# Acquiring Adaptation Cases for Scientific Workflows

Mirjam Minor and Sebastian Görg

University of Trier
Department of Business Information Systems, D-54286 Trier, Germany
{minor,goergs}@uni-trier.de
http://www.wi2.uni-trier.de/

**Abstract.** This paper addresses the automated acquisition of adaptation cases for the modification of scientific workflows. Pairs of workflow versions from community repositories are analysed to extract transformation pathways from one workflow version to another. An algorithmic solution is provided and investigated by experiments with promising results.

**Keywords:** scientific workflows, workflow reasoning, adaptation.

## 1 Introduction

Scientific workflows [3] are dedicated to support data-intensive scientific experiments by applying workflow technology. A workflow organizes work by *tasks* that descibe a human activity or a computational step. The workflow arranges the tasks and the according data in a certain execution order. This order can be specified by data dependencies forming the *data flow* as well as by routing constructs like sequences, parallel branches, sub-workflows and multiple instances that govern the *control flow* of execution. Prominent application areas for scientific workflows are molecular biology, astronomy, geology, or atom physics. Sample tasks are data transformation and analysis steps or data extraction steps like accessing measurement readings from an external database. The workflow technology controls the execution of the experimental process and operates large parts of it *in silico*, i.e. in the computer [8]. In business workflow scenarios, human tasks are rather prevalent while scientific workflows use to be computational workflows that consist exclusively of computational steps. An increasing number of Web services is available for computational steps from different fields like genome analysis in bioinformatics. BLAST (Basic Local Alignment Search Tool)[1], for example, provides many Web services on the analysis of biological sequences. It has several benefits to represent an experiment formally as a workflow instead of copying the outputs of one computational step to the inputs of another step by hand or by a program written in a script language. The scientists are enabled "to focus on domain-specific (science) aspects of their work,

[1] http://blast.ncbi.nlm.nih.gov/Blast

rather than dealing with complex data management and software issues" [8, p. 32], the execution of the experiments can be optimized on available ressources in a distributed environment, the provenance of the output data can be recorded, and the scientific workflows can be shared and reused.

*Workflow reasoning* is recently an emerging research field that provides automated methods for reasoning about workflows and execution traces [7, 12, 6, 2]. This work addresses the automated adaptation of scientific workflows as reasoning method and particularly the automated acquisition of adaptation cases for scientific workflows. An *adaptation case* records experience from a previous workflow adaptation episode. Adaptation cases occur frequently in scientific workflows. For instance, an update of a Web service with slightly different input parameters may require additional data transformation steps in the scientific workflow. This is a sample for an adaptation case that occurs during workflow modeling at build time. An adaptation episode may also occur during workflow execution at run time caused by an unforeseen event, for instance if the quality of an intermediary result is not sufficient. Additional steps may be inserted, for instance a split of the data into two subsets before re-running some computational steps. Of course, changes at run time are only feasible in case of an agile workflow system [11] that is able to continue the execution of workflows that have been adapted. In our previous work on workflow reasoning [11, 10], we investigated agile workflow technology including case-based adaptation of business workflows. An automated adaptation support has turned out to be benefitial for the persons that are responsile for the workflow modeling and adaptation. As an adaptation of scientific workflows has to consider the data flow in addition to the control flow, it is an even more challenging modeling task than the adaptation of business workflows that focus mainly on the control flow. Hence, an automated support by adaptation cases would be very benefitial for the scientists dealing with scientific workflows.

In this work[2], we aim at (I) *confirming the hypothesis that case-based adaptation methods are applicable for scientific workflows* at all and (II) *developing a novel method for extracting adaptation cases automatically* from community repositories of scientific workflows. The opportunities to apply workflow adaptation cases - may they be acquired automatically or by hand - go even beyond the traditional Case-Based-Reasoning (CBR) idea of case reuse: The transformation pathways from one workflow to another that are recorded by an adaptation case can be used to visualize or measure deviations between workflows addressing the same topic for compliance or reconcilability purposes. Scientific workflows may be checked, for instance, if teams of scientists aim to cooperate. Business workflows may be compared, for instance, after mergers and acquisitions. The remainder of the paper is organized as follows: In Sect. 2, workflow adaptation cases are introduced. Sect. 3 sketches the case-based adaptation of workflows. In Sect. 4, the automated acquisition of adaptation cases from community repos-

[2] This work is part of the WEDA project. WEDA is funded by Stiftung Rheinland-Pfalz für Innovation, grant no. 974.

itories is presented. Sect. 5 deals with an experimental evaluation. In Sect. 6, related work is discussed. A conclusion is drawn in Sect. 7.

## 2 Workflow Adaptation Cases

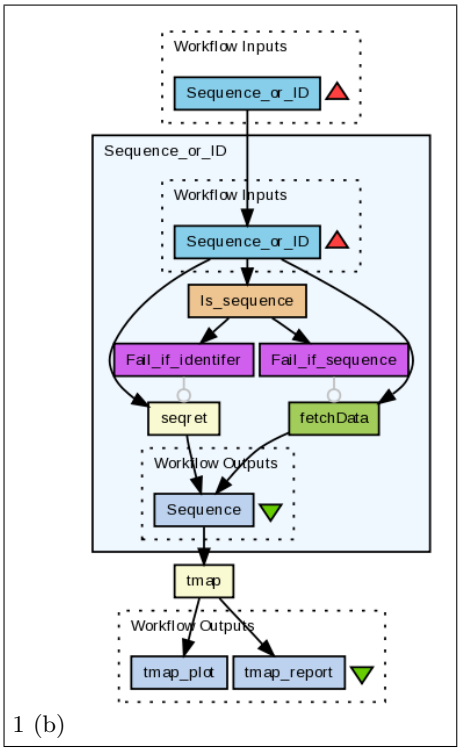
Case-based adaptation methods for workflows use adaptation cases for recording adaptation episodes. The case structure is impacted by the workflow language in which the workflows are specified. Most of the present workflow languages are graph-based in the sense that they consist of *workflow elements* as atomic parts organized in a flow-oriented manner, i.e. the elements can be represented by nodes and edges forming the flow of tasks. In this work, we focus on graph-based languages. We use the Simplified conceptual unified workflow language (Scufl) [13] as an example of an XML-based language whose elements describe nodes and edges of a workflow graph. Scufl workflows are dedicated to scientific workflows that are to be executed by the Taverna system [13]. Scufl has the following types of workflow elements that are subsumed to the set of nodes: tasks (`<s:processor>`) including placeholder tasks for sub-workflows, data objects for workflow inputs (`<s:source>`), and data objects for workflow outputs (`<s:sink>`). In the sample on the left hand side (1b) of Fig 1, the workflow has five nodes at top-level depicted by rectangular boxes: one workflow input node, two workflow output nodes, one placeholder task for the sub-workflow 'Sequence_or_ID' (containing further nodes at sub-workflow level), and one task 'tmap', which calls a Soaplab invocation as indicated by the light colour. Other types of Scufl tasks call on, for instance, a single Web service operation or a local Java function. The type of a task as well as further properties like input and output ports are specified as child elements of the XML element but do not belong to the set of nodes in the graph representation and are consequently not depicted. The edges in the workflow graph are derived from the following types of workflow elements of Scufl: Data flow edges (`<s:link>`) are depicted as arcs; coordination constraints (`<s:coordination>`) play the role of control flow edges as depicted by a connecting line with a circle. In the sample workflow (1b) of Fig. 1, such a constraint is specified, for instance, between the tasks 'Fail_if_identifer' and 'seqret', which means that task 'Fail_if_identifier' has to be completed before 'seqret' can be scheduled for execution. The details of the coordination constraints are specified again by child elements of the XML elements but are likewise not part of the graph representation.

An adaptation case can now be described as follows (compare our previous work [10]):

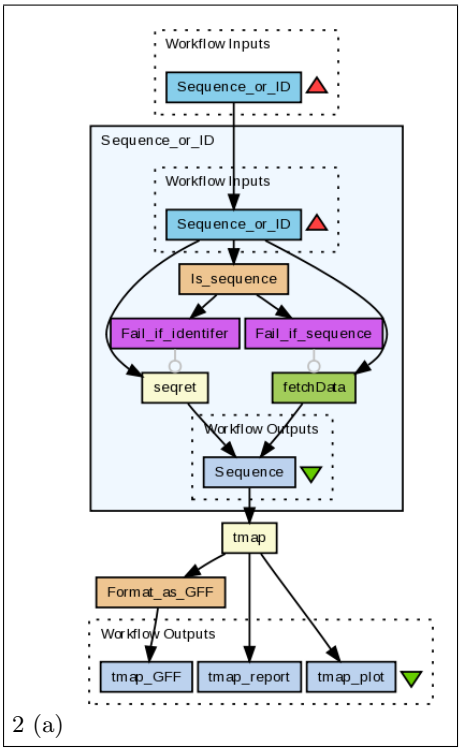
1. The *problem part* consists of
   (a) a semantic description of the change
   (b) a graph-based representation of the anterior workflow version prior to the adaptation.

2. The *solution part* contains

1 (a) Change description: “Add GFF output.”




1 (b)




2 (a)

2 (b)

```
<ADDList>
  <Chain>
    <Pre>
      <s:processor name="tmap" refID="20" >
    </Pre>
    <WorkflowElements>
      <s:link source="tmap:outfile" sink="Format_as_GFF:tmap_output"
        sourceID="20" sinkID="52" refID="51" />
      <s:processor name="Format_as_GFF" refID="52" />
      <s:link source="Format_as_GFF:tmap_gff" sink="tmap_GFF"
        sourceID="52" sinkID="54" refID="53" />
      <s:sink name="tmap_GFF" refID="54" />
    </WorkflowElements>
    <Post>
    </Post>
  </Chain>
</ADDList>

<DELList>
</DELList>
```

**Fig. 1.** Sample adaptation case with two versions of a workflow for the analysis of proteine genomes (retrieved from the workflow repository `www.myexperiment.org`).

(a) the posterior workflow version, i.e. the adapted workflow, in graph-based representation
(b) the description of the adaptation steps(added and deleted workflow elements) that have been executed to transform the anterior workflow version into the posterior.

Fig. 1 depicts a sample adaptation case in that an additional processing step 'Format_as_GFF' with an additional data output object 'tmap_GFF' is inserted into the workflow (compare the two workflow versions depicted in (2a) and (2b)). The semantic change description characterizes the changes that have been made from the anterior to the posterior version. This part of the problem description makes use of traditional case representation approaches, e.g. a structural representation or a textual representation. The sample change description in (1a) of Fig. 1 is a text. Workflow versions are represented in a graph-based way by sets of nodes and edges as described before. The representation of the adaptation steps deserves some special attention. Similar to STRIPS operators, 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 according edit operations are either fully applied or not applied at all while reusing the adaptation case. Furthermore, each chain records a pair of anchor sets that describe the positions within the workflow graph where the edit operations have taken place. The pre anchors are the workflow elements (in the anterior workflow) after which workflow elements from the chain have been added or deleted. The post anchors are the workflow elements from the anterior workflow following the last elements of the chain. Hence, the set of anchors describes the connectors at which a sub-graph has been inserted or pruned out. The XML snippet in the lower part of Fig. 1 shows a representation of an add list with scufl elements. It contains one chain of four workflow elements to be inserted namely the task 'Format_as_GFF', the output data object 'tmap_GFF' and two data links. The chain has only one connector to the anterior graph namely the 'tmap' task. It is a pre anchor as it is connected via an outgoing edge with the new sub-graph. The set of post anchors of this chain is empty as well as the delete list of the entire case. The 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. (2a) is mostly redundant within the case structure as it could be reconstructed from (1b) and (2b). It is recorded for reasons of readability only (graph layout).

## 3 Case-based Workflow Adaptation

Although the workflow adaptation itself is not in the scope of this work we will briefly sketch the case-based method for workflow adaptation to make the automated creation of adaptation cases more plausible to the reader. The case base consists of adaptation cases. The new problem to be solved (query) consists of a target workflow (which may be already partially executed) and the description

of the current change request to the target workflow. Hence, the similarity measure must be able to assess the similarity of two change descriptions as well as of two workflows. Please refer to our previous work [11] and to the literature [9, 1] for similarity measures for workflows. The similarity-based retrieval provides the most relevant adaptation cases from the case base. The best matching case is selected for reuse (either automatically according to the values of the similarity function or by user interaction). In the reuse phase, the best matching adaptation case is applied to the target workflow in order to adapt it according to the change request. This occurs in two distinct steps. First, the concrete location in the target workflow is determined that needs to be changed. This is necessary as there are usually many different positions within the workflow at which a chain from the add list can be inserted or to which the deletions in the delete list can be applied. Second, the changes are applied to the target workflow at the determined locations. The resulting adapted workflow is then the proposed solution. During the subsequent revise phase the user can validate the workflow adaptations proposed: she can either confirm them or revise them by manually performing appropriate adaptations herself. First experiments on automated, case-based adaptation of workflows have been conducted successfully (see [10] for the results). Hence, we can now turn to the case acquisition task and present a solution to automate this tedious work in the following.

## 4 Case Acquisition

The automated acquisition of workflow adaptation cases follows the idea of determining the difference between a pair of subsequent workflow versions and deriving a case from this delta. A huge amount of scientific workflows in different versions are available in community repositories[3]. Mostly, machine-readable representations of the workflow graphs are available and textual change descriptions are stored as revision comments. Hence, cases can be acquired from the content of such repositories by the following steps:

1. Extract pairs of subsequent workflow versions with a change description
2. Derive atomic edit operations (add an element, delete an element) from the differences of the sets of workflow elements from both versions
3. Organize the edit operations in chains with anchors.

The first step of the case acquisition is to select subsequent pairs of workflow versions and store their formal representations together with the textual description of the change in parts 1 (a),(b) and 2 (a) of a new case. Cases with an empty change description are not considered.

The second step is to gain the atomic edit operations from the formal representations. Set differences are determined for each type of workflow element as follows. The hierarchical structure of the top-level workflow with all nested

[3] Sample repositories for scientific workfows are available at `www.vistrails.org` and `www.myexperiment.org`

sub-workflows is analysed and recorded in a *sub-workflow tree*. The root node of the sub-workflow tree stands for the top-level workflow while the other nodes represent a sub-workflow at the according level each. Fig. 2 a) illustrates this by a sub-workflow tree generated for the sample case in Fig. 1. As the hierarchical structure of the anterior workflow version may differ from the structure of the posterior version, two trees $T_a$, $T_p$ have to be generated initially. The two constructed trees will be merged at the end in order to get one sub-workflow tree containing all atomic edit operations between the two versions. In depth-first search, the nodes of $T_a$ are successively enriched by add and delete lists for each type of workflow element at the level of the actually investigated sub-workflow. The set difference between the set of data objects for workflow inputs of a sub-workflow $X$ of the anterior version and the set of data objects for inputs of the same sub-workflow $X'$ of the posterior version, for instance, forms the according delete list $X.DEL\_inputDataObjects$. The 'same' sub-workflow means that both sub-workflows have equal names and have the same position in $T_a$ and $T_p$ with respect to the path from the root node. The set $X' \setminus X$ forms the corresponding add list. A sub-workflow that has been added entirely is stored by the placeholder task in the according list for placeholder tasks as well as by an additional sub-workflow node in $T_a$. The sub-workflow node is enriched by add and delete lists for the inner elements of the sub-workflow. Where required, further sub-workflow nodes are inserted into $T_a$ below the new node. From the resulting, fully expanded and enriched sub-workflow tree $T'_a$ all edit operations that have taken place can be reproduced. The workflow elements and their position within the hierarchy of sub-workflows is recorded unambigously except for the order of sibling sub-workflow nodes. The latter is not significant as the dependencies between tasks and objects are specified explicitely by edges in the formal representation (data links and coordination constraints).

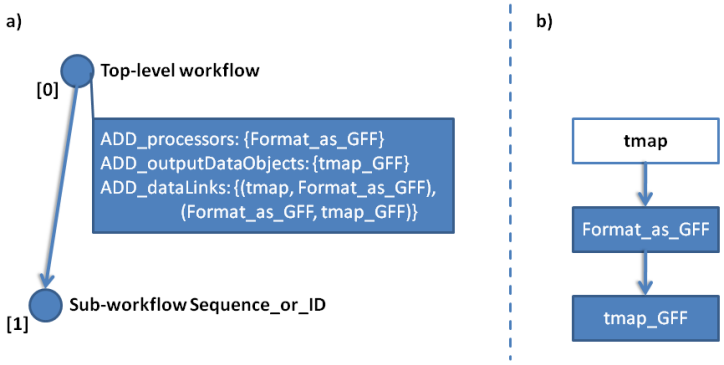



**Fig. 2.** Sub-workflow tree and add graph generated for the sample case in Fig. 1.

Though the sub-workflow tree is capable for a reconstruction of the change, it alone is not sufficient for a transfer of the change to another target workflow. Rather than applying the edit operations directly to the target workflow, the connected workflow elements should be grouped in order to preserve the connectivity and to enable the application of a chain of edit operations 'fully or not at all'. The anchor principle as described in Sect. 2 comes into play for mapping the positions of the chains. Thus, the third step of the automatic case acquisition is to group the atomic edit operations in chains and to determine appropriate anchors for each chain. This is done by constructing maximum connected sub-graphs of the workflow graph consisting of workflow elements conjointly affected by the same type of atomic edit operation, the so-called *add graphs* and *delete graphs*. Fig. 2 b) depicts a sample add graph. An add graph consists of added data objects and tasks including placeholder tasks as nodes and added data links and coordination constraints as directed edges. As a graph can only contain edges with a source and a sink, the workflow elements that are source of an added data link or coordination constraint are included as nodes in the add graph also if they are not added themselves. The nodes of this particular set are designated as the pre anchors of the chain. The same holds for workflow elements that are sink of an added link or coordination constraint. The set of those particular nodes forms the post anchors of the chain. The delete graphs are built analogously. Add and delete graphs may span several sub-workflows.

**Algorithm 1 Build add graphs**
**input**

```
   Sub-workflow tree t (with sub-workflow nodes numbered in the
      order of a breadth first search from 0 to sizeOfTree)
```

**output**

```
   Forest addGraphs
```

**begin**

```
   dispoN[sizeOfTree]:= array of lists of workflow elements;
   dispoE[sizeOfTree]:= array of lists of workflow elements;
   anCands[sizeOfTree]:= array of lists of workflow elements;
   addTrees:=∅;
   foreach i=0..sizeOfTree do
      dispoN[i]:=unification of all add lists of type task, data
         input object, data output object, or placeholder task
         of i-th (sub-)workflow node;
      dispoE[i]:= unification of all add lists of type data link
         or coordination constraint of i-th (sub-)workflow;
      anCands[i]:= unification of all tasks, data input objects,
         data output objects and placeholder tasks of i-th (sub-)
         workflow except the elements of any add or delete list;
```

```
21      od

22      foreach i=0..sizeOfTree do

23          addGraphs:=addGraphs ∪
24             createMaxConGraphs(i,dispoN,dispoE,anCands);

25      od

26      return addGraphs;

27  end
```

```
28  function graphSet createMaxConGraphs(i,dispoN,dispoE,anCands)
29      graphSet:=∅;
30      while dispoN[i]!=∅ do
31          openPH:=∅;
32          currentN:=dispoN[i].firstEl;
33          dispoN[i].delete(currentN);
34          currentG:={currentN};
35          if currentN is placeholder do
36              openPH.add(currentN);
37          od
38          openGE:={currentN};
39          while openGE!=∅ do
40              currentEl:=openGE.firstEl;
41              openGE.delete(currentEl);
42              if currentEl is node do
43                expandE(currentG,i,openGE, dispoE,currentEl);
44              else
45                expandN(currentG,i,openPH,openGE,dispoN,anCands,currentEl);
46              od
47          od
48          while openPH!=∅ do
49              currentPH:=openPlaceHolder.firstEl;
50              expandSFW(currentG,currentPH,openPH,dispoE,dispoN);
51              openPH.delete(currentPH);
52          od
53          graphSet.add(currentG);
54      od
55      while dispoE[i]!=∅ do
56          currentEdge:=dispoE[i].firstEl;
57          dispoE[i].delete(currentEdge);
58          currentG:={currentEdge};
59          expandN(currentG,i,openPH,openGE,dispoN,anCands,currentEl);
60          graphSet.add(currentG);
61      od
62      return graphSet;

63  function expandE(currentG,i,openGE,dispoE,currentEl)
64      foreach j=0..dispoE[i].size do
65          currentEdge:=dispoE[i][j];
66          if currentEdge touches currentEl do
67              openGE.add(currentEdge);
68              currentG.add(currentEdge);
69              dispoE[i].delete(currentEdge);
70          od
71      od
72      openGE.delete(currentEl);
```

```
73      function expandN(currentG,i,openPH,openGE,dispoN,anCands,currentEl)
74          expanded:=false;
75          foreach j=0..dispoN[i].size do
76              currentN:=dispoN[i][j];
77              if currentN touches currentEl do
78                  openGE.add(currentN);
79                  currentG.add(currentN);
80                  dispoN[i].delete(currentN);
81                  if currentN is placeholder do openPH.add(currentN);
82                  od
83                  expanded:=true;
84              od
85          od
86          if !expanded do
87              foreach j=0..anCands[i] do
88                  currentN:=anCands[i][j];
89                  if currentN touches currentEl do
90                    currentG.addAnchor(currentN);
91                  od
92              od
93          od
94          openGE.delete(currentEl);
95      function expandSWF(currentG,currentPH,openPH,dispoE,dispoN)
96          openGE:=∅;
97          foreach el ∈ currentG do
98              if el touches currentPH do
99                  openGE.add(el);
100             od
101         od
102         while openGE!=∅ do
103             currentEl:=openGE.firstEl;
104             openGE.delete(currentEl);
105             if currentEl is node do
106                 expandE(currentG,currentPH.index, openGE,dispoE,currentEl);
107             else
108                 expandN(currentG,currentPH.index,openPH,openGE,dispoN,
109                    anCands,currentEl);
110         od
111     od
```

Algorithm 1 details the steps of building add graphs. The input is a sub-workflow tree including the atomic edit operations that is built from the two workflow versions as described above. The algorithm passes through the sub-workflow graph in a breadth first search to start constructing the add graphs at each sub-workflow level by the function `createMaxConGraphs`. The function creates and extends one sub-graph after the other employing an 'open' list of

graph elements (nodes and edges), which is a well known principle from search algorithms in artificial intelligence. If the recent add graph is extended by a workflow element the element is moved from the list of disposable elements (nodes `dispoN` and edges `dispoE`) to the list of open graph elements (`openGE`). If the element has been fully expanded in the sub-graph, i.e. if all of its touching edges (or nodes, in case of an edge) from the lists of disposable elements have been investigated, it is deleted from the list of open graph elements. The algorithm starts with the initialization of the dispo lists at each sub-workflow level (lines 7, 8, and 12 – 16 of Alg. 1). The workflow nodes that are not part of any add or delete list are stored in the `anCands` lists as they might serve as anchors (lines 9 and 17 – 19). For each node of the sub-workflow tree, the maximum connected graphs are computed by calling `createMaxConGraphs`. Within this function, an additional recursive decent into lower sub-workflow levels is required if an add graph extends into lower-level sub-workflows. However, the creation of an add graph is always started at the higest possible level. At one level, several distinct sub-workflows may start. An add graph starts with one node as initial graph (l. 34) and is extended by edges (l. 43) and nodes (l. 46) from the disponable elements. `openGE` records all graph elements (nodes and edges) of an add graph that are still to be expanded (line 38 and within the functions `expandE` and `expandN`). Placeholders are recorded in `openPH` (lines 35 – 37 and within the function `expandN`) in order to expand the add graph further in function `expandSFW` when all elements at the same level have been investigated (lines 47 – 52). After all initial graphs have been expanded, some edges might be left in openEdges. They are expanded by anchor nodes (lines 59 – 66) to form chains with one element only.

## 5 Experimental Results

The hypotheses posed at the beginning of this paper have been tested by experiments with scientific workflows retrieved from the community repository at `www.myexperiment.org`. Hypothesis I on the applicability of case-based adaptation methods for scientific workflows has been split into two parts investigated by manual experiments, while hypothesis II on the automated construction of adaptation cases has been investigated by comparing the results of an implementation of Algorithm 1 with the manual results gained from the experiments on hypothesis I. The following questions guided the experiments:

(Ia) *Applicability*: Can adaptation cases with chains and anchors be constructed that record the adaptation episodes of scientific workflows?
(Ib) *Applicability*: Does the reuse of the cases lead to feasible results?
(II) *Automated construction*: Can adaptation cases be captured automatically from community repositories? Is the quality of the automated results comparable to the quality of reference results acquired by experts?

We started to manually create a case base with eleven cases including the sample case in Fig. 1. The xml files retrieved from the repository only consist of

atomic elements such as tasks and data objects and relational link elements, so we would have to draw the workflows manually to understand all dependencies. For that reason we used as additional source for the creation the automatically rendered representation of workflow versions because it would have been a very tedious work to compare the xml files without further utilities. Although this seems to be a quite small number of cases for a case base it took a large part of time of this work. This is purely owed the complexity of scientific workflows. We ensured that the workflows in our adaptation cases use all workflow elements from the Scufle modeling language. In some adaptation cases we have chains with more than 45 conjoined workflows elements which are connected over several hierarchical levels. In such cases it is difficult for a human to follow all links of a complex workflow bi-directional. Nevertheless the eleven adaptation cases could be created successfully. This provides a reference solution for the computed cases in (II) and confirms hypothesis (Ia).

Hypothesis (Ib) has been investigated by grab samples only. One sample target workflow is depicted in Fig. 3 which contains the same tmap Soaplab service as the anterior workflow of our sample adaptation case in Fig. 1. Hence, the change described by the adaptation case could be transferred to the sample target workflow. Admittedly, the grap samples give a first hint only that the automated adaptation would work in principle. Further evidence is required from experiments that will be part of our future work. The next step in our work is to test if the chains in the automatically acquired cases can be used to reconstruct the given cases. For this purpose we will investigate whether the mapping of anchors [10] can be applied to scientific workflows too. If these tests will be successful we will be able to create a large case base with reusable adaptation cases and conduct further experiments on re-purposing the adaptation episodes on other target workflows.

For hypothesis (II), we implemented Algorithm 1 in Java. Running the algorithm lead to slightly different results from the reference solution. Only six cases were identical to the reference solution. Surprisingly, four of the other five cases showed up small failures in the reference solution. The algorithm can also detect port changes in a link between data objects and tasks. Because these details are not illustrated in the graphical representation as described in Sect.2 it is hardly possible to determine such changes manually. Other failures that orrcurred in the reference solution chains are miss-spellings of element names that have been overseen (e.g. 'organsim' instead of 'organism') and small changes between workflow versions which only lead to very small chains in a case. The anchors of the reference solution only differred in one case from the automatically created solution, but also in this case it was due to a human error that an anchor was overseen in the reference solution. Only in one case a minor deviation from the 'optimal' solution occurred: Changes crossing a sub-workflow by a continuous path but not affecting the entire sub-workflow led to the effect that the placeholder task for this sub-workflow did not appear within the change lists. As a consequence, the algorithm divided the edit chain into two chains, while the reference solution modeled one chain only. In order to deal with such special cases,

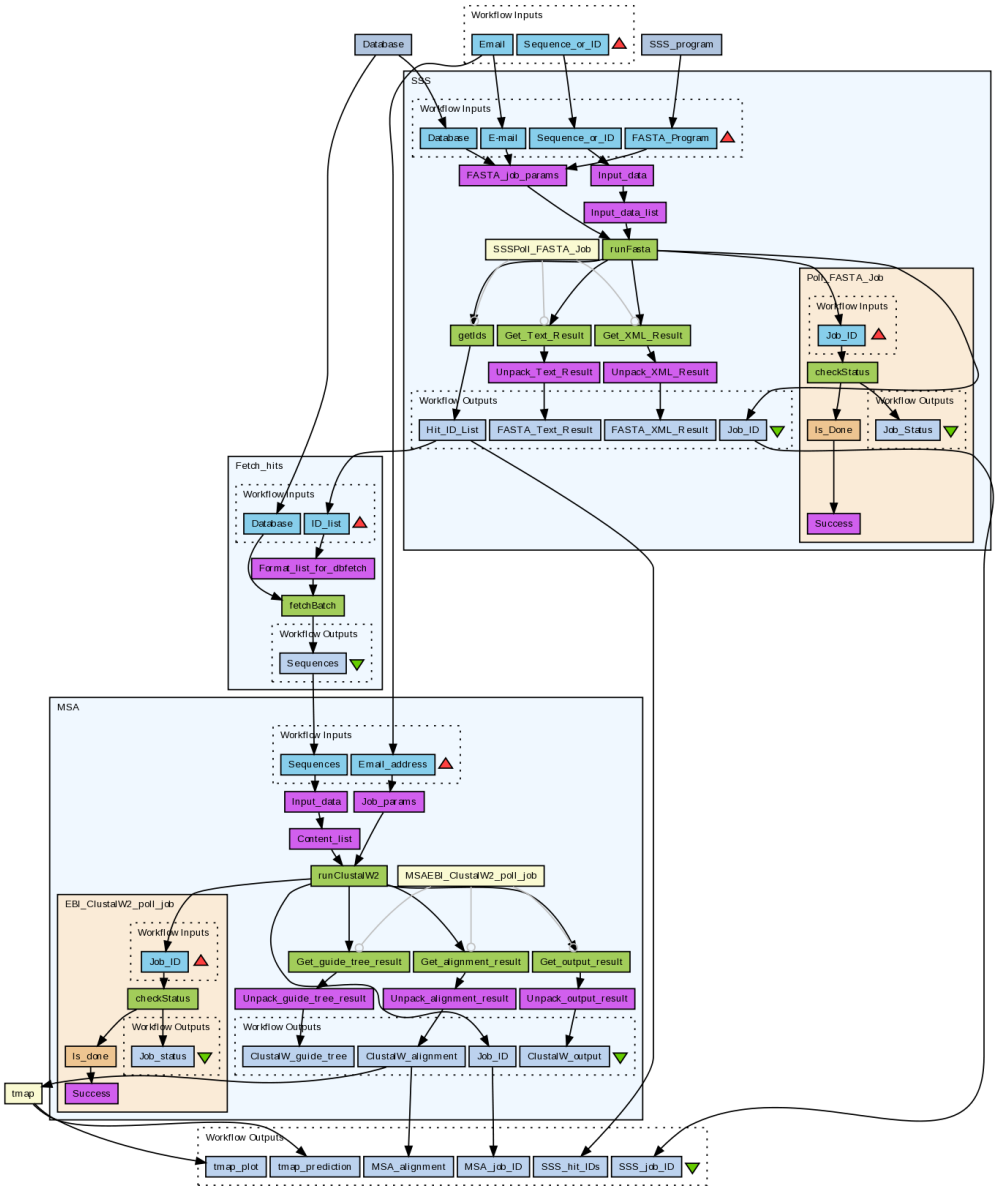



**Fig. 3.** Sample target workflow retrieved from `www.myexperiment.org` for that the case from Fig. 1 would be applicable.

the algorithm could be extended by unifying chains that contain paths through a sub-workflow. As a result from the automatic acquisition of adaptation cases we got the experimental confirmation that the algorithm works very well. It is evident that the manual acquisition is not only time consuming but also fault-prone for small changes in workflow versions. Concluding we regard hypothesis II to be confirmed by the experimental results.

## 6 Related Work

Related work from the field of scientific workflow reasoning and from the field of case-based workflow adaptation will be discussed in the following. Goderis [4] deals with the discovery of scientific workflows and workflow elements by techniques of information retrieval, ontology reasoning, and graph matching. Workflow adaptation is not addressed. Gil [2] develops a vision of a knowledge level view on workflows for reasoning tasks like workflow generation and validation, automated data/parameter selection or metadata generation. The workflow system Wings assists scientists in some of these tasks by means of semantic metadata. Adaptation support is not mentioned but is very closed to workflow generation. The following approaches as well as our own previous work [11] employ case-based methods to give adaptation advice to the users. Leake and Morwick [7] provide case-based support for workflow generation by evaluating the execution traces of scientific workflows. Weber et al. [15] employ conversational CBR for the adaptation of health workflows. Montani and Leonardi [12] focus on case-based workflow monitoring based on execution traces also in the health domain. Kapetanakis et al. [6] employ case-based workflow monitoring by means of temporal relationships in the area of business workflows. In our recent previous work [10], we extend the scope towards automated adaptation of workflows. In contrast to this work, the adaptation knowledge is still captured by experts.

## 7 Conclusion

In this paper, the automated acquisition of adaptation cases from community repositories of scientific workflows has been investigated. First, the work resulted in the confirmation that case-based adaptation methods developed for workflow reasoning on business workflows can be transferred to scientific workflows in principle. Data dependencies can be included into the methods that have been control flow-oriented so far. However, details of data dependencies like the data ports of the concerned Web services have not yet been addressed by the solution provided so far. Future work on the internals of workflow elements could close this gap. Second, an algorithm for the automated acquisition of adaptation cases led to feasible experimental results. The overall outcome of this work is promising for an application of the case-based workflow adaptation methods to further fields like workflows in computer games[14] or personal workflows[5].

## References


1. J. Becker, P. Bergener, D. Breuker, and M. Räckers. On measures of behavioral distance between business processes. In A. Bernstein and G. Schwabe, editors, *Proceedings of the 10th International Conference on Wirtschaftsinformatik*, volume Vol. 2, pages 665–674. 2011.
2. Y. Gil. From data to knowledge to discoveries: Artificial intelligence and scientific workflows. *Scientific Programming*, 17(3):231–246, 2009.
3. Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24 – 32, 2007.
4. A. Goderis. *Workflow Re-use and Discovery in Bioinformatics*. PhD thesis, University of Manchester, 2008.
5. S. Y. Hwang and Y. F. Chen. Personal workflows: Modeling and management. In *Mobile Data Management*, page 141–152, 2003.
6. S. Kapetanakis, M. Petridis, B. Knight, J. Ma, and L. Bacon. A case based reasoning approach for the monitoring of business workflows. *Case-Based Reasoning. Research and Development*, page 390–405, 2010.
7. D. B. Leake and J. Kendall-Morwick. Towards Case-Based support for e-Science workflow generation by mining provenance. In *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings*, pages 269–283, 2008.
8. B. Ludäscher, M. Weske, T. McPhillips, and S. Bowers. Scientific Workflows: Business as Usual? In U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, editors, *Proceedings BPM 2009, LNCS 5701*, pages 31 – 47, Heidelberg, 2009. Springer.
9. T. Madhusudan, J. L. Zhao, and B. Marshall. A case-based reasoning framework for workflow model management. *Data & Knowledge Engineering*, 50(1):87–115, 2004.
10. M. Minor, R. Bergmann, S. G\örg, and K. Walter. Towards Case-Based adaptation of workflows. In *Case-Based Reasoning. Research and Development*, page 421–435, 2010.
11. M. Minor, A. Tartakovski, and R. Bergmann. Representation and structure-based similarity assessment for agile workflows. In *Case-Based Reasoning Research and Development*, page 224–238, 2007.
12. S. Montani and G. Leonardi. A Case-Based approach to business process monitoring. *Artificial Intelligence in Theory and Practice III*, 331/2010:101–110, 2010.
13. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045 –3054, Nov. 2004.
14. A. Ram, S. O. nón, and M. Mehta. Artificial intelligence for adaptive computer games. In *Twentieth International FLAIRS Conference on Artificial Intelligence*, 2007.
15. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In P. Funk and P. A. González-Calero, editors, *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, volume 3155 of *Lecture Notes in Computer Science*, pages 434 – 448. Springer, 2004.