Case-based adaptation of workflows

Mirjam Minor a,*, Ralph Bergmann b, Sebastian Görg b

a Goethe University Frankfurt, Department of Business Information Systems, D-60325 Frankfurt, Germany
b University of Trier, Department of Business Information Systems, D-54286 Trier, Germany

A B S T R A C T

This paper presents on a Case-based Reasoning approach for automated workflow adaptation by reuse of experience. Agile workflow technology allows structural adaptations of workflow instances at build time or at run time. The approach supports the expert in performing such adaptations by an automated method. The method employs workflow adaptation cases that record adaptation episodes from the past. The recorded changes can be automatically transferred to a new workflow that is in a similar situation of change. First, the notion of workflow adaptation cases is introduced. The sample workflow modeling language CFCN is presented, which has been developed by the University of Trier as a part of the agile workflow management system Cake. Then, the retrieval of adaptation cases is briefly discussed. The case-based adaptation method is explained including the so-called anchor mapping algorithm which identifies the parts of the target workflow where to apply the changes. A formative evaluation in two application domains compares different variants of the anchor mapping algorithm by means of experts assessing the results of the automated adaptation.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Workflows are a well-established concept to formalize business processes and support their execution by a workflow management system. Workflows are “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” [41]. While early workflow management systems put their focus mainly on production workflows, additional application fields like financial services, telecommunications, the public sector and even creative areas like media industry or nutrition services are addressed by workflow technology today [32]. With the changing requirements in this wide range of areas, we identified a research gap for more flexibility support of workflows at run time as described in the following.

In traditional workflow management systems, workflow definitions (also called workflow templates or process models) are created at build time. The basic constituents of workflow definitions are tasks that describe an activity to be performed by an automated service (e.g. within a service-oriented architecture) or a human (e.g. an employee). The procedural rules to control the tasks are usually described by routing constructs like sequences, loops, parallel and alternative branches that form the control flow of the tasks. Documents and information are represented by data objects; the relationships between data objects and tasks form the data flow. Workflow instances (in short workflows) are derived from the workflow definitions to be enacted, i.e., to be scheduled for execution within a workflow engine of the workflow management system.

A significant limitation of traditional workflow technology is the missing flexibility of the workflow concept and the lack of flexibility of workflow engines. Once
described, workflow definitions are instantiated repeatedly and the instances are executed in the same manner over a long period of time. However, changes of ongoing work are quite common in areas like media industry or telecommunication. In order to address today's requirements concerning easy and fast adaptation of workflows, a new class of workflow systems has been emerging since the late Nineties: Adaptive workflow systems (also called agile workflow systems) [30,37,36,25,31] facilitate structural changes of workflows at run time. Workflow instances can be created (for example based on a workflow definition from a repository [9]) and tailored for a particular demand or business case. Workflow instances can still be adapted after they have been started, for example if some unforeseen events occur [39]. Currently, adaptive workflow technology has reached such a degree of maturity that the adaptive workflow systems are about to be transferred into practice. Hence, the creation and adaptation of workflows has now become an important activity to enable the flexibility of business processes. Though, in real-world scenarios both are nontrivial modelling tasks for which intelligent support is needed.

We put forward the hypothesis that supporting the users "by sample", i.e., by retrieving and automatically transferring past modifications, alleviates the creation and adaptation of workflow instances during their entire life-cycle. Case-based Reasoning (CBR) [1] is a problem-solving technology that facilitates the reuse of experience in form of cases. A case records a problem situation together with the experiences gained during a problem-solving episode, which includes a solution. CBR provides techniques for representing, storing, indexing, retrieving, and adapting cases. Instead of composing new solutions from scratch, retrieved cases or portions of cases can be reused. We assume that workflow modeling experts who are faced with the problem of how to adapt a workflow instance benefit from solutions and experiences from similar problem situations in the past. The paper addresses a case-based approach supporting the adaptation of workflow instances by automated workflow retrieval and adaptation at build time and run time. This is a contribution towards more flexibility support in workflow management.

The paper is organized as follows. Section 2 gives a brief introduction into agile workflow technology. In Section 3, we present a representation form for adaptation knowledge called adaptation cases. Section 4 is on an automated method for workflow adaptation. In Section 5, we provide evaluation results. Section 6 contains a discussion of related work. Finally, we draw a conclusion in Section 7.

2. Agile workflow technology

Agile workflow technology is the technical prerequisite for adaptation support at run time. We will briefly introduce its core concepts before we deal with the actual adaptation methods. Agile workflow technology addresses structural changes of workflow instances at run time. The changes apply to workflow elements, i.e., to atomic parts of the workflow. The types of workflow elements depend on the workflow modeling language. In case of Petri Nets [33], for instance, the types would be places, transitions, and connections while in BPMN [13] they might be activities, gateways, events, connecting objects, and artifacts. The workflow adaptation is carried out by deleting, modifying, or adding one or several workflow elements, or by changing the order of elements.

Two types of structural changes at run time are considered in the approach (for a more detailed discussion of change patterns see [39]): Ad hoc changes occur unexpectedly at any time. The workflow elements that will be affected by the change are not known in advance. For instance, a device might be disabled accidentally during a software update process. The workflow supporting the software update process might be adapted by inserting a task for the re-installation of the driver. In contrast to ad hoc changes, late modeling refers to structural changes that are projectable to some extent. It is known when the time for the modification will come, but it is not fully foreseeable which workflow elements will be affected. For example, a customer might require to test several variants of a product component before making the decision which variant should actually go into the production process. As a consequence of the decision, the product development process might require changes like further tests or modified integration activities.

We use the Cake Flow Cloud Notation (CFCN, compare [28]) as a sample modeling language to illustrate the work. It has been developed in recent research project at the University of Trier [28,11]. It is a part of the Collaborative Agent-based Knowledge Engine (Cake) [4,26]. Cake provides modeling and enactment support for agile workflows. CFCN consists of several types of workflow elements. A CFCN workflow element can be a task, a data object, a data link (from a data object to a task or vice versa), a start element, an end element, a breakpoint, or a control flow element like an AND-split, AND-join, XOR-split, XOR-join, LOOP-split, or LOOP-join. The control flow elements border block-oriented control flow structures (AND-block, XOR-block, etc.). Blocks cannot be interleaved but they can be nested. For example, the software update workflow depicted in Fig. 1 (I) has the following workflow elements: A start element, an AND-split and AND-join bordering two tasks “Evaluate updates” and “Plan update release”, a further task “Deploy updates”, and an end element. In part (II) of Fig. 1, an ad hoc change is started by setting a breakpoint. The breakpoint suspends the subsequent workflow elements from execution, so that only the lower branch of the AND-block with the task “Plan update release” can continue execution, while the upper branch is under construction. In part (III), an additional task “Re-install driver” has been inserted. Finally, in part (IV), the breakpoint has been deleted again and the upper branch of the AND-block can continue execution. Each workflow element has a unique identifier (ID) and – except for data objects and data links – an execution state. For instance, the execution state “ACTIVE” is assigned to elements that have already started.

\footnote{See \url{http://www.aristaflow.com} for a commercial sample of an adaptive workflow system and \url{http://www.uni-trier.de/index.php?id=21075} for a research prototype.}
execution. "READY" means that a workflow element has been enabled for execution, and "COMPLETED" denotes the status of workflow elements that have already finished execution. For details on how the Cake workflow management system executes the workflow instances including the handling of breakpoints and how to deal with changes within LOOP-blocks, we refer to our previous work [25].

3. Workflow adaptation cases

The adaptation of workflow instances is a difficult modeling task. We employ Case-based Reasoning to support the user in adapting workflow instances at build time or performing ad hoc changes and late modeling at run time. Traditionally, adaptation knowledge in CBR is represented by adaptation rules [8]. Our approach differs as it is based on adaptation episodes recorded in adaptation cases [19]. Broadly speaking, adaptation rules provide general executable instructions, while adaptation cases describe situation-specific adaptation traces, which can be transferred to another, similar situation and replayed there.

A workflow adaptation case can be described by a problem part and a solution part:

1. The problem part consists of
   (a) a semantic description of the change request (what causes the change)
   (b) the original workflow instance prior to the adaptation.
2. The solution part contains
   (a) the adapted workflow
   (b) the description of the adaptation steps (added and deleted workflow elements) that have been executed to transform the original workflow version into the adapted version.

The change request 1(a) is a semantic description of the cause of the request. This part of the problem description makes use of traditional case representation approaches [5], e.g. a structural representation by metadata or a textual representation. The workflows in 1(b) and 2(a) are represented by a directed graph. Most of the present workflow languages are graph-based. They can be transformed into a directed graph as they consist of workflow elements as atomic parts organized in a flow-oriented manner, i.e., the elements can be represented by nodes and the edges represent the flow of tasks and data. A sample of such a graph-based workflow language is CFCN that we described before. The representation of the adaptation steps 2(b) deserves some special attention. The adaptation steps are described by an add and a delete list. Each list contains a set of chains of workflow elements. A chain encapsulates a connected sub-graph of workflow elements that are to be added or deleted ‘in a chain’, i.e., the chain is either fully applied or not at all while reusing the adaptation case.
Furthermore, each chain records a pair of anchor sets that describe the positions within the workflow where the edit operations have taken place. The pre-anchors are the workflow elements (in the original workflow) directly preceding the workflow elements from the chain that have been added or deleted. The post-anchors are the workflow elements from the original workflow following the last elements of the chain. Hence, the set of anchors describes the connector nodes at which a sub-graph has been inserted or pruned out. These anchors are further used during the reuse of the workflow adaptation to identify similar points in new workflows at which the proposed adaptation can be applied.

Fig. 2 illustrates a sample workflow adaptation case. The considered workflow describes the software update process for Windows 7 including service pack 1 (SP1). During the evaluation of the future updates, a CD-Rom drive is not working any more as described in the change request in Fig. 2 (1(a)). The original workflow in Fig. 2 (1(b)) in UML activity diagram notation is the same as the sample workflow depicted in Fig. 1 (I). The adapted workflow in Fig. 2 (2(a)) is quite similar to Fig. 1 (IV). There is only one chain required to transform 1(b) into 2(a), namely the add chain with two tasks an anchor pair depicted in Fig. 2 (2(b)) ADD list. The pre-anchor is the task “Evaluate updates” with ID 165. The post-anchor is the AND-join with ID 167. For simplicity reasons, the sample case does not involve data objects. A sample adaptation case including data objects can be found in the literature [27].

Note that the overall description of the adaptation chains can be automatically computed from the original workflow 1(b) and the adapted workflow 2(a) (compare [24]) or alternatively it can be captured as part of a modeling tool that is used for manual workflow adaptation. The structural correctness of the adapted workflow in the case has to be maintained by the modeling tool or algorithm applied for its creation following the principle of correctness-by-construction [20]. For instance, a modeling tool for a block oriented language like CFCN guarantees that AND-splits cannot be deleted without deleting the according AND-join (control flow correctness). The data consistency [34] can be maintained by similar mechanisms preserving the correctness of read and write operations, e.g., by a versioning of data objects.

4. Workflow adaptation

The case-based workflow adaptation approach requires a case base, i.e. a repository of adaptation cases. When a new problem situation occurs, i.e., a user aims at changing the workflow, a similarity-based retrieval mechanism is performed to determine the best matching cases from the case base. The result of the retrieval is one or more workflow adaptation cases. We refer to our previous work [25, 27] and to the literature (compare Section 6) for similarity measures that can be applied for the retrieval of workflow adaptation cases. The similarity measures consider both, the workflow structure, as well as the semantic description of the change request. In the current approach, we accept only one adaptation case per reuse phase. The adaptation mechanism reuses the best matching adaptation case. The adaptation knowledge from this case is to be transformed and applied to the new workflow.

1. a) Change request: “The DVD-ROM drive is not working any more.”

1. b) Original workflow before adaptation:

![Original workflow diagram](image1)

2. a) Workflow after adaptation:

![Workflow after adaptation diagram](image2)

2. b) ADD list:

```
Task (ID=165) --- Re-install driver --- Test CD-ROM drive --- AND (ID=167)
```

2. b) DELETE list:

```
---
```

Fig. 2. Sample workflow adaptation case for the workflow “Software update Win7 SP1”. 
4.1. Determine change location

In a manual adaptation scenario the workflow modeler has to determine manually where a workflow needs to be adapted. We aim at supporting the user in this step, hence aiming at automating workflow adaptation therefore. The positions for applying the edit operations in the target workflow have to be determined automatically for every chain. This problem can be solved by means of the anchor concept. The anchors of the chains have to be mapped to appropriate positions in the target workflow. Workflow elements are selected to specify the locations where to apply the chains of add and delete operations. The anchor mapping method (compare also Algorithm 1) consists of three steps: (1) First, it determines candidate positions for the anchors in the target workflow. (2) Then, it assesses the candidate positions to restrict the set of potential positions to a set of valid candidate positions for each anchor. (3) Last, the mapping from the anchors to the set of valid candidate positions is constructed. The resulting mapping might be partial, but it has to satisfy the following consistency criterion: A mapping of anchors to positions in the target workflow is called consistent if it preserves the inner order of anchor pairs with respect to the precedence relation induced by the workflow graph. This means that for any mapped pair of anchor sets the positions of the pre-anchors must be before to the positions of the post-anchors in the target workflow. A pair of candidate positions is called consistent pair of candidate positions if it does not violate the consistency of a mapping.

Fig. 3 illustrates the mapping method by the sample anchor pair from Fig. 2. The target workflow provides six candidate positions (three for the pre-anchor and three for the post-anchor). The result of the mapping is only a partial mapping as the pre-anchor could not be mapped to an appropriate position.

In the following, the three steps of the mapping method will be described in more detail. For simplicity reasons, we restrict the method to chains that form paths instead of arbitrary sub-graphs. This means, the anchor sets are restricted to singletons. Furthermore, a special ‘null’ element is used as an anchor in case an appropriate workflow element is not available, e.g., if a data object with a link to a task is newly created without having a predecessor. Algorithm 1 illustrates the following explanations.

1. The first task of determining candidate positions uses a filter for workflow elements that provides candidate positions. It would not be useful to take workflow elements into consideration that have already completed their execution. Hence, the execution state of the workflow elements is evaluated to filter out workflow elements that are not editable any more (those with the status “ACTIVE” or “READY” and the data objects remain). The remaining workflow elements provide the candidate positions. This first step of the mapping method can be omitted in case the workflows have not yet started execution. In this case, the positions of all workflow elements in the target workflow are regarded candidate positions.

2. The second step of the mapping method assesses the candidate positions for a particular anchor. The similarity of the element at the candidate position to the element at the anchor position in the chain is computed. All candidate positions for which this similarity value is above a validity threshold are considered valid candidate positions for an anchor. The other candidate positions are refused.

3. The third step selects valid candidate positions to construct a consistent mapping of anchors to candidate positions. Four different mapping methods have been investigated (compare Section 5).

Pre-anchor mapping (PRAM): The post-anchors are ignored, while for each pre-anchor the best matching candidate position is selected according to the workflow similarity measure. In case a candidate position is the best matching position for multiple pre-anchors, one pre-anchor is chosen arbitrarily and the others are mapped to the next best matching candidates. In Algorithm 1, the “for each” loops (b) and (c) are omitted.
Post-anchor mapping (POAM): The pre-anchors are ignored, while the best matching positions for the post-anchors are selected analogously to the PRAM method. In Algorithm 1, the “for each” loops (a) and (c) are omitted.

Composite anchor mapping (CAM): The mapping chooses candidate positions from the set of consistent pairs of valid candidate positions and from the set of single candidates as follows. A pair of valid candidate positions is valid for an anchor pair from the add list if the candidate position of the pre-anchor is a direct predecessor of the candidate position of the post-anchor. A pair of valid candidate positions is valid for an anchor pair from the delete list if it is an indirect successor of the candidate position of the post-anchor, i.e., there are workflow elements between the two candidate positions in the target workflow that form a path from the first candidate to the second candidate according to the precedence relation. The similarity value of a valid pair of valid candidate positions is the sum of the workflow element similarities at the pre-anchor and at the post-anchor positions. The similarity value of the best matching pair (in case there is one) is compared to the similarity value of the best matching single candidate according to the PRAM and PROM method. The pair or single candidate with the highest similarity value is chosen. In case a candidate is selected several times, one anchor pair is selected arbitrarily and the others are mapped to the next best matching pairs or single candidates. Algorithm 1 as it stands describes this method.

Maximum anchor mapping (MAM): This method is very similar to the CAM method. Just the ordering of candidate pairs is different: The sum aggregating the similarity values for the pre- and post-anchor positions of a pair is replaced by a maximum function. In Algorithm 1, line (d) is replaced.

The mapping result plays a different role for chains from add lists and from delete lists: In case of a chain from the add list, the anchor positions are the only criterion to determine the insert positions within the target workflow. In case of a chain from the delete list, the main criterion to determine the positions of workflow elements that have to be deleted from the new workflow is a mapping of the workflow elements from the delete list to workflow elements in the target workflow. Again, the similarity measure for workflow elements is applied. The best matching workflow elements are candidates for being deleted. The fitting anchor positions are a secondary criterion to prevent premature delete operations.

4.2. Application of changes

The positions of the anchors determined by the mapping method specify where to apply the changes to the target workflow. A chain is either fully applied or not applied at all. The add operations are executed immediately for all chains of which at least one anchor has been mapped successfully. In case there is a position for the pre-anchor, the add operations are performed starting at this point. Otherwise, the add operations are performed ending at the position of the post-anchor. Chains whose anchors could not be mapped cannot be applied automatically. They might be applied by the user who revises the adaptation result. Chains from the delete list are applied only if the mapping of their elements is complete and the similarity values for the individual workflow elements are above a delete threshold. Additionally, it is required that the elements to be deleted are organized in exactly the same control flow than those in the chain. Thus, the delete operations have to fulfill stronger constraints than the add operations to be applied.

5. Experimental evaluation

The described methods have been fully implemented as part of the Collaborative Agent-based Knowledge Engine (Cake) [4,26] developed at the University of Trier within several research projects. It uses CFCN as a process modeling language (see Section 2). The workflow technology offers allows late modeling and ad hoc changes of ongoing workflow instances. Cake also offers data modeling facilities using a common object-oriented data model and a comprehensive structural case representation framework, a library of configurable similarity measures, and a similarity-based retrieval component. As part of our current research, Cake has been extended as follows:

- the representation of adaptation cases,
- the application of the existing similarity measures for the retrieval of adaptation cases, and
- the reuse of adaptation cases as described in Section 4.

Algorithm 1. Construction of an anchor mapping.

input:
\( W \): target workflow a set of workflow elements with a precedence relation,
\( A \): set of anchors \([\text{preAnchor}, \text{postAnchor}, \text{anchorType}]\),
\( \Theta \): validity threshold

output:
map: set of mappings \([\text{anchor}, \text{workflowElement}]\)

begin
// Step (1) & (2) //
cands ← ∅

pos ← set of workflow elements ∈ W with execution state READY or ACTIVE or data objects

for each \( e \) ∈ pos do

    for each \([\text{preAnchor}, \text{postAnchor}, \text{anchorType}]\) ∈ A do

        sim₁ ← sim(preAnchor, e)

        if \( sim₁ ≥ \Theta \) do

            cands ← cands ∪ \([\text{preAnchor}, e, sim₁]\)

        end

        sim₂ ← sim(postAnchor, e)

        if \( sim₂ ≥ \Theta \) do

            cands ← cands ∪ \([\text{postAnchor}, e, sim₂]\)

        end

    end

for each \([\text{anchor}, e, sim₁]\) ∈ cands such that \( \text{anchor} = \text{preAnchor} \)

    valid ← valid ∪ \([e, \text{null}, sim₁]\)

end
adaptation cases. Most of the adaptation steps involve experiences in cooking resulting in a case base with 30 workflows were constructed by using our own common change request and the resulting change in the each workflow the interviewees were asked to describe a execution on a regular basis in our University. Then, for interview sessions with office secretaries, we first cooking instructions from recipes. In several structured processes as they typically occur in University offices and tion in two domains: the domain of administrative pro-

The evaluation itself is a big challenge, since immediately useful test data about workflow cases is not available. Although some public workflow repositories exist, none usef test data about workflow cases is not available. For the second domain, we created 39 cooking work-

The extended Cake system has then been used in a formative evaluation of the proposed adaptation methods. The evaluation itself is a big challenge, since immediately useful test data about workflow cases is not available. Although some public workflow repositories exist, none of them considers and records workflow changes. Therefore, we needed to spend a significant effort to develop and elaborate particular test scenarios for this evaluation. To evaluate our approach, it was necessary to obtain a case base of adaptation cases and a set of new test problems each of which consists of a target workflow and a change request. We decided to perform this evalua-

5.1. Evaluation goal and domains

The extended Cake system has then been used in a formative evaluation of the proposed adaptation methods. The evaluation itself is a big challenge, since immediately useful test data about workflow cases is not available. Although some public workflow repositories exist, none of them considers and records workflow changes. Therefore, we needed to spend a significant effort to develop and elaborate particular test scenarios for this evaluation. To evaluate our approach, it was necessary to obtain a case base of adaptation cases and a set of new test problems each of which consists of a target workflow and a change request. We decided to perform this evalua-

5.2. Experiment 1: adaptation case reconstruction

The aim of the first experiment was to evaluate whether each of the proposed reuse methods is able to correctly reconstruct the adaptations described in each of the 49 adaptation cases. For each adaptation case, the four variants of reuse methods have been applied to the contained original workflow, i.e., the description of the adaptation steps is used to compute an adapted workflow from the original workflow. By comparing the computed adapted workflow with the correct adapted workflow noted in the adaptation case, the correctness of the reuse method can be measured. The correctness then mainly depends on the capability to correctly reconstruct the anchor positions to which the adaptations have to be applied. As a quality criterion, we measured the average number of correctly applied add chains, wrongly applied add chains, and not applied add chains as well as the average number of correctly applied delete chains, wrongly applied delete chains, and not applied delete chains.

Fig. 4 shows the results of this experiment for the 19 adaptation cases from the office domain. There are two bars for each variant. The left bar shows the results for the 22 add chains and the right bar shows the results for the 7 delete chains occurring altogether in the 19 adaptation cases. Only CAM and MAM are able to correctly recon-

5.3. Experiment 2: adaptation of test problems

In the second experiment, the adaptation cases were applied to new workflows. For each test problem, the best matching adaptation case has been selected manually with support from a domain expert. Then, the adaptation steps in the selected case are applied to the target work-

---

2 The workflow collection and the adaptation case base are available at http://www.uni-trier.de/?id=26836.
3 See www.computercookingcontest.net.
transferring the respective adaptation steps to the target workflow.

In the office domain, we investigated the capability of the four reuse variants to correctly adapt the workflows of the 9 test problems. In this experiment, we used a high validity threshold of 0.9 for add lists. Hence, adaptations are only performed if there is high evidence that the positions are correct. Finally, the resulting adapted workflow is compared with the reference solution noted within the test problem description. The same quality criterion is applied as in experiment 1. Fig. 5 illustrates the results of this experiment. As in experiment 1 the left bar shows the results for the add chains and the right bar the results for the delete chains. The best results are achieved by CAM and MAM. However, there is no difference of the four variants applied to the delete chains and there is only little differences of CAM and MAM applied to the add chains.

To investigate the influence of the validity threshold, we lowered the validity threshold to 0.7 and run the experiment again. With a lower threshold, adaptation steps are transferred even if the matching of the anchors is less perfect, leading to increased matching opportunities. If the similarity measure is appropriate, we expect also an increasing number of correctly applied adaptation steps. The results are shown in Fig. 6. The best result was obtained by CAM leading to 7 correctly applied and none not applied add chains. With all four methods, the number of correctly applied adaptation steps is increased with the lower threshold, while the number of wrongly applied steps is not changed. These results indicate that at least in this setting a lower validity threshold improves the results of the anchor mapping.

In the cooking domain, we focused on the CAM method. This is because the adaptation of recipes involves data flow changes (on ingredients and their usage) which makes the adaptation cases and the parametrization of the algorithm more complex. The validity thresholds have been configured individually for data objects, data links, and other workflow elements and for add and delete operations. We tested
variants which differed by 0.1 for each parameter. It turned out for all different configurations that the following configuration achieved the best results for the test set: For the add chains, data objects have been inserted above a threshold of 0.1 and data links above 0.7. For the delete chains, a threshold of 0.8 for data object and 0.5 for data links were best. By accident, only one other workflow element (a task) occurred in an add chain and none in a delete chain in the overall collection of adaptation cases. It is not surprising that the configurations of the threshold for the other workflow elements always achieved the same results. Half of the test cases were adapted correctly. Fig. 7 depicts the results in more detail. All data objects could be added correctly while the correctness of added data links is only approximately 50%. Also only 60% of the possible delete operations of data links could be applied. It is quite likely that this effect is strongly dependent on the similarity measure used. This issue must be investigated in more depth in future experiments with improved similarity measures.

6. Related work

In the following, related work on four topics will be discussed: Approaches related to agile workflow technology, to flexibility of workflows at run time in general, to workflow retrieval, and to case-based adaptation.

Agile workflow technology has been implemented in different workflow management systems [30,38,36,25]. The following approaches allow the structural adaptation of workflows at run time. Some of them provide guidance for the workflow modeling experts who perform the adaptation. The main difference to our work is that they do not automate the adaptation. The ADEPT system [30] enables ad hoc modifications of single workflow instances at runtime and workflow definition evolution with instance migration, i.e., to propagate workflow definition changes to already running instances. Conversational CBR has been applied in the CBRFlow system [38] to guide the user in adapting a workflow to changing circumstances. Pockets of flexibility [36] allow the user to compose workflows dynamically from a previously defined set of tasks.

More restricted types of flexibility have been investigated in a large number of approaches (for a survey see [39]). In comparison to agile workflow technology and to our work, they provide a lower degree of freedom in adapting workflows. They are restricted to previously specified workflow structures, while agile workflow technology
facilitates structural changes at unforeseen parts of the workflow. Configurable workflows [12] allow to define workflow elements that can be switched on or off at run time. Exception handling [35] allows to annotate workflows with exception handling patterns that provide countermeasures for foreseen exceptions. Aspect-oriented service composition techniques [6] support dynamic adaptation of BPEL workflows at defined points (pointcuts) at run time. Model reuse technologies [17] require generalized representations of modification episodes to be reused.

Several approaches in the literature, including own previous work, employ workflow retrieval to support a human expert in the creation and adaptation of workflows. This goes beyond the use of workflow templates (process models) to build workflow instances as is widely spread in recent commercial workflow management systems. Similarity measures from case-based reasoning or graph matching algorithms are employed to automate the retrieval of workflows. However, a serious weakness of the discussed approaches in comparison to our case-based adaptation approach is that the actual modeling effort has to be performed by the human expert who applies the retrieved information. The workflow retrieval is applied to support the incremental modeling of workflow definitions [22], the selection of workflow definitions [4,9,21,10], or the modification of ongoing workflows [20,25,18,29,15]. A generic framework for representing and retrieving workflows has been introduced in our previous work [3] and another one is described in this special issue [16].

Several techniques for adaptation in CBR have been proposed so far (for a review see [8]). More recent approaches to adaptation in CBR are motivated by the fact that the acquisition of explicit adaptation knowledge (e.g., adaptation rules) is a very difficult and time-consuming task. Several methods have been developed that exploit the knowledge already captured in the cases as source of adaptation knowledge. In our own earlier previous work [40], we propose a general framework for learning adaptation knowledge from cases. Meanwhile, several successful examples of such methods exist that either apply inductive learning to extract adaptation knowledge [14,7,2] or that apply a case-based adaptation approach for an estimation task [23]. The adaptation approach proposed in this paper is a case-based adaptation as it is based on a dedicated case base of previous successful adaptations. Each adaptation case describes a particular kind of workflow modification that can be transferred to new situations. Hence, the adaptation case base can be considered experiential adaptation knowledge.

7. Conclusion and open issues

In this paper, we presented a novel framework for case-based adaptation of workflows together with results from a formative evaluation in two domains. Workflow adaptation cases store pairs of workflows and a change request describing the cause why the first workflow has been converted into the second. The adaptation steps are described by edit operations (add and delete steps), which are applicable to similar workflows with similar change requests as well. Organizing the edit operations in chains following the correctness-by-construction principle maintains the syntactic correctness during change reuse. The reuse phase consists of two tasks: First to determine the change location in the target workflow and second to apply the changes from the retrieved case. An anchor concept has been introduced to find appropriate locations for chains of edit operations. The evaluation on sample workflows from the office domain has shown that considering two anchors per chain (CAM method) performs better results than single anchors only (PRAM, POAM, and MAM methods). The anchor concept has turned out to be suitable for transforming adaptation knowledge from one workflow to another in two real-world domains.

Our work so far has confirmed the feasibility of our approach to automatic case-based adaptation of workflows and provided formative results. The next research step that is planned to prove our hypothesis is to put the system in use and perform a summative evaluation including an empirical study with an application partner. Further, open issues arise that concern the quality of the adaptation results as well as the acquisition and maintenance of the adaptation cases. In our future work, we will investigate more sophisticated anchors integrating several workflow elements. Further, similarity measures will be investigated including semantics. Furthermore, application-related change patterns [20,39] of basic edit operators could be investigated. For instance, high-level change operators like “replace” or “move” could be implemented for such macros or patterns. This would increase the readability of the chains of edit operations, for instance in applications where they cannot be derived automatically from a given adapted workflow. Further, domains like e-science workflows, media production workflows, or software installation instructions could be investigated. We believe that our results could be useful for process change management in many further application areas.

References
