# Adaptation of Cooking Instructions Following the Workflow Paradigm

Mirjam Minor, Ralph Bergmann, Sebastian Görg, Kirstin Walter

Business Information Systems II
University of Trier
54286 Trier, Germany
{minor|bergmann|goer4105|walt4701}@uni-trier.de

**Abstract.** Former CCC systems have considered mainly the ingredients of cooking recipes. This paper contributes to the open challenge with a novel approach that targets the preparation instructions. Our demo system ***CookingCakeWf*** employs the workflow paradigm to represent and adapt cooking instructions in form of cookery workflows. Adaptation cases on former modification episodes of cookery workflows are reused for current change requests. A small experimental evaluation with cookery workflows created from pasta recipes of the CCC 2010 recipe base provides first insights into case-based adaptation of cookery workflows.

## 1 Introduction

The International Computer Cooking Contest (CCC) is going into its third year. Cooking recipes are given in a case base to be retrieved and reused with the assistance of a computer system. The participating teams compete by their systems providing recipes in order to answer cooking wishes. The main task has been nearly the same over the years while additional challenges are changing to address recent developments in Case-based Reasoning. This year, the contest includes an open challenge for the first time. This paper adopts the open challenge by focusing on the cooking instructions and on how to adapt them to cooking wishes. For this, the textual cooking instructions are formally represented. We employ the workflow paradigm to represent cooking instructions as cookery workflows. A case-based adaptation method [1] developed for agile workflow technology is extended to adapt cookery workflows. Both, the control flow (of cooking steps) and the data flow (of ingredients and their products) are considered. The latter has not yet been addresses by previous work on automated workflow adaptation [1]. The output of our CCC open challenge system **CookingCakeWf** is an adapted workflow not yet a textual cooking instruction. The automated generation of text from the workflow representation is a topic of future work as well as the automated transformation of the original textual description of a cooking description into a formal workflow. Additionally, we

participate at the CCC main challenge with our last year's system **CookingCake**[1] which is described in the literature [2].
The paper is organized as follows: Section 2 introduces a formal representation of the cookery workflows. Section 3 presents a novel case format for adaptation knowledge. Section 4 describes the methods for applying the adaptation knowledge to the workflows. Section 5 demonstrates the feasibility of the approach by means of a first experimental evaluation.

```
<RECIPE>
        <TI>Baked Spaghetti</TI>
        <IN>1 c Chopped onion</IN>
        <IN>1 c Chopped green pepper</IN>
        <IN>1 tb Butter/margarine</IN>
        <IN>1 cn (28 oz.) Tomatoes with liquid; cut up</IN>
        <IN>1 cn (4 oz.) Mushroom stems and 1 piece - drained</IN>
        <IN>1 cn (2-1/4 oz.) Ripe olives, sliced and drained</IN>
        <IN>2 ts Dried oregano</IN>
        <IN>1 lb Ground beef, browned and drained (optional)</IN>
        <IN>12 oz Spaghetti, cooked & drained</IN>
        <IN>2 c (8 oz.) shredded cheddar cheese</IN>
        <IN>1 cn (10-3/4 oz) Condensed cream</IN>
        <IN>Mushroom soup; undiluted</IN>
        <IN>1/4 c Water</IN>
        <IN>1/4 c Parmesan cheese, grated</IN>
        <PR>In a large skillet, saute onion and green pepper in butter until
        tender. Add tomatoes, mushrooms, olives and oregano. Add ground
        beef if desired. Simmer, uncovered, for 10 minutes. Place half of the
        spaghetti in a greased 13-inch x 9-inch x 2-inch baking dish. Top
        with half of the vegetable mixture. Sprinkle with 1 cup of cheddar
        cheese. Repeat layers. Mix the soup and water until smooth; pour
        over casserole. Sprinkle with Parmesan cheese. Bake, uncovered, at
        350 degrees for 30-35 minutes, or until heated throughout. </PR>
</RECIPE>
```

**Fig. 1: Sample cooking recipe on Baked Spaghetti from the CCC recipe base.**

## 2 Cookery workflows

Cooking recipes are usually described by a list of ingredients and a cooking instruction. Fig. 1 shows a sample recipe description of a pasta recipe from the CCC recipe base in XML. The title of the recipe (<TI>…</TI>) is followed by a list of ingredients (<IN>…</IN>) and a textual cooking instruction (<PR>…</PR>). Many internet platforms on cooking use a comparable XML representation of the cookery data.
This semi-structured representation has been transformed into a more formal workflow representation for our CCC system by a human expert. Workflows describe processes by means of *tasks* (activities) that are organized within a control flow. The

[1] See also http://proj2.wi2.uni-trier.de/

CAKE workflow modeling language (compare [3]) consists of several types of *workflow elements* whose instances form block-oriented control flow structures. A workflow element can be a task, a *start symbol*, an *end symbol*, or a *control flow element* like an AND, XOR, loop etc. Control flow elements always define related blocks (AND-block, XOR-block etc.). Blocks cannot be interleaved but they can be nested. In addition to the control flow, a workflow can have a data flow, which is the flow of *data objects* from one workflow element to a successor workflow element. The data flow is specified by means of *data links* that connect the data objects with workflow elements. In a *cookery workflow*, every ingredient is considered a data object while the preparation steps form the workflow tasks within a control flow. Preparation steps and control flow are extracted from the textual cooking instruction as well as the data links that connect the data objects (ingredients) with the tasks (preparation steps). Fig. 2 a) illustrates this by showing the workflow representation for the sample recipe from Fig. 1. Some challenges concerning the control flow (1), the data objects (2), and the data links (3) had to be be faced during the transformation:

(1) The cooking instructions suffer from both, a granularity and a paraphrase problem. *Granularity* means that some authors describe instruction steps in great detail (e.g. peel and chop onions, melt butter in a large skillet, add onions and beef, wait until brown) while others aggregate minor steps to higher-level activities (e.g. brown beef and onions). Despite granularity models have been discussed in the workflow literature [4], we have decided to avoid the granularity problem in the work by harmonizing the granularity of workflow tasks by hand. However, the granularity problem has to be solved in future work on the automated transformation of recipes into the workflow representation. The second challenge when defining the control flow of cookery tasks is the vocabulary. The *paraphrase problem* occurs here as well, which addresses the non-uniform wording used in texts from different authors (e.g. sauté vs. brown). Using a task ontology would solve this problem to a large extent (compare the discussion at the end of Section 4).

(2) The representation of data objects (ingredients) raises some issues concerning the preparation states, amounts, and double (or multiple) occurrences of ingredients. Although these issues have been discussed by former CCC systems already, they deserve special attention in the context of workflow adaptation. The *preparation states* of the ingredients (one piece, chopped, sautéed, cooked etc.) could be represented by multiple data objects (whole onions, sliced onions, chopped onions, etc.) or by the same data object with the preparation state represented as a variable property. The *amounts* of the ingredients can be represented as a data object property as well. *Double (or multiple) occurrences* of ingredients in different roles (e.g. butter to brown onions and flakes of butter for a topping) can be represented either as different objects (e.g. butter and flakes of butter, or butter one and butter two) or as a single object. For simplicity reasons, we have chosen the latter and to abstain from representing any properties (amounts, state of

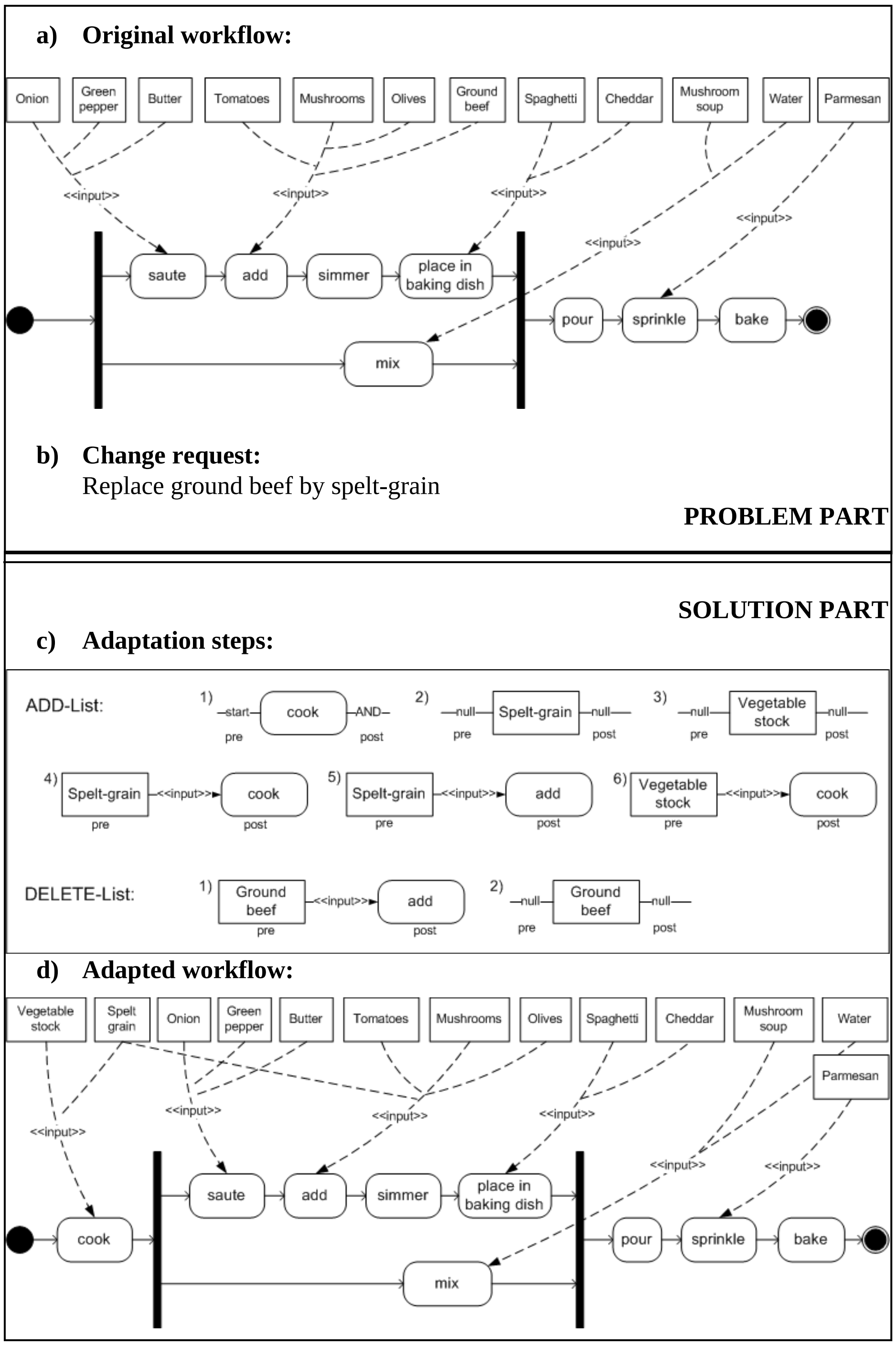


**Fig. 2: Adaptation case on the sample cooking recipe from Fig. 1.**

preparation) at the moment. In the future, this approach can be improved by an extended representation.

(3) Preparation tasks consume ingredients and produce transformed ingredients or aggregates of ingredients. Data links connect data objects with tasks. In general workflows, the data objects can play the role of *input or output data* for the tasks. A complete model would include an input connection between a data object and every task using it as an input and an output connection for every product resulting from a task. Data objects for aggregated objects (e.g. a document collection made from two different forms and a copy of a certificate, or a sauce made from onions and tomatoes) would be created by the task that aggregates the objects. However, textual cooking instructions tend to be underspecified due to the human principle of economy in language. Our analysis of the CCC recipe base has shown that the data links for the input of ingredients are more frequently described than for the outputs. Transformed ingredients and aggregates occur very seldom in the instructing texts. Hence, we do consider only inputs at their first occurrence, except for those ingredients that occur in multiple roles as described above.

## 3 Adaptation cases for cookery workflows

An adaptation case represents the knowledge from a previous adaptation episode of a workflow. Our case representation consists of a problem and a solution part as following: The *problem part* contains a semantic description of the *change request* (specifies a desire to modify the recipe, e.g. to replace ground beef by spelt-grain to reduce the fat content) and the *original workflow* prior to the adaptation. The *solution part* contains the *adapted workflow* and the description of the *adaptation steps* that have been executed to transform the original workflow into the adapted workflow (added and deleted workflow elements, e.g. delete the ground beef with its data links and add new data objects for spelt-grain and stock plus the preparation steps for preparing them and the required data links). Fig. 2 shows a sample adaptation case by which baked spaghetti have been made according to the original recipe given in Fig. 2 a) but with spelt-grain instead of ground beef. Fig. 2 d) shows the adapted workflow: The spelt-grain has to be cooked in stock before it can be added. Please note that aggregates and preparation states are omitted in the current representation. A complete model would include an aggregated object for “cooked spelt-grain” with an output data link from the task “cook” and an input data link to the task “add”.
Like in our previous work [1], the adaptation steps are organized within an *add* and a *delete* list. In extension of the previous work, an add or delete step may refer to a workflow element, a data object or a data link. The add and delete lists organize the add and delete steps in the form of chains. A chain encapsulates a set of adaptation steps on connected workflow elements, data objects, and data links. A chain is intended to be either fully applied or not applied at all while reusing the adaptation case. Further, each chain records a pair of *anchors*. A *pre anchor* is the workflow element or data object (in the original workflow) after which the add or delete steps

from the chain have been applied. A special ‘null’ element is used as anchor in case an appropriate data or workflow element is not available, for instance if a data object is newly created or deleted. A *post anchor* is the workflow element from the original workflow following the last element of the chain. We assume that data objects do not play the role of a post anchor at the moment. This might change in future in case output data links will be included. Hence, the pre anchor describes the position after which the adaptation starts and the post anchor describes the first position in the original workflow that is not anymore affected by the adaptation described in the chain. Fig. 2 c) illustrates this by sample add and delete lists. We have decided to model very fine granular chains in order to apply as much of the adaptation steps as possible. Another modeling strategy would be to maximize the chains, for instance to combine the add chains 1, 2, 3, 4, and 6 to one chain with one anchor pair. This would reduce the effort for the anchor matching at the cost of a higher modeling effort to revise the workflow.

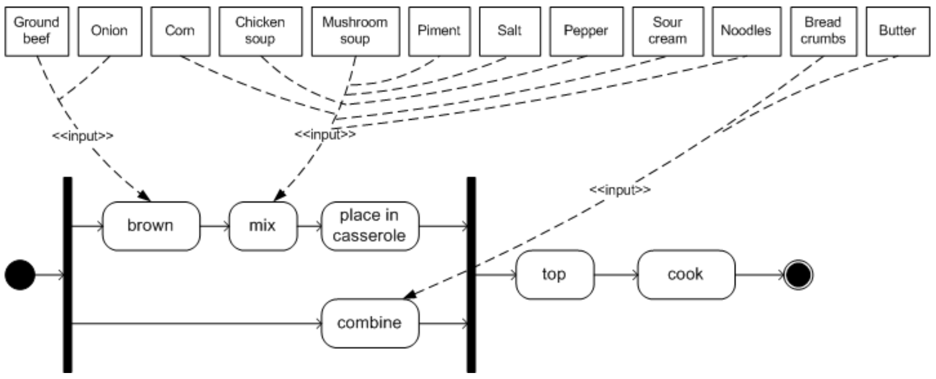
## 4 Applying adaptation cases to new situations

We now focus on the retrieve and reuse phase of the CBR cycle [5]. The revise and retain phases are not yet considered to be automated. The retrieval of workflow adaptation cases that may be reused for the adaptation of new cookery workflows is also not within the scope of this paper. We refer to the literature [1] for a discussion of similarity measures that come into consideration for the retrieve phase. Fig. 3 depicts a sample query for which the adaptation case shown in Fig. 2 would be applicable.

The reuse phase has to solve two main issues: First to determine the locations where the adaptation steps from the retrieved adaptation case should be applied to the target workflow and, second, to execute these adaptation steps. The change locations in the target workflow are determined by mapping the anchors from the retrieved case. The composite anchor mapping method described in our previous work [1] is extended in order to consider the data flow in addition. The mapping method consists of two steps, which we briefly sketch below:

(1) Valid candidate positions for anchors are chosen within the set of workflow elements and data objects of the target workflow. A data object anchor can be mapped to the position of a data object in the target workflow only if the data objects are sufficiently similar (above a validity threshold for data objects). Workflow element anchors can be mapped only to similar workflow element positions analogously (above a validity threshold for workflow elements). Similarity functions for both, data objects and workflow elements will be discussed below. The valid candidate positions are described by a set of triples [$wfl\_el_{retrieved}$, $wfl\_el_{target\_wfl}$, $sim_{wfl\_el}(wfl\_el_{retrieved}, wfl\_el_{target\_wfl})$] and [$data\_obj_{retrieved}$, $data\_obj_{target\_wfl}$, $sim_{data\_obj}(data\_obj_{retrieved}, data\_obj_{target\_wfl})$] with similarity values higher than the specified validity thresholds.

**a) Original workflow:**



**b) Change request:**

Replace ground beef by spelt-grain

**Fig. 3: Sample query to adapt a recipe on a beef noodle casserole.**

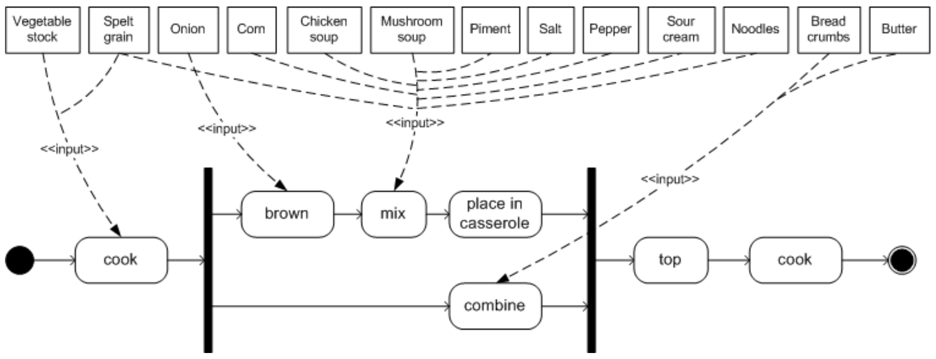


**Fig. 4: Adaptation result (reference solution).**

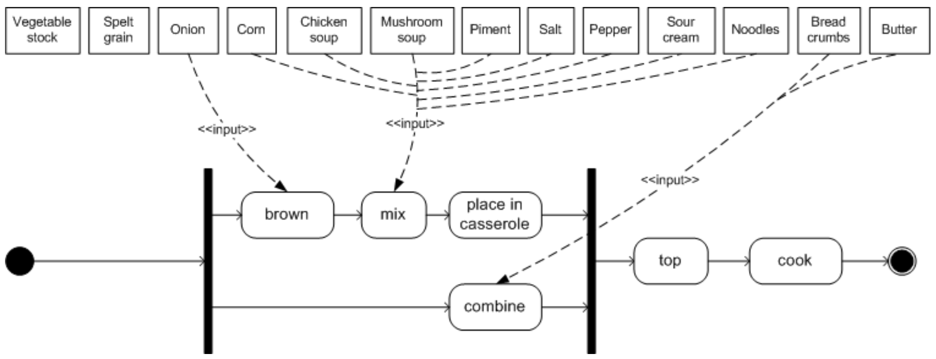


**Fig. 5: Adaptation result (actually generated solution).**

(2) The best matching positions from the set of valid candidate positions are selected to construct the anchor mapping. Pairs of workflow element anchors (pre and post anchor) have to preserve their order in the target workflow according to a precedence relation of workflow elements that is induced by the control flow. Additionally, the mapped positions of pairs of workflow element anchors must be at direct neighbors according to the same precedence relation in case of a chain from the add list while they must not be direct neighbors in case of a chain from the delete list. The mapping algorithm employs a hill climbing search to find mapping positions with an optimal overall anchor similarity.

The application of the adaptation chains is different for add and delete operations. The add operations are executed immediately for all chains of which at least one anchor has been mapped successfully. Chains from the delete list are applied only if a complete mapping of their particular delete steps to workflow and data flow elements can be constructed. The mapping algorithm for delete steps considers pairs of elements (workflow elements, data objects and data links) from the retrieved and the target workflow whose similarity value is above a threshold called *delete threshold*. Additionally, it is required that the elements to be deleted are organized in exactly the same structural order (control flow or data flow) than those in the chain. Thus, in order to be applied, the delete operations have to fulfill stronger constraints than the add operations.

The similarity measures for workflow elements ($sim_{wfl\_el}$), data objects ($sim_{data\_obj}$) and data links ($sim_{data\_links}$) that are required during the reuse phase are specified as follows:

$$sim_{wfl\_el}(x,y) = \begin{cases} sim_{task}(x,y) & , \quad x,y \text{ are tasks} \\ sim_{ctrl\_flow\_el}(x,y) & , \quad x,y \text{ are control flow elements} \\ 0 & , \quad else \end{cases}$$

$sim_{wfl\_el}$ distinguishes tasks from control flow elements. In case of tasks it aggregates the similarity measure for the sets of input parameters ($sim_{inputs}$), the similarity measure for the task name ($sim_{task\_name}$), and the similarity of the task description ($sim_{task\_descr}$) by a weighted sum in $sim_{task}$. The three constituent measures of $sim_{task}$ are specified based on the Levenshtein distance. The Levenshtein distance is purely syntactic and measures the minimum number of edit operations on the character level that is required to transform one string into another. $sim_{task\_name}$ and $sim_{task\_descr}$ employ the Levenshtein distance directly on the task name and on the textual task description. $sim_{inputs}$ matches the input parameters (data links) based on the Levenshtein distance of the names of their according data objects by means of a hill climbing search. $sim_{control\_flow\_el}$ measures the similarity of the set of tasks included in the block in case of a block-building control flow element like AND-split or AND-join (again, a hill climbing search is applied to map the two sets of tasks to each other). $sim_{data\_obj}$ employs the Levensthein distance on the names of the data objects as the data objects do not yet store any properties like amounts or preparation states. If the special data object ‘null’ is one of the arguments of $sim_{data\_obj}$, the value is set to 0

and to 1 if both arguments are ‘null’. $sim_{data\_links}$ aggregates the local similarity measures $sim_{data\_obj}$ and $sim_{wfl\_el}$.
Fig. 4 and 5 illustrate that the chosen similarity measures are still to be improved as the automatically generated solution in Fig. 5 could not transfer all adaptation steps from the case described in Fig. 2 c). For instance, the data links from spelt-grain to mix is missing as the syntactic similarity measure was not able to map the anchor at “add” in the retrieved workflow on “mix” in the target workflow. Also, the task names within the AND block of the retrieved workflow {“saute”, “add”, “simmer”, “place in baking dish”, “mix”} have not been similar enough to the task names within the AND block of the target workflow {“brown”, “mix”, “place in casserole”, “combine”} so that the post anchor for adding the task “cook” could not be positioned in the target workflow. Semantic similarity measures, for instance based on a task ontology, would probably provide better results.

## 5 Experimental evaluation and discussion

The system is implemented in a demo version. As starting point for a first evaluation we use a reduced recipe base containing 39 pasta recipes from in the CCC recipe base. Based on seven recipes taken from this pasta excerpt 30 different change requests were constructed by using our own experiences in cooking. Typical change requests replace one ingredient by another or avoid a certain ingredient. Thereby, an experimental case base of 30 adaptation cases (see Chapter 4 for the case structure) was created by hand. Most of the included adaptation steps concern the data objects and the data links, some adaptations involve adding or deleting workflow elements (preparation steps). We conducted two experiments described below.

The aim of the first experiment was to evaluate whether the suggested adaptation method is able to correctly reconstruct the adapted workflows from the adaptation steps described in the 30 adaptation cases. This requires finding the proper anchor positions, which should not be affected by the chosen similarity measures as only identical matches are required for this reconstruction. In line with our hypothesis, all adapted workflows could be reconstructed correctly.

In the second experiment the adaptations cases were applied to new recipes. For this purpose, 14 test queries have been created by arbitrarily selecting 14 different change requests from the adaptation case base of the first experiment and combining them with other cookery workflows. We selected recipes from the pasta recipe base for that the change requests can be tastefully applied. For instance, a change request on replacing ground beef by spelt-grain applies to a recipe only that contains ground beef as an ingredient and that would still be tasty when becoming vegetarian.

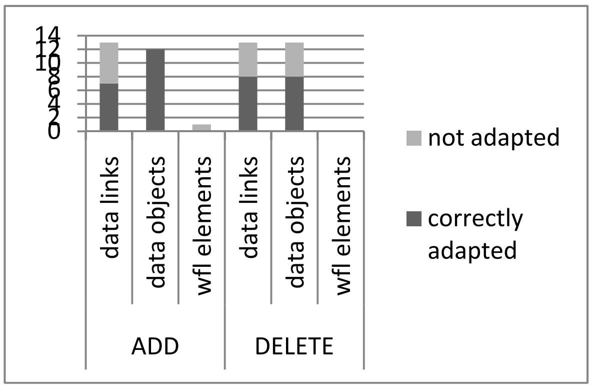


**Fig. 6: Results of Experiment 2.**

As the retrieval is not part of this evaluation the best-fitting adaptation case is chosen for each query by hand. The resulting adapted workflow is compared to a solution obtained by manually transferring the respective adaptation steps to the workflow (see Fig. 6). Altogether, half of the test cases were adapted totally correctly. In the remaining cases, at least the data objects could be added correctly, while the correctness of added data links is only approximately 50%. Also only about 60% of the possible delete operations of data links are applied. This is a clear indication that the currently used syntactic similarity measure is insufficient for our purposes. Further experiments with improved similarity measures using the CookingCake's ingredient ontology [2] as well as a task ontology (for preparation steps) will be necessary. However, we believe that the first results are quite promising and confirm that our idea on following the workflow paradigm during the automated adaptation of cookery instructions is feasible in principle.

## 6 References

1. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards Case-Based Adaptation of Workflows. In: Proc. ICCBR'10, to appear.
2. Fuchs, C., Gimmler, C., Günther, S., Holthof, L., Bergmann, R.: Cooking Cake, In: S. Delany (ed.) ICCBR'09 Workshop Proceedings, pp. 259 – 268, 2009.
3. Minor, M., Tartakovski, A., Schmalen, D., Bergmann, R.: Agile Workflow Technology and Case-Based Change Reuse for Long-Term Processes. International Journal on Intelligent Information Technologies 4(1), pp. 80-98, 2008.
4. Holschke, O., Rake, J., Levina, O.: Granularity as a Cognitive Factor in Effectiveness of Business Process Model Reuse, In: U. Dayal et al. (Eds.), BPM 2009, LNCS 5701, pp. 245 – 260, Springer, 2009.
5. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications, 7(1), pp. 39-59, 1994.