

Workflow Streams: A Means for Compositional Adaptation in Process-Oriented CBR

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 compositional adaptation of workflows, thus addressing the adaptation step in process-oriented case-based reasoning. Unlike previous approaches to adaptation, the proposed approach does not require additional adaptation knowledge. Instead, the available case base of workflows is analyzed and each case is decomposed into meaningful subcomponents, called workflow streams. During adaptation, deficiencies in the retrieved case are incrementally compensated by replacing fragments of the retrieved case by appropriate workflow streams. An empirical evaluation in the domain of cooking workflows demonstrates the feasibility of the approach and shows that the quality of adapted cases is very close to the quality of the original cases in the case base.

Keywords: process-oriented case-based reasoning, compositional adaptation, workflows

1 Introduction

Adaptation in Case-Based Reasoning (CBR) is still a very important research field, even after more than 30 years of research in CBR. One obstacle that prevents comprehensive adaptation capabilities is the knowledge acquisition bottleneck for adaptation knowledge, which is required by many adaptation methods. Therefore, most commercial applications of CBR are developed for application domains in which adaptation is not a primary concern, such as in knowledge management or service support. However, in application domains involving complex cases with highly structured solutions, adaptation cannot be disregarded as it is an ability that could lead to significant benefits for the users. One such area is process-oriented case-based reasoning (POCBR) [11], which deals with CBR applications for process-oriented information systems (POIS). POCBR has the potential of enabling POIS to support domain experts in defining, executing, monitoring, or adapting workflows. Thus, workflow adaptation is an important field, in which still little research exist so far.

Existing methods to adaptation in CBR can be roughly classified into transformational and generative adaptation [18]. Transformational adaptation relies

on adaptation knowledge representing links between differences in the problem description and resulting modifications of the solution. For POCBR we investigated a transformational adaptation approach using adaptation cases as adaptation knowledge [9], but the acquisition of adaptation cases, although addressed in [10], is still a difficult issue. Methods for learning adaptation knowledge from a case base have been proposed by various authors [7,18,8,3], but no approaches yet exist that are appropriate for complex case representations such as used in POCBR.

Generative adaptation, on the other hand, demands general domain knowledge appropriate for an automated (knowledge-based) problem solver to solve problems from scratch. While this approach is quite successful in planning because knowledge about actions need to be present anyway, it is not appropriate for POIS as the task descriptions of workflows may describe human activities, which cannot be formalized in sufficient detail.

To circumvent the problems with these approaches, we investigate the idea of compositional adaptation for POIS. While compositional adaptation usually means that several cases are used during adaptation, still incorporating transformational or generative adaptation methods involving adaptation knowledge (such as in COMPOSER [12], *DéjàVu* [15] or PRODIGY/ANALOGY [17] – to cite some classic examples), we propose a pure compositional adaptation approach waiving of any adaptation knowledge (see [1] p.232ff and [16]). Instead, the available case base of workflows is analyzed and each case is decomposed into meaningful subcomponents, called *workflow streams*. During adaptation, deficiencies in the retrieved case are incrementally compensated by replacing fragments of the retrieved case by appropriate workflow streams. The proposed method depends on cases being represented as block-oriented workflows, as the workflow structure is exploited to define the borders of the workflow streams, ensuring their reusability. Hence, the next section introduces the workflow representation as well as certain formal notations being further used. In Section 3, the workflow streams are formally defined as a prerequisite for the compositional adaptation method described in Section 4. Finally, we report on the results of an empirical evaluation of the proposed method in the domain of cooking workflows and wrap-up by discussing related and potential future work.

2 Foundations

2.1 Workflows

A workflow is “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [19]. Broadly speaking, workflows consist 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). Tasks and control-flow structures form the *control-flow*. 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. A cooking recipe is represented as a workflow describing the instructions for cooking a particular dish [14]. 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 pizza recipe is illustrated in Figure 1 also showing the commonly used graph representation, formally defined below.

Definition 1. A workflow is a graph $W = (N, E)$ with a set of nodes N and edges $E \subseteq N \times N$. The nodes $N = D \cup T \cup C$ can be of different types, namely data nodes D , task nodes T , and control-flow nodes C . The control-flow nodes $C = C_* \cup C^*$ construct blocks of sequences and are either opening $C_* = C_A \cup C_X \cup C_L$ or closing control-flow nodes $C^* = C^A \cup C^X \cup C^L$ representing AND, XOR and LOOP nodes. The set of sequence nodes is defined as $S = T \cup C$. Edges $E = CE \cup DE$ can be control-flow edges $CE \subseteq S \times S$, which define the order of the sequence nodes or data-flow edges $DE \subseteq (D \times S) \cup (S \times D)$, which define how the data is shared between the tasks.

Figure 1 shows an opening AND control-flow node A_* and a related closing AND control-flow node A^* . Further, it is important to note that the control-flow edges CE 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.

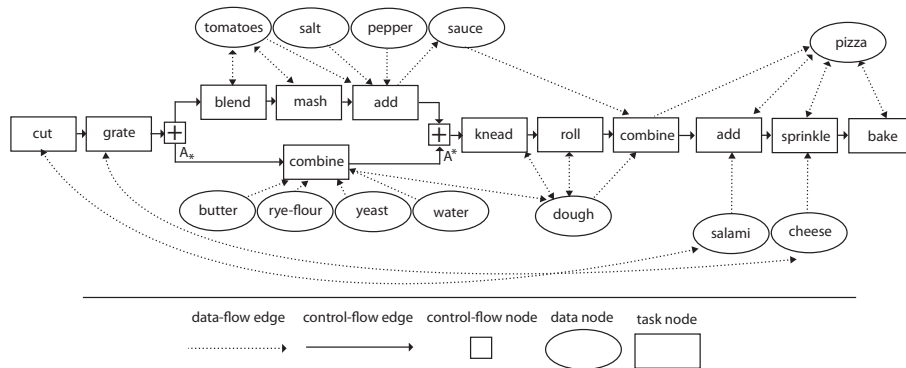


Fig. 1. Example of a block-oriented cooking workflow

2.2 Block-oriented Workflows

Block-oriented workflows are workflows that are constructed from blocks of workflow elements as defined in Definition 2.

Definition 2. A block-oriented workflow $W = (N, E)$ is a workflow with a block-oriented graph structure according to the following rules:

- a) A block element is a workflow subgraph of the Graph $(N \setminus D, CE)$ which contains either:
 - a task node
 - a sequence of block elements
 - a LOOP block containing an opening loop node, 1 block element, and a closing loop node
 - a XOR/AND block containing an opening XOR/AND node, 2 branches containing a block element, and a matching closing XOR/AND node¹
 - a XOR block containing an opening XOR node, 1 branch with a block element, an empty branch and a closing XOR node
- b) Each block element must either contain one task or another block element
- c) The workflow W is a single block

The workflow shown in Figure 1 is a block-oriented workflow. Figure 2 illustrates the workflow blocks, which are marked with dashed rectangles. Workflow block elements may be nested (e.g. see loop block), but not interleaved. The construction of block-oriented workflows restricts the usage of control-flow edges w.r.t. block elements by the correctness-by-construction principle [13,4]. Such a construction ensures the syntactic correctness of the workflow, e.g., that the workflow has one start node (node without ingoing control-flow edges) and one end node (node without outgoing control-flow edges). Such workflows are referred to as consistent workflows. In contrast to the control-flow, the data-flow is not restricted by the block-oriented workflow construction.

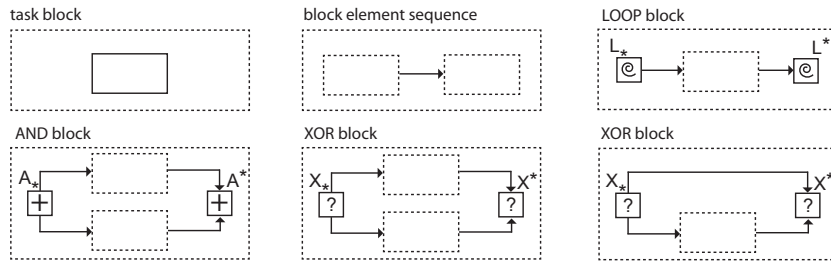


Fig. 2. Illustration of workflow block elements

2.3 Partial Workflows

As we aim at reusing meaningful parts of workflows, we define the notion of a partial workflow as follows.

¹ For the sake of simplicity only 2 branches are contained in an XOR/AND block, as multiple branches can be easily be constructed by nesting other XOR/AND blocks.

Definition 3. For a subset of tasks $T' \subseteq T$, a partial workflow W' of a block-oriented workflow $W = (N, E)$ is a block-oriented workflow $W' = (N', E' \cup CE'_+)$, with a subset of nodes $N' = T' \cup C' \cup D' \subseteq N$. $D' \subseteq D$ is defined as the set of data nodes that are linked to any task in T' , i.e., $D' = \{d \in D \mid \exists t \in T' : ((d, t) \in DE \vee (t, d) \in DE)\}$. W' contains a subset of edges $E' = E \cap (N' \times N')$ connecting two nodes of N' supplemented by a set CE'_+ of additional control-flow edges that retain the execution order of the sequence nodes, i.e., $CE'_+ = \{(n_1, n_2) \in S' \times S' \mid n_1 < n_2 \wedge \nexists n \in S' : ((n_1, n) \in CE' \vee (n, n_2) \in CE' \vee n \in [n_1, n_2])\}$

In general, control-flow nodes are part of a partial workflow if they construct a workflow w.r.t. the block-oriented workflow structure. This basically means that each block nested in a control-flow block element must either contain one task or some child block element containing a task. The additional edges CE'_+ 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 $s_2 \notin N'$. Figure 3 illustrates a partial workflow W' of the workflow W given in Figure 1. One additional edge is required in this example, depicted by the double-line arrow since “grate” and “add” are not linked in W .

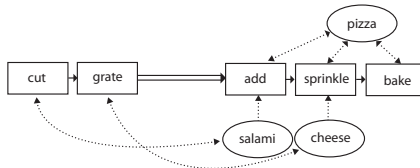


Fig. 3. Example of a partial workflow W'

2.4 Semantic Workflows and 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 similarity of task or data items reflects the closeness in the taxonomy, and further regards the level of the taxonomic elements. In particular, if a more general query element such as “meat” is compared with a specific element below it, such as “pork”, the similarity value is 1. This ensures that if the query asks for a recipe containing meat, any recipe workflow from the case base containing any kind of meat are considered highly similar.

The similarity measure performs a kind of inexact subgraph matching, optimizing the overall similarity between the matched workflow elements. It is used

during case retrieval in order to find workflows which best match a certain query. Within the adaptation method described in this paper, the same similarity based retrieval method is used to identify reusable workflow streams, as we will show in Section 4.

3 Workflow Streams

According to Davenport, “[...] a process is simply a structured, measured set of activities designed to produce a specific output for a particular customer on the market” [5]. We define this specific output as the goal of a workflow. This goal is reached by producing partial outputs and combining them to the specific output. Thus, a workflow that, for example, prepares a pizza dish, also prepares the dough, the pizza sauce, and the toppings. Hence, these partial outputs represent partial goals of a workflow. While the entire workflow is designed to fulfill the overall goal of a workflow, particular parts of the workflow attain partial goals (see Fig. 4). Tasks in a workflow that fulfill a partial goal, produce a data node that is not consumed by the same task, i.e., task having at least one outgoing data edge but no ingoing data edge referring to the same data node d . We call such tasks *creator tasks*. For a workflow W the set of creator tasks is defined as follows:

$$CT = \{t \in T \mid \exists d \in D : ((t, d) \in DE \wedge (d, t) \notin DE)\} \quad (1)$$

The creator tasks of the workflow illustrated in Figure 4 are marked with \odot symbols. They create the dough, the sauce or combine the dough, and sauce to create the pizza.

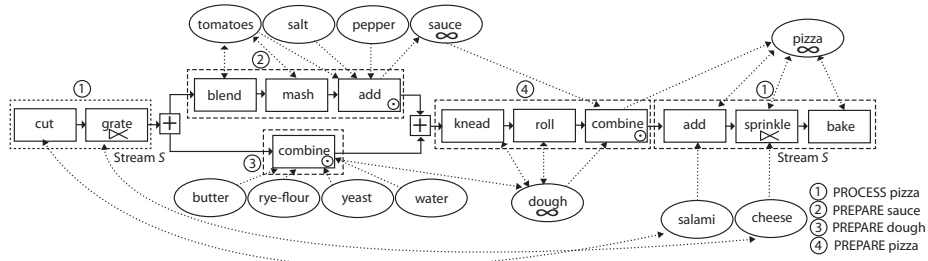


Fig. 4. Workflow and workflow streams

Furthermore, not only the creator task is sufficient to produce new data. For example, the preparation steps to blend and mash the tomatoes have to be completed before the pizza sauce can be produced. Thus, all tasks that are required, before the creator task can be completed have to be identified. Such tasks can be recognized by aid of the data-flow. We define that two tasks $t_1, t_2 \in T$ are data-flow connected $t_1 \times t_2$ if $t_1 < t_2$ and t_2 consumes a data node produced

by t_1 . Thus, t_1 has to be completed before task t_2 can be executed as otherwise the data wouldn't be processed in the correct order. Hence, if the data-flow connectedness is regarded and retained when modifying the workflow, the semantic correctness of the workflow is ensured. We define:

$$t_1 \times t_2, \text{ iff } t_1 < t_2 \wedge \exists d \in D : ((t_1, d) \in DE \wedge (d, t_2) \in DE) \quad (2)$$

Further, let $t_1 \overset{d}{\times} t_2$ denote that two tasks $t_1, t_2 \in T$ are data-flow connected via the data object d . Figure 4 shows that the tasks “grate cheese” and “sprinkle cheese over the pizza” (marked with \times) are data-flow connected via the data node “cheese”. Additionally, we define $t_1 \times^* t_2$ to express that two tasks $t_1, t_2 \in T$ are transitively data-flow connected:

$$t_1 \times^* t_2, \text{ iff } t_1 \times t_2 \vee \exists t \in T : (t_1 \times t \wedge t \times^* t_2) \quad (3)$$

We use the definition of creator tasks and data-flow connectedness to decompose a workflow W into partial workflows each of which represent disjoint partial goals. Each workflow W can be partitioned by the definition given below.

Definition 4. A partition $\mathcal{S}^W = \{\mathcal{S}_1^W, \dots, \mathcal{S}_n^W\}$ of a block-oriented workflow W is a set of partial workflows \mathcal{S}_i^W , such that each task $t \in T$ is contained in a partial workflow \mathcal{S}_i^W and such that the tasks in each \mathcal{S}_i^W are transitively data-flow connected and not contained in any other partial workflow $\mathcal{S}_{j \neq i}^W$. All creator tasks $y \in CT$ are end nodes of any partial workflow in \mathcal{S}^W . Each partition \mathcal{S}_i^W is called workflow stream.

In order to partition the workflow W into streams, for each creator task $y \in CT$ a set of transitively data-flow connected and disjoint tasks $T_{\mathcal{S}}$ is identified. $T_{\mathcal{S}}$ is constructed, by adding all data-flow connected tasks that are not data-flow connected to any predecessor creator task of CT (as the creator tasks represent end nodes of partial workflows):

$$T_{\mathcal{S}}(y) := \{t \in T \mid t \times^* y \wedge t \notin CT \wedge \nexists x \in CT : (t \times^* x \wedge y \not\prec x)\} \cup \{y\} \quad (4)$$

Based on the set of tasks $T_{\mathcal{S}}$ that belong to a workflow stream \mathcal{S} , a partial workflow can be constructed according to Definition 3, which is then finally referred to as workflow stream. For each of the creator tasks illustrated in Figure 4 (marked with \odot) the dashed lines represent the tasks assigned to the referring stream (see stream 2,3,4). Regarding task disjointness, the task “mash tomatoes”, for example, does not belong to the workflow stream 4, although it is transitively data-flow connected, as it is separated though a data-flow connected creator task (see “combine” task of stream 4).

According to Definition 4, each task is contained in a partial workflow \mathcal{S}_i^W . Thus, each set of transitively data-flow connected tasks not already contained in any stream \mathcal{S}_i^W derived from a creator task, is assigned to a new stream. Thus, not only streams that produce new data nodes are regarded, but also streams processing a data node. As an example, see stream 1 in Fig. 4 for putting the toppings on the pizza.

To summarize, the workflow is partitioned in such a manner that each task is assigned to exactly one stream of the workflow. The extracted workflow streams are themselves consistent workflows as they are block-oriented. Furthermore, due to the data-flow connectedness, the streams maintain their “semantic correctness” as they represent meaningful connected subcomponents of the original workflow.

The basic idea for compositional adaptation is, to adapt a workflow by using the workflow streams of other workflows that fulfill the same partial goal in a different manner, e.g., with other tasks or data. In the pizza domain, for example, toppings or preparation steps, can be replaced. Therefore it is required to identify whether some streams can be substituted. We require that substitutable streams must produce the same data and that they must consume the identical data nodes. This ensures that replacing an arbitrary stream doesn’t violate the semantic correctness of the workflow, e.g. that a data object is never produced or consumed (except of those never produced or consumed by the entire workflow). The following notion of *anchors* is used to define the relevant data notes to be considered to decide whether two streams are substitutable.

Definition 5. For a stream \mathcal{S} , a set of anchors is defined as $A_{\mathcal{S}} = \{d \in D_{\mathcal{S}} | (\exists t \in T \setminus T_{\mathcal{S}} \wedge \exists t_{\mathcal{S}} \in T_{\mathcal{S}}) : (t \stackrel{d}{\times} t_{\mathcal{S}} \vee t_{\mathcal{S}} \stackrel{d}{\times} t)\}$

Thus, anchors of a stream \mathcal{S}_i^W are data nodes of this stream that are either produced or consumed by a task of another stream $\mathcal{S}_{j \neq i}^W$ of the same workflow W . Two streams are substitutable if they have identical anchor sets. The anchor nodes of all streams are marked with ∞ symbols in Figure 4, i.e., sauce, dough, and pizza.

4 Compositional Adaptation using Workflow Streams

In the following, we assume that each workflow in the case base has been decomposed into workflow streams according to Section 3. These streams are linked to each workflow in the case base and stored in a separate workflow stream repository. The adaptation procedure is now presented and explained by an example scenario. After the retrieval of a most similar workflow W the user might want to adapt this workflow to his or her preferences. Let us assume that the user retrieved the workflow W given in Figure 4.

4.1 Change Request

Following the retrieval, a change request is defined by specifying lists of task or data nodes that should be added (ADD list) or removed (DELETE list) from workflow W . The change request can be either manually acquired from the user after the workflow is presented or it can be automatically derived based on the difference between the query and the retrieval case. As the tasks and data are taxonomically ordered (see Section 2.4), 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.

4.2 Search of Substitute Stream Candidates

To perform a workflow adaptation, for each stream \mathcal{S} in the retrieved workflow W a substitute stream candidate \mathcal{S}' is searched in the constructed stream pool repository. If a candidate can be found, the stream \mathcal{S} is substituted with stream \mathcal{S}' . The identification of a substitute stream candidate \mathcal{S}' requires to identify those workflows that can be replaced with a stream \mathcal{S} . As already mentioned, stream \mathcal{S} and \mathcal{S}' must have the identical set of anchor elements. This condition ensures that the adapted workflow is semantically correct, i.e., that the workflow doesn’t contain any data nodes that are never produced or consumed by an other workflow stream. Additionally, the substitute stream candidate must regard the change request. Hence, streams are searched that do not contain any node given in the DELETE list and that contain the highest number of nodes from the ADD list of the change request. For all streams that fulfill these conditions, a retrieval of the most similar stream to stream \mathcal{S} is executed, considering the semantic similarity measure sketched in Section 2.4. The retrieved most similar stream is the substitute stream candidate. This approach ensures that a substitute stream is selected that only changes the workflow as much as required w.r.t. the change request.

The following example illustrates this approach. Assume that for a certain query, the workflow from Fig. 4 is retrieved and that the change request is to “DELETE salami” and to “ADD ground beef”. Lets assume that stream 1 of Fig. 4 is stream \mathcal{S} and the stream illustrated in Figure 5 is the substitute stream candidate \mathcal{S}' . Both streams have an identical set of anchor elements (see ∞) which only contains the “pizza” node. Furthermore, workflow stream \mathcal{S}' regards the change request as it contains the node given in the ADD list (“ground beef”) and no node defined in the DELETE list (“salami”).

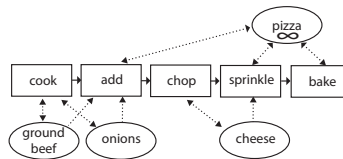


Fig. 5. Substitute stream \mathcal{S}'

4.3 Replacing Workflow Streams

To replace stream \mathcal{S} by the substitute stream candidate \mathcal{S}' , \mathcal{S} is first removed from the workflow. This means that a partial workflow (see Sec. 2.3) is con-

structured, containing all tasks of W except of those contained in S . The removal of the workflow stream in the given scenario is illustrated in Figure 6.

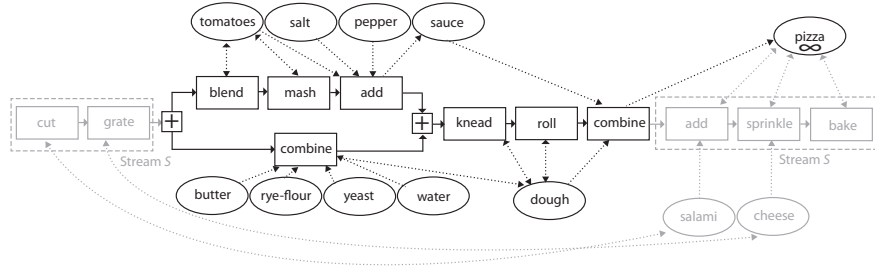


Fig. 6. Stream S removed from W

Then, the new stream S' is inserted at the position of the last sequence node of the workflow stream S in W . This means that all edges, tasks, and data nodes (if not already present) of S' are inserted into the workflow W . Then, the inserted stream S' is connected with an additional control-flow edge that links the tasks of the stream at the old position to the last sequence node of the removed stream in W . In the illustrated scenario, the stream S' is inserted behind the last “combine” task (see Fig. 7). In the special case that the last sequence node of S is a control-flow node that is still a part of the partial workflow after removing workflow stream S , the stream S' is inserted behind this control-flow node.

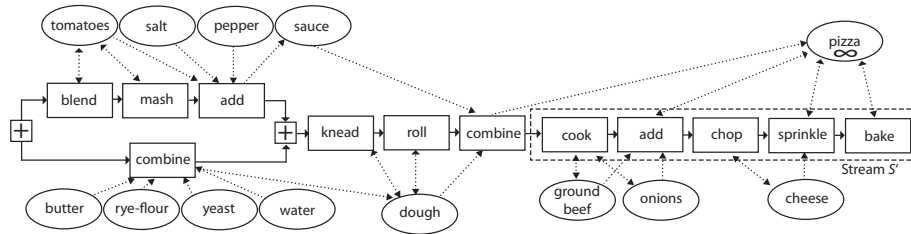


Fig. 7. Stream S' added to W

Inserting the stream S' may lead to a shift of tasks. In the given scenario, for example, the start node of the original workflow and the adapted workflow differ (compare Fig. 4 and Fig. 7). However, this does not violate the semantic correctness of the workflow, i.e., the data is still being processed in the right order. This is ensured by the workflow stream definition as all tasks that are transitively data-flow connected are included in the workflow stream as long as

they are not separated by a creator task. Additionally, the anchors ensure that only streams are replaced representing an identical partial goal.

5 Evaluation

The described approach to compositional adaptation of workflows has been fully implemented as part of the CAKE framework² that already includes a process-oriented case-based reasoning component for similarity-based retrieval of workflows. To demonstrate its usefulness, the approach is experimentally evaluated focussing on two hypotheses explained below.

In Section 3 it was already stated that workflows adapted by the workflow stream method are consistent and semantically correct. Hence, this property will not be subject to the empirical evaluation. However in the experiments the consistency and semantic correctness of the constructed workflows was checked (and confirmed) in order to validate the correctness of the implementation. We conducted an empirical evaluation to analyze whether change requests are fulfilled by the adapted workflows (Hypothesis H1) and to validate whether the adapted workflows are of an acceptable quality (Hypothesis H2).

- H1.** The compositional adaptation approach is able to produce workflows that fulfill the change requests.
- H2.** The compositional adaptation produces workflows, whose quality is comparable to the quality of the workflows in the case base.

5.1 Experimental Setup

We developed an automatic workflow generator that enables us to produce workflow repositories for experimental purposes with predefined properties. The generator uses the fast downward planner³ to produce sequences of tasks, which are further organized into a (partly parallelized) workflow structure. The generated workflows are block-oriented and semantically correct w.r.t. the definition of the planning domain. For the experiments, we generated 240 different cooking workflows, each of which produces a pizza. Overall, 17 different ingredients and 8 different tasks occur in the workflows. Due to the limitation of the workflow generator, the workflows do not contain any XOR or LOOP structures, but AND blocks occur frequently.

The experimental setup to validate the hypotheses defined above is illustrated in Figure 8. The repository of 240 workflows was split into two data sets: One repository containing 10 arbitrary workflows (referred to as test workflows) and a case base containing the remaining 230 workflows. A stream repository was computed for adaptation based on the workflows contained in the case base. In

² cake.wi2.uni-trier.de

³ www.fast-downward.org

total 920 workflow streams were identified and stored in the stream repository. For each test workflow TW_i we retrieved a random workflow from the case base (referred to as retrieved workflow) and constructed a pair of workflows (TW_i, RW_i) . If the retrieved workflow RW_i was already paired with another test workflow $TW_{i \neq j}$ or if the set of data and task nodes was identical, a different workflow RW_i was selected from the case base. Each pair (TW_i, RW_i) was used to automatically generate a change request for RW_i by determining the set of nodes to be added and deleted in order to arrive at TW_i . A change request “DELETE salami”, for example, means that the retrieved workflow RW_i uses a salami topping while the test workflow TW_i does not. We executed the proposed compositional adaptation method for each of the 10 workflows RW_i using the corresponding change request. Thus, 10 adapted workflows are computed.

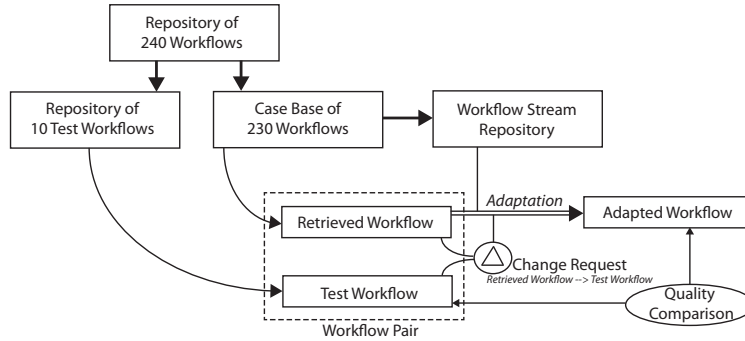


Fig. 8. Experimental Setup

5.2 Experimental Evaluation and Results

To verify hypothesis H1 we analyzed the resulting adapted workflows whether they fulfilled the change request, i.e., whether they did not contain any node from the DELETE list but all nodes from the ADD list. It turned out that each of the 10 adapted workflows fulfilled their change request, thus Hypothesis H1 was confirmed. This outcome was not very surprising, as the workflows in the case base all represent pizza recipes and are thus relatively similar to each other. Due to the large number of 920 workflow streams generated, there was a high chance that streams could be found that perfectly matches the change request.

To evaluate Hypothesis H2 a blinded experiment was performed involving 5 human experts. The experts rated the quality of the 10 test workflows 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

5 rating items on a 5 point Lickert scale (from 1=very low to 5=very high). The rating items are comprised of the intelligibility of the entire preparation process, the correctness of the recipe (w.r.t. order of tasks and data-flow), the level of process optimization (w.r.t. parallel execution and task order), the plausibility of the preparation steps, and the overall quality of the recipe.

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.

Table 1 illustrates the results for each rating item in isolation as well as for the aggregated quality assessment. It shows the number of workflows for which the test workflow or the adapted workflow is better, as well as the number of workflows which were equally rated. In 22 out of 50 rated workflow pairs, the adapted workflow was rated of higher or equal quality (concerning the aggregated quality), whereas 28 test workflows were rated higher. Thus in 44% of the assessments, the adaptation produced workflows with at least the same quality as the workflow from the case base from which they were assembled. When only the rating of the overall quality is regarded, even 66% of the assessments indicate that the adaptation produces a workflow with at least the same quality as the 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 test workflow are rated 1.34 higher than those of the adapted workflow, which means that each item was rated about 0.27 times better than the corresponding item of the adapted workflow. Thus, the experts rated the items and hence the quality of the test workflows only slightly higher. Altogether, Hypothesis H2 is mostly confirmed.

Table 1. item rating assessment

	better test workflows	better adapted workflows	equal
intelligibility of recipe	24	10	16
correctness of recipe	14	7	29
level of process optimization	20	15	15
plausibility of preparation steps	19	11	20
overall quality	17	12	21
aggregated quality	28	15	7

Table 2. average differences on item ratings

intelligibility of recipe	0.26
correctness of recipe	0.42
level of process optimization	0.20
plausibility of preparation steps	0.18
overall quality	0.28
average per item	0.27
average per workflow	1.34

6 Conclusions and Related Work

Adaptation is a major challenge in case-based reasoning. However, most research only considers cases represented by attribute-values [7,3,8] and there is only little work addressing workflow adaptation. The presented approach of Minor et al. [9], for example, executes workflow adaptations by transforming workflows in a kind of a rule-based manner. Dufour-Lussier et al. [6] presented a compositional adaptation approach for processes, which differs as in their work processes are represented as trees and additional adaptation knowledge is required.

We presented a novel approach to compositional adaptation of workflows by decomposing them into reusable workflow parts (workflow streams). We investigated how to identify and to replace workflow streams and developed an approach to adapt entire workflows regarding a change request. A major advantage is that no manually acquired adaptation knowledge is needed, thus expensive knowledge acquisition is avoided. Instead, the available knowledge contained in the case base is accessed. The proposed approach ensures that the adapted workflows are consistent and semantically correct. Our evaluation indicates that the adaptation process does not significantly decrease the quality of the resulting workflows. However, the presented approach is limited, as it requires similarly structured workflows and workflow streams to be present. Further, we employ the domain knowledge in the ontology during the similarity assessment and for the representation of change requests. Additional adaptation knowledge is yet not used, but could be considered in future extension of the proposed approach.

Future work will extend the existing evaluation towards other domains and varying characteristics of the case base. The developed workflow generator now makes such evaluations feasible. Furthermore, we will explore various ways to improve the proposed method by weakening the equality condition on anchors in the substitutability of streams, and by generalizing the streams by using the taxonomy of tasks and data items. To further increase the flexibility of workflow reuse, workflow streams might be a means to construct abstract workflows, to identify subworkflows, or to optimize workflows (w.r.t. parallel executions).

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

References

1. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNAI, vol. 2432. Springer (2002)
2. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. *Inf. Syst.* 40, 115–127 (Mar 2014)
3. Craw, S., Jarmulak, J., Rowe, R.: Learning and applying case-based adaptation knowledge. In: Aha, D., Watson, I. (eds.) *Case-Based Reasoning Research and Development*, LNCS, vol. 2080, pp. 131–145. Springer (2001)
4. Dadam, P., Reichert, M., Rinderle-Ma, S., Göser, K., Kreher, U., Jurisch, M.: Von ADEPT zur AristaFlow BPM Suite-Eine Vision wird Realität: "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen. University of Ulm (2009), <http://dbis.eprints.uni-ulm.de/489/>
5. Davenport, T.: *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business Review Press (2013)
6. Dufour-Lussier, V., Lieber, J., Nauer, E., Toussaint, Y.: Text adaptation using formal concept analysis. In: Bichindaritz, I., Montani, S. (eds.) *Case-Based Reasoning. Research and Development*, LNCS, vol. 6176, pp. 96–110. Springer (2010)
7. Hanney, K., Keane, M.T.: Learning adaptation rules from a case-base. In: Smith, I.F.C., Faltings, B. (eds.) *EWCBR*. LNCS, vol. 1168, pp. 179–192. Springer (1996)
8. McSherry, D.: Demand-driven discovery of adaptation knowledge. In: Dean, T. (ed.) *IJCAI*. pp. 222–227. Morgan Kaufmann (1999)
9. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: *Case-Based Reasoning. Research and Development*, pp. 421–435. Springer (2010)
10. Minor, M., Görg, S.: Acquiring adaptation cases for scientific workflows. In: *Case-Based Reasoning. Research and Development, ICCBR 2011*. LNCS, vol. 6880, pp. 166–180. Springer (2011)
11. Minor, M., Montani, S., Recio-Garca, J.A.: Process-oriented case-based reasoning. *Information Systems* 40(0), 103 – 105 (2014)
12. Purvis, L., Pu, P.: Adaptation using constraint satisfaction techniques. In: Veloso, M.M., Aamodt, A. (eds.) *Case-Based Reasoning Research and Development, IC-CBR'95, Sesimbra, Portugal, Proc.* LNCS, vol. 1010, pp. 289–300. Springer (1995)
13. Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. University of Ulm (2000), <http://dbis.eprints.uni-ulm.de/433/>
14. Schumacher, P., Minor, M., Walter, K., Bergmann, R.: Extraction of procedural knowledge from the web. In: *Workshop Proc.: WWW'12*. Lyon, France (2012)
15. Smyth, B., Cunningham, P.: Deja vu: A hierarchical case-based reasoning system for software design. In: Neumann, B. (ed.) *Proc. ECAI'92*. pp. 587–589 (1992)
16. Stahl, A., Bergmann, R.: Applying recursive CBR for the customization of structured products in an electronic shop. In: Blanzieri, E., Portinale, L. (eds.) *Advances in Case-Based Reasoning, 5th European Workshop, EWCBR'00, Trento, Italy, Proc.* LNCS, vol. 1898, pp. 297–308. Springer (2000)
17. Veloso, M.M.: *Planning and Learning by Analogical Reasoning*, LNCS, vol. 886. Springer (1994)
18. Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. In: *Tasks and Methods in Applied Artificial Intelligence*, pp. 497–506. Springer (1998)
19. Workflow Management Coalition: *Workflow management coalition glossary & terminology*. http://www.wfmc.org/docs/TC-1011_term_glossary_v3.pdf (1999), last access on 04-04-2014