchor tasks A_d is considered. This partial workflow contains the anchor tasks as well as all connected data nodes. To match the anchor, the similarity between this partial workflow and W is determined. If this similarity exceeds the threshold Δ_S, the matching tasks in W are determined by the computed admissible mapping function. Further, we require that the output data nodes of the anchor tasks A_d are the same as the output data nodes of the mapped tasks in W. After a successful anchor mapping, the insertion streamlet is added at the position of the best matching anchor. This means

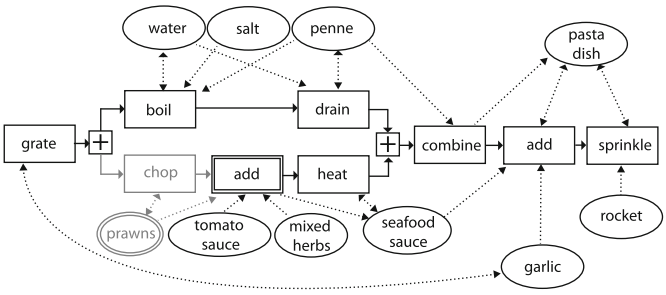

Fig. 4. Streamlet W_d removed from W

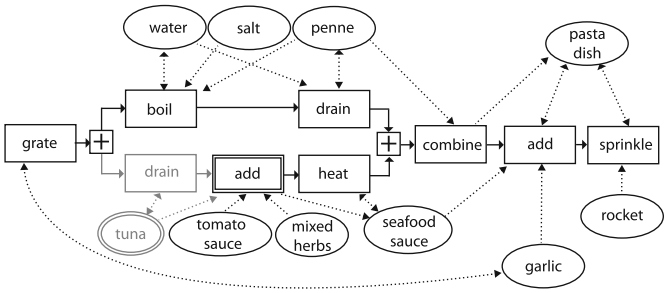

Fig. 5. Streamlet R_I added to W

that all edges, tasks (except of anchors), and data nodes (if not already present) of o_I are inserted into the workflow W. Then, the inserted streamlet o_I is connected with an additional control-flow edge that link the tasks of the streamlet into the workflow in front of the best matching anchor in W. In the illustrated scenario, the streamlet o_I is inserted in front of the first "add" task (see Fig. 5). In the special case that the insertion streamlet contains more than one anchor, the streamlet is split. To this end, for each anchor, the tasks are determined that belong to this anchor, i.e., previous tasks (w.r.t. to control-flow edges), which are transitively data flow connected until another anchor is reached. The procedure is then applied for each part of the split streamlet.

4 Automatic Learning of Workflow Adaptation Operators

The basic idea behind the automatic learning of workflow adaptation operators (see Algorithm 1) is to explore the knowledge already present in the case base [4,9,27]. To achieve this, pairs of similar workflows in the case base are compared, i.e., a query workflow W_q and a case workflow W_c that have a similarity value higher than a given threshold Δ_W, i.e. $sim(W_q, W_c) \geq \Delta_W$. The adaptation operators are then constructed from the differences in the data nodes between those two workflows. Hence, they describe which data nodes have to be exchanged, inserted or deleted in order to transform the set of data nodes N_q^D to N_c^D. The differences are determined by accessing the mapping produced during the similarity computation (see Sect. 2.3). More precisely, the data nodes of a query workflow are mapped to those of the case workflow. Mapped data nodes are then assumed to be replaceable data nodes. However, only mappings are regarded between data nodes that have a similarity value higher than a threshold Δ_D. This ensures that data nodes that are not similar to each other are not considered as replaceable. For each mapping, two streamlets are constructed based on the corresponding query data node and case data node. We thereby assume that these two streamlets can be exchanged by each other, i.e., the query streamlet represents the deletion streamlet and the case streamlet represents the insertion streamlet. For the remaining data nodes (the ones with no mapping with a similarity value larger than Δ_D), insert or delete operators are created.

Algorithm. LEARN_OPERATORS(CB);
Input: Case base CB
Output: Set of operators *operators*
$operators = \emptyset$;
forall the $W_q \in CB$ **do**
  forall the $\{W_c \in CB | sim(W_q, W_c) \geq \Delta_W \wedge W_q \neq W_c\}$ **do**
    forall the $d \in N_q^D$ **do**
      Init operator o;
      $o.insert = \emptyset$;
      $o.delete = construct\_streamlet(W_q, d)$;
      if $sim(d, m(d)) \geq \Delta_D$ **then**
        $o.insert = construct\_data\_streamlet(W_c, d)$;
      $operators = operators \cup \{o\}$;
    forall the $\{d \in N_c^D | \nexists d' \in N_q^D : (m(d') = d \vee sim(d', d) \geq \Delta_D)\}$ **do**
      Init operator o;
      $o.delete = \emptyset$;
      $o.insert = construct\_streamlet(W_c, d)$;
      $operators = operators \cup \{o\}$;
return *operators*

Algorithm 1: Learning algorithm of workflow adaptation operators

Although, the operators describe how to transform the set of data nodes N_q^D to N_c^D the workflow streamlet also contains information about how to exchange delete, or insert tasks or control-flow nodes. This is because identical data nodes that are mapped can possibly be processed differently, i.e., different task nodes are used in their streamlets to process this data.

5 Workflow Adaptation Using Adaptation Operators

We now present the adaptation procedure in more detail. After the retrieval of a most similar workflow W the user might want to adapt it according to his or her preferences.

5.1 Change Request

Following the retrieval, a change request C is defined by specifying sets of tasks or data nodes that should be added C_{add} or removed C_{del} from workflow W. The change request can be either acquired manually from the user after the workflow is presented or it can be automatically derived based on the difference between the query and the retrieved case. As the tasks and data nodes are taxonomically ordered (see Sect. 2.3), the change request can also be defined by a higher level concept of the taxonomy in order to define a more general change of the workflow. For example, a change request specified as "DELETE meat" ensures that the adapted recipe is a vegetarian dish. We define the change request fulfilment $\mathcal{F}(C, W) \rightarrow [0, 1]$ for a change request C and a workflow $W = (N, E)$

as the number of desired nodes contained plus the number of undesired nodes not contained in relation to the size of the change request:

$$\mathcal{F}(C, W) = \frac{|N \cap C_{add}| + |C_{del} \setminus N|}{|C_{add}| + |C_{del}|} \tag{3}$$

5.2 Adaptation Procedure

The goal of the adaptation procedure is to maximize the value of $\mathcal{F}(C, W)$. Therefore, it uses a kind of a hill climbing local search algorithm in order to optimize the change request fulfillment $\mathcal{F}(C, W)$. Basically, the general idea of the adaptation is that for each streamlet W_d in the retrieved workflow an applicable operator o is searched and applied to the workflow W, if its application increases the change request fulfillment. This leads to a chain of adaptation operators. During this search process, the change request, i.e., the set C_{add} is updated according to the applied operator. This ensures that nodes which were already inserted are not inserted again by subsequent operators.

Prior to the search, a partial order of data nodes is constructed w.r.t. their usage in the workflow. More precisely, data nodes are ordered with respect to which data is used first in the control-flow of the workflow, i.e., as input of a task. During adaptation this partial order is traversed starting with the data node used first in the workflow. For each streamlet W_d in the retrieved workflow W at most one applicable adaptation exchange or delete operator o is selected which maximizes $\mathcal{F}(C, o(W))$. Further, delete operators must not remove a desired node. Operators with the highest similarity between the streamlet W_d of W and the deletion streamlet o_D are preferred during selection, aiming at selecting the best possible operator. Afterwards, insert operators are applied to further improve change request fulfillment. Therefore, we select insert operators whose insertion streamlet o_I contains at least one desired but not an undesired node. Insert operators whose head node of o_I is already in W are disregarded. The adaptation process terminates, if no further insert operator can be applied which improves the change request fulfillment.

6 Evaluation

The described approach on operator-based workflow adaptation has been implemented as component of the CAKE framework [1]. To demonstrate its usefulness, the approach is experimentally evaluated to analyze whether the workflows can be improved regarding the change request fulfillment (Hypothesis H1) and to validate whether the adapted workflows are of an acceptable quality (Hypothesis H2). As only workflow operators are applied leading to a consistent workflow, the resulting workflows are consistent, which was checked and confirmed in order to validate the correctness of the implementation.

H1. The operator-based workflow adaptation considerably improves the change request fulfillment of a workflow.

H2. The operator-based workflow adaptation does not significiantly reduce the quality of workflows.

6.1 Evaluation Setup

We manually constructed 60 pasta recipe workflows from the textual recipe descriptions on www.studentrecipes.com with an average size of 25 nodes and 64 edges [23]. Altogether, they contain 162 different ingredients and 67 tasks. For ingredients and tasks, a taxonomy was manually constructed. The extracted workflows, contained AND, XOR, as well as LOOP structures. The repository of 60 workflows was split into two data sets: One repository containing 10 arbitrary workflows (referred to as query workflows) and a case base containing the remaining 50 workflows. For the learning algorithm, we set the parameter $\Delta_W = 0$ as the case base only contained pasta workflows and thus only similar recipes. Further, we chose the threshold $\Delta_D = 0.5$ in order to only create exchange operators if the corresponding head data nodes have been mapped with a similarity of at least 0.5. In total 9416 workflow adaptation operators (1504 exchange, 4460 insert, 3452 delete operators) were learned from the workflows in the case base. For each query workflow QW_i we retrieved the most similar workflow CBW_i from the case base (referred to as case base workflow) and automatically generated a change request for CBW_i by determining the set of nodes to be added and deleted in order to arrive at QW_i. A change request "DELETE prawns", for example, means that the case base workflow CBW_i uses prawns while the query workflow QW_i does not[1]. We executed the proposed operator-based adaptation method for each of the 10 workflows CBW_i using the corresponding change request. Thus, 10 adapted workflows are computed. During adaptation we chose a similarity threshold between the deletion streamlet and streamlet in the workflow Δ_S as 0.5 in order to only apply operators if at least half of the workflow elements are identical. In average, 8.9 operators (1.6 exchange, 2.8 insert, 4.5 delete operators) had been applied in order to adapt a workflow.

6.2 Experimental Evaluation and Results

To verify hypothesis H1 we computed the average change request fulfillment of the 10 adapted workflows which stands at 69,4 %. As the change request was rather large (in average 22,6 nodes) and not any combination can be represented by the automatically learned adaptation operators a change request of 100 % is not to be expected. Hence, Hypothesis H1 is confirmed.

To evaluate Hypothesis H2 a blinded experiment was performed involving 5 human experts. The experts rated the quality of the 10 case base worklows (workflow before adaptation) and the 10 corresponding adapted workflows. These 20 workflows were presented in random order, without any additional information. Thus, the experts did not know whether the workflow was an original workflow from the case base or an adapted workflow. The experts were asked to assess the quality of each workflow based on 3 rating items on a 5 point Lickert scale (from 1=very low to 5=very high). The rating items comprised the culinary quality

[1] The change request only contained ingredients and preparation steps present in the workflows from the case base and no ingredients that are used as mixtures of multiple ingredients (e.g., vegetable mix).

Table 1. Item rating assessment

	Better case base workflows	Better adapted workflows	Equal
Correctness of preparation	23	11	16
Culinary quality	21	12	17
Plausibility of preparation order	19	13	18
Aggregated quality	27	16	7

of the recipe, the correctness of the preparation (e.g. slice milk would violate the correctness), and the plausibility of the preparation order. Additionally, we computed an aggregated quality using the three rating items.

The ratings from the 5 experts of all 10 workflow pairs were compared, leading to 50 ratings. We define that one item was rated better for a workflow if it was scored with a higher value than the corresponding item of the compared workflow. Based on this, we conclude that a workflow has a higher aggregated quality, if more of its items were rated better than those of the compared workflow.

The results for each rating item in isolation and for the aggregated quality assessment are given in Table 1. It shows the number of workflows for which the case base workflow or the adapted workflow is better, as well as the number of workflows which were equally rated. In 23 out of 50 rated workflow pairs, the adapted workflow was rated of higher or equal quality (concerning the aggregated quality), whereas 27 case base workflows were rated higher. Thus, in 46 % of the assessments, the adaptation produced workflows with at least the same quality compared to the corresponding workflow from the case base. Additionally, Table 2 illustrates the average rating difference on the items of all 50 workflow pairs. In total, the items of each case base workflow are rated 1.1 higher than those of the adapted workflow, which means the single items were rated about 0.37 times better than the corresponding item of the adapted workflow. Thus, the experts rated the items and hence the quality of the case base workflows only slightly higher. This has also been proved by a paired t-test on the aggregated quality, which showed that the quality difference between the case base workflows and the adapted workflows is statistically not significant ($p = 0.19$). Altogether, Hypothesis H2 is confirmed.

Further, we asked the experts to give textual explanations in case of bad quality ratings. We identified three major reasons for quality degradations caused by the adaptation process and we sketched first ideas to overcome them.

1. It must be ensured that some components should occur only once (e.g. a sauce). This could be achieved by a few general operators that have to be defined manually.
2. After the removal of an input data object from a task the name of the output produced by the task might have to be changed (e.g. removing meat from a mixture doesn't produce a meat mixture anymore). To identify the best

Table 2. Average differences on item ratings

Correctness of preparation	0.46
Culinary quality	0.44
Plausibility of preparation order	0.20
Average per item	0.37
Average per workflow	1.1

suitable output name, outputs from similar tasks with similar input data could be employed.

3. The application of insert operators may also insert new data objects, e.g. ingredients that usually have to be processed before they are used (e.g. bolognese sauce). However, this information was not always included (e.g. a packet sauce could be used instead). To remedy this shortcoming, insert operators processing the desired data can be searched and included into the workflow.

7 Conclusions and Related Work

We presented a novel approach to operator-based adaptation of workflows, including a new representation for workflow adaptation operators, an algorithm for learning such operators, and a search-based process for applying the operators in order to address a specified workflow change request.

The major challenge of adaptation in case-based reasoning is the acquisition bottleneck of adaptation knowledge. Hence, various approaches for learning and applying of adaptation knowledge for cases represented as attribute-values have been proposed [4,8,9,14,16]. Only litte work is published that addresses this problem for more complex case representations (e.g., [15]), or for POCBR in particular. In POCBR, related work was presented by Minor et al. [18]. They propose a workflow adaptation approach which transform workflows by applying a single adaption case that can be acquired automatically [19]. In our own previous work, we introduced a compositional workflow adaptation method [22] which identifies subcomponents of a workflow, called streams. In contrast to this, the operators proposed here represent smaller subcomponents with a higher granularity. Further, the operator-based adaptation not only exchanges similar components, but inserts, deletes or exchanges different components of the workflow. Moreover, in this paper we propose to learn operators from pairs of similar workflows, while in compositional adaptation each single case is decomposed into streams. Dufour-Lussier et al. [6] presented a compositional adaptation approach for processes, requiring additional adaptation knowledge.

Our evaluation showed that the presented approach is promising for the purpose of workflow adaptation. However, the expert evaluation revealed some shortcomings on the quality of the adapted cases. Hence, we sketched ways how to overcome these which we will expore in the future. Future work will also investigate generalization [23] to improve the applicability of the learned adaptation

operators. Moreover, an extension by operators exchanging a data node with multiple data nodes and vice versa is planned. We will also investigate methods to control the retention of learned adaptation operators, as already proposed by Jalali and Leake [11] for adaptation rules. Further, we plan to perform comparisons of the different adaptation approaches we already proposed [17,22,23] and other related work (e.g., [18]). Finally, we aim at integrating them into a more comprehensive formal framework.

Acknowledgements. This work was funded by the German Research Foundation (DFG), project number BE 1373/3-1.

References

1. Bergmann, R., Gessinger, S., Görg, S., Müller, G.: The collaborative agile knowledge engine cake. In: Proceedings of the 18th International Conference on Supporting Group Work, pp. 281–284, ACM (2014)
2. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014)
3. Bergmann, R., Wilke, W.: Towards a new formal model of transformational adaptation in case-based reasoning. In: Prade, H. (ed.) 13th European Conference on Artificial Intelligence (ECAI 1998), pp. 53–57. John Wiley & Sons (1998)
4. Craw, S., Jarmulak, J., Rowe, R.: Learning and applying case-based adaptation knowledge. In: Aha, D.W., Watson, I. (eds.) ICCBR 2001. LNCS (LNAI), vol. 2080, pp. 131–145. Springer, Heidelberg (2001)
5. Davenport, T.: Process Innovation: Reengineering Work Through Information Technology. Harvard Business Review Press, Boston (2013)
6. Dufour-Lussier, V., Lieber, J., Nauer, E., Toussaint, Y.: Text adaptation using formal concept analysis. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 96–110. Springer, Heidelberg (2010)
7. Dumas, M., van der Aalst, W., ter Hofstede, A.: Process-aware Information Systems: Bridging People and Software Through Process Technology. Wiley, Hoboken (2005)
8. Fuchs, B., Lieber, J., Mille, A., Napoli, A.: Differential adaptation: an operational approach to adaptation for solving numerical problems with CBR. Knowl. Based Syst. **68**, 103–114 (2014)
9. Hanney, K., Keane, M.T.: Learning adaptation rules from a case-base. In: Smith, I.F.C., Faltings, B. (eds.) EWCBR 1996. LNCS, vol. 1168, pp. 179–192. Springer, Heidelberg (1996)
10. Hung, P., Chiu, D.: Developing workflow-based information integration (WII) with exception support in a web services environment. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences 2004, p. 10 (2004)
11. Jalali, V., Leake, D.: On retention of adaptation rules. In: Lamontagne, L., Plaza, E. (eds.) ICCBR 2014. LNCS, vol. 8765, pp. 200–214. Springer, Heidelberg (2014)
12. Kapetanakis, S., Petridis, M., Knight, B., Ma, J., Bacon, L.: A case based reasoning approach for the monitoring of business workflows. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 390–405. Springer, Heidelberg (2010)
13. Leake, D.B., Kendall-Morwick, J.: Towards case-based support for e-Science workflow generation by mining provenance. In: Althoff, K.-D., Bergmann, R., Minor,

M., Hanft, A. (eds.) ECCBR 2008. LNCS (LNAI), vol. 5239, pp. 269–283. Springer, Heidelberg (2008)
14. Li, H., Li, X., Hu, D., Hao, T., Wenyin, L., Chen, X.: Adaptation rule learning for case-based reasoning. Concurr. Comput. Pract. Exp. **21**(5), 673–689 (2009)
15. Lieber, J., Napoli, A.: Using classification in case-based planning. In: ECAI, pp. 132–136, Citeseer (1996)
16. McSherry, D.: Demand-driven discovery of adaptation knowledge. In: Dean, T. (ed.) IJCAI, pp. 222–227, Morgan Kaufmann (1999)
17. Minor, M., Bergmann, R., Görg, S.: Case-based adaptation of workflows. Inf. Syst. **40**, 142–152 (2014)
18. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 421–435. Springer, Heidelberg (2010)
19. Minor, M., Görg, S.: Acquiring adaptation cases for scientific workflows. In: Ram, A., Wiratunga, N. (eds.) ICCBR 2011. LNCS, vol. 6880, pp. 166–180. Springer, Heidelberg (2011)
20. Minor, M., Montani, S., Recio-Garcia, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014)
21. Montani, S., Leonardi, G., Lo Vetere, M.: Case retrieval and clustering for business process monitoring. In: Proceedings of the ICCBR 2011 Workshops, pp. 77–86 (2011)
22. Müller, G., Bergmann, R.: Workflow streams: a means for compositional adaptation in process-oriented CBR. In: Lamontagne, L., Plaza, E. (eds.) ICCBR 2014. LNCS, vol. 8765, pp. 315–329. Springer, Heidelberg (2014)
23. Müller, G., Bergmann, R.: Generalization of workflows in process-oriented case-based reasoning. In: 28th FLAIRS Conference, AAAI, Hollywood (Florida), USA (2015)
24. Schumacher, P., Minor, M., Walter, K., Bergmann, R.: Extraction of procedural knowledge from the web. In: Workshop Proceedings WWW 2012, Lyon, France (2012)
25. Taylor, I.J., Deelman, E., Gannon, D.B.: Workflows for e-Science. Springer, London (2007)
26. Weber, B., Wild, W., Feige, U.: CBRFlow: enabling adaptive workflow management through conversational case-based reasoning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 434–448. Springer, Heidelberg (2004)
27. Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. In: del Pobil, A.P., Mira, J., Ali, M. (eds.) IEA-1998-AIE. LNCS, vol. 1416, pp. 497–506. Springer, Heidelberg (1998)