Representation and Structure-Based Similarity Assessment for Agile Workflows

Mirjam Minor, Alexander Tartakovski, and Ralph Bergmann

University of Trier
Department of Business Information Systems II
D-54286 Trier, Germany
minor@uni-trier.de
www.wi2.uni-trier.de

Abstract. The dynamics of the market requires workflow management systems that support agile workflows - workflows which are flexible concerning the adaptation to innovations. This paper presents a case-based approach to representation and index-based retrieval of past workflows in order to give authoring support for adaptation of recent workflow instances. The utility of the presented methods is demonstrated by an experimental evaluation.

1 Introduction

Thomas Herrmann reports the observation that many collaborative tasks in companies can be partly seen as recurrent routines but partly to contain innovation. ... This phenomenon will increase with the dynamics of the market and its requirements to the flexibility of the company and to the individual customer care." [1, p. 145, own translation]. Traditional workflow systems are able to support the recurrent tasks quite well. In order to deal with the flexible, innovative part, the workflows have to be adaptable to the innovation. Moreover, in highly flexible domains like medicine or chip design, situations occur where the ongoing workflows need to be changed. For instance, an alternative course of action has to be taken when a certain therapy is not successful for a patient or when a certain algorithm does not work for a new chip technology. This is not possible with traditional workflow systems.

Case-Based Reasoning (CBR) is a quite natural approach to support the flexibility of workflows. Experience from the adaptation of workflows in the past can be reused for the adaptation of an ongoing workflow. A case base contains past workflows in a certain state of execution together with the subsequent workflow modifications. When a current workflow has to be adapted it can be used as a query to the case base. Modifications of similar workflows from the case base can be reused in order to change the current workflow. In this paper, we present a new representation formalism for agile workflows [2] as well as a retrieval approach based on graph edit distances [3] that operates directly on the workflow structure. We show in some experiments that our approach is suitable for this kind of workflow.

R.O. Weber and M.M. Richter (Eds.): ICCBR 2007, LNAI 4626, pp. 224–238, 2007.

In the literature, a number of approaches for agile workflows exist that require further information in addition to the workflow structure such as context information [4,5] or conversational knowledge [6]. However, this information is not always available and can be processed automatically only with considerable effort. Furthermore, there is an approach that is related to our approach as it operates directly on the workflow structure: Luo et al. [7] have developed a building block similarity for traditional workflows. Unfortunately, this method is not suitable for changes of the order of workflow elements which are typical for agile workflows. Minor changes, for instance, moving a task to a different block leads to major restructuring activities within the building block tree and consequently seems to impact the similarity values to an excessive degree.

The remainder of this paper is organized as follows: Section 2 provides an introduction to agile workflows. In Sect. 3 we present a novel approach to representation and index-based retrieval of agile workflows. Section 4 provides an evaluation of our methods, while Sect. 5 concludes this paper with a discussion and an outlook.

2 Agile Workflows

In the following, we give an introduction to agile workflows [2] for which we have developed a retrieval approach based on graph edit distances. Agile workflows allow the incremental and flexible modeling of processes. Initial workflow instances are derived from a set of templates called workflow definitions. The instances can be adapted during the ongoing process. The term 'agile workflows' neither covers work on process mining [8] nor Herrmann's [1] approach to learning workflows from sets of loosely coupled tasks. We call these kinds of workflows 'emergent' rather than 'agile'. There is a small research community on agile workflow technology whose work we classify according to three types of process changes at run time:

- Ad-hoc changes that apply to individual workflow instances only [9,6],
- Modifications to a worflow definition that is already in use by instances [9], and
- Late-planning and hierarchical decomposition [4,5].

Our work fits in the first and third classifications.

Figures 1 and 2 show two UML activitiy diagrams of sample workflow definitions that we modeled for the chip design domain in order to support ad-hoc changes and late-planning. Each workflow consists of a control flow structure of tasks and of a context model. The context is described by a set of context factors with default values [10].

The control flow structure follows the design flow 'SciWay 2.0', i.e. a standardized description of the step by step design process for all digital design projects of our industrial partner Silicon Images GmbH (formerly sci-worx). The language to describe the control flow is based on the notation of workflow patterns introduced by van Aalst et al [11]. Workflow patterns "address business requirements in an imperative workflow style expression" [11, p. 4]; broadly speaking,

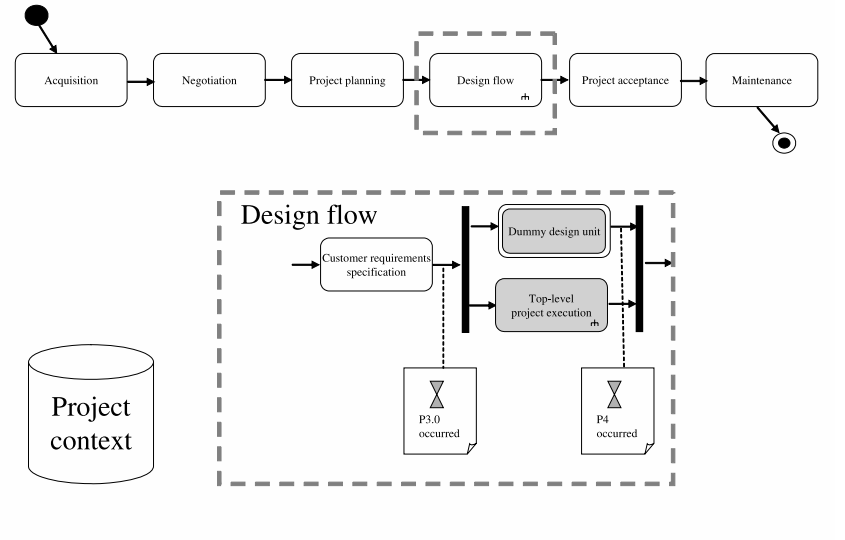


Fig. 1. The workflow definition of a design project following SciWay 2.0

they are useful routing constructs within workflows. In terms of van Aalst et al, our workflow modelling language consists of the five basic control flow elements (workflow patterns) *sequence*, *AND-split*, *AND-join*, *XOR-split*, and *XOR-join*, as well as loops. We regard loops as structured cycles with one entry point to the loop (LOOP-join) and one exit point from the loop (LOOP-split). A diamond with an '[L', one incoming and several outgoing arrows with conditions in squared brackets stands for the LOOP-split; a diamond with an 'L]', several incoming and one outgoing arrows stands for the LOOP-join (see Fig. 2). Loops cannot be interleaved but they can be nested, i.e. an inner loop may be set into one or several outer loops. For reasons of adaptability, we have extended this modelling language by three own workflow elements: (1) placeholder tasks for sub-workflows are depicted as rounded boxes with double borders (see 'Dummy design unit' in Fig. 1); (2) placeholder tasks for sub-diagrams are marked by a fork symbol (see the placeholder task for 'Design flow' in Fig. 1); (3) breakpoints are symbolized by stop signs (see Fig. 5). Sub-diagrams have only been introduced for reasons of clarity. In contrast to sub-workflows, sub-diagrams do not have an own workflow enactment service nor an own context model. Breakpoints are necessary for the implementation of long-term workflows. Decisions about how to modify a workflow region may take considerable time; setting a breakpoint prevents the workflow engine from overrunning tasks that are about to be modified.

Figure 3 shows a sample workflow instance that has been modified by late-planning. In comparison with the workflow definition in Fig. 1, the sub-workflow placeholder 'Dummy design unit' has been replaced by three sub-workflow placeholders for real design units. This has been done by the task 'Project planning'. Figure 4 expands the sub-workflow instance of the design unit '10a' which has

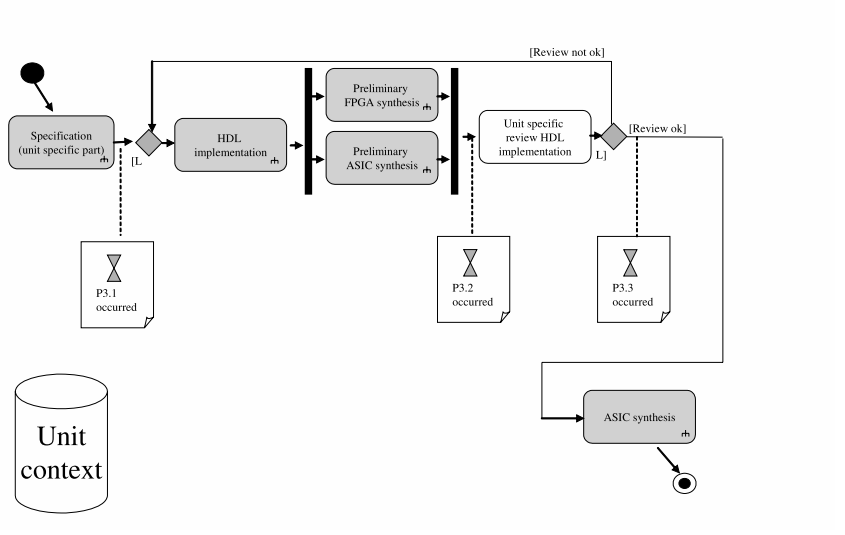


Fig. 2. The workflow definition of a design unit following SciWay 2.0

been derived from the template in Fig. 2. In addition to the workflow definition, it has the task 'Check whether feature set confirmed'. Figure 5 shows a further revision of this case that includes the implementation of additional features in hardware description language (HDL). This has been driven by a change request from the customer in a late project phase.

3 Representation and Retrieval of Workflow Instances

According to the above sample workflow instances, the representation has two parts: one for the control flow structure of tasks and another one for the context model.

The context is represented by a structural CBR approach with attribute-value pairs in a straightforward way. The representation of workflow structure makes use of the fact that the instances are derived from a particular workflow definition. As the instances usually differ only slightly from their templates, they can be described by means of the difference to their workflow definition.

A workflow definition is represented as a set of elements, such as tasks and control flow elements, as well as a successor-predecessor relation on this set. The difference between an ongoing instance and its workflow definition covers the following issues:

1. the structural modifications of tasks and control flow elements
2. the state of processing

Both can be encoded by sets of added and deleted workflow elements with respect to the original template. Hereby, completed tasks as well as passed control flow elements are regarded as deleted.

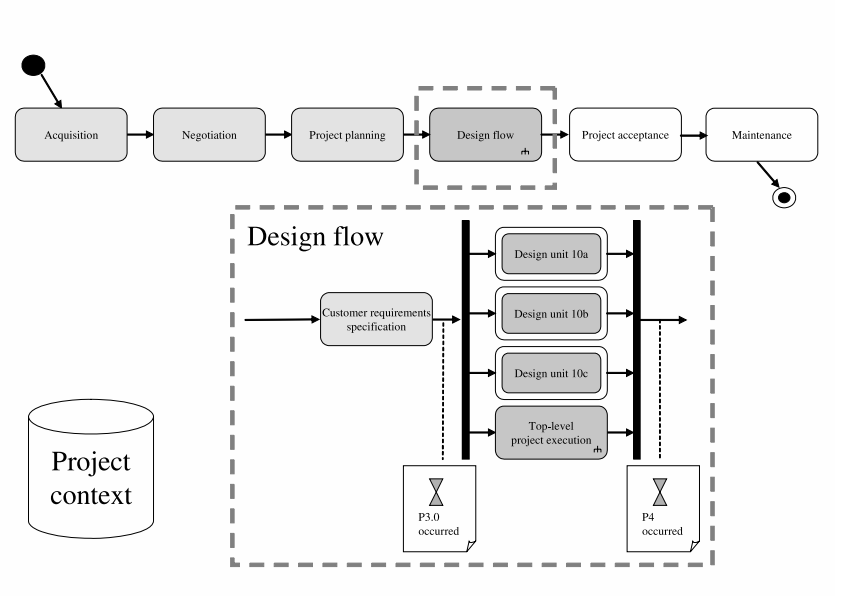


Fig. 3. Sample workflow instance of a design project

The experience that is contained in an ongoing workflow instance and the changes applied to it can be captured within cases according to the CBR approach. A case consists of a pair of subsequent revisions of a workflow instance $[X, X']$ (compare the two revisions in Figures 4 and 5). The previous revision X is the problem part of the case; X' is the solution part of the case.

3.1 Similarity Assessment and Index-Based Retrieval

The main challenge for the development of a similarity measure for agile workflows is comparing the structure of workflows. The comparison of context models can be realized according to the local-global-principle of the structural CBR approach. The similarity value for the context part is aggregated with the value for the structure part to an overall similarity value.

On the one hand the comparison of workflow structure should be kept computationally efficient and on the other hand the measure has to approximate the usability sufficiently well.

In the literature, several approaches have been developed for similarity assessment between graphs [12], among them graph matching measures and graph edit distance measures. To the first group belong measures which are based on such characteristics as "graph isomorphis" [13,14], "sub-graph isomorphism" [15], and "largest common sub-graph" [16,17]. To the second group belong algorithms dealing with graph edit distance, e.g. "weighted graph edit distance" [3].

For similarity assessment in our system we have chosen the idea of the weighted graph edit distance. The workflow definition (template) can be used to

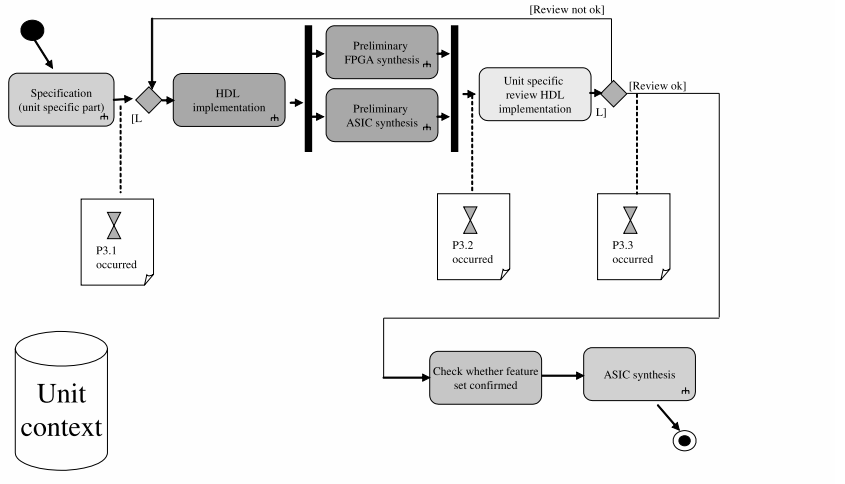


Fig. 4. Sub-workflow instance 'design unit 10a' from Fig. 3

accelerate the similarity assessment. However, this leads to completely different algorithms than those described in the literature.

Bunke and Messmer's [3] measure generalizes the string edit distance [18]. It is defined for attributed directed graphs but can be easily applied in a simplified form to standard graphs as well. Similarity is modeled through a set of edit operations on graphs. Each edit operation e transforms a graph into a successor graph performing a modification of the following kind: insert a new node or a new edge, delete a node or an edge, change a node or an edge label. Each edit operation has assigned a certain cost $c(e) \in [0, 1]$. A difference can be defined based on the total cost of a sequence of edit operations which transform one graph into the other graph. The cheaper and the fewer the operations are that are required to transform a graph into another the smaller is the difference and hence the higher is the similarity between the two of them. These considerations lead to the following difference function:

$$\delta(x, y) = \min\{\sum_{i=1}^{k} c(e_i) \mid (e_1, \ldots, e_k) \,\text{transforms}\, x \,\text{to}\, y\} \tag{1}$$

The computation of the graph edit distance measure is an NP-complete [3] problem and can be performed by a state-space search, e.g. by an A* algorithm. Hence, this similarity measure should be used quite carefully.

Our similarity measure for the structure of workflows will be explained in the two following sections. While the first section presents the similarity assessment only for restricted workflows, the second section presents an extension to this measure which can be applied to workflows with arbitrary tasks and control flow elements as well.

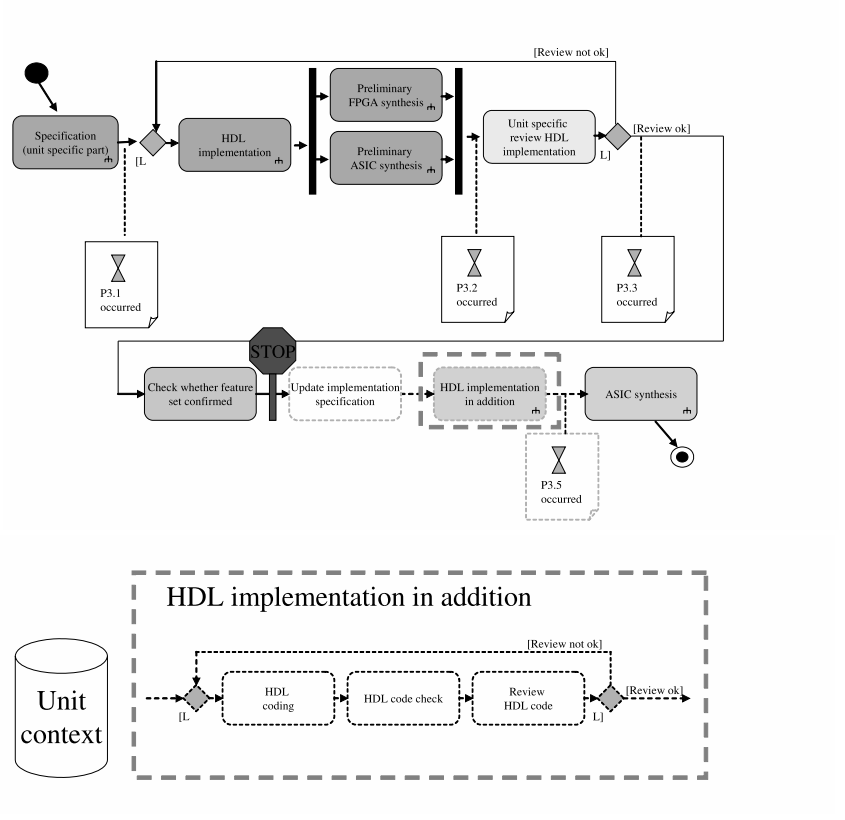


Fig. 5. Late revision of sub-workflow instance 'design unit 10a' from Fig. 3

3.2 Similarity Measure for Restricted Workflows

This section regards similarity assessment for restricted workflows that contain arbitrary tasks as well as control flow elements only of the type "sequence". For the purpose of similarity assessment an abstract view on workflows will be defined. It includes only tasks, names of tasks, and ordering on tasks, given through control flow elements of the type "sequence". The view can be represented as a directed and attributed graph:

$$View =< N, E, name > \tag{2}$$

The nodes N in this graph represent workflows' tasks and the edges E represent the control flow elements of type "sequence". Furthermore, every node is labelled by the name of a respective task:

$$name : N \rightarrow TaskNames \tag{3}$$

There are two important characteristics of workflow instances that allow an efficient computation of the graph edit distance $\delta(V_1, V_2)$ between two arbitrary

views $V_1 =< N_1, E_1, name_1 >$ and $V_2 =< N_2, E_2, name_2 >$. The first characteristic is that the name of every task is unique within a single workflow instance. The second characteristic is that two tasks, T_1 from one workflow and T_2 from another workflow, can be seen as identical if and only if their names are equal. This leads to following definitions:

$$\begin{aligned}
&\text{Nodes within } V_1 \text{ but not within } V_2 : \hat{N_1} := N_1 \setminus N_2 \\
&\text{Nodes within } V_2 \text{ but not within } V_1 : \hat{N_2} := N_2 \setminus N_1 \\
&\text{Edges within } E_1 \text{ but not within } E_2 : \hat{E_1} := E_1 \setminus E_2 \\
&\text{Edges within } E_2 \text{ but not within } E_1 : \hat{E_2} := E_2 \setminus E_1
\end{aligned} \tag{4}$$

Two nodes $n_1 \in N_1$ and $n_2 \in N_2$ are defined to be equal if and only if their labels are equal: $name(n_1) = name(n_2)$. Two edges $e_1 \in E_1$ and $e_2 \in E_2$ are defined to be equal if and only if $name(predecessor(e_1)) = name(predecessor(e_2))$ and $name(successor(e_1)) = name(successor(e_2))$.

We can now define the distance $\delta(V_1, V_2)$ between the views V_1 and V_2. Suppose, we are going to edit the view V_1 until it is equal to V_2. For this purpose the nodes $\hat{N_1}$ have to be deleted from V_1, since they are not in V_2. The number of edit operations is $|\hat{N_1}|$. Then the edges $\hat{E_1}$ have to be deleted for the same reason. The number of edit operations is $|\hat{E_1}|$. The sets $\hat{N_2}$ and $\hat{E_2}$ have to be added to the view V_1, since the nodes and edges are within V_2, but not within V_1. The number of operations is $|\hat{N_2}| + |\hat{E_2}|$. The overall sum of edit operations is $|\hat{N_1}| + |\hat{E_1}| + |\hat{N_2}| + |\hat{E_2}|$. It can be simply proven that this number of edit operations is minimal. Therefore the distance is set to:

$$\delta(V_1, V_2) = |\hat{N_1}| + |\hat{E_1}| + |\hat{N_2}| + |\hat{E_2}| \tag{5}$$

It should be mentioned that for this special case the complexity of the distance assessment is not exponential but quadratic. However, the average complexity could be further improved. The improvement is based on the fact that instances to be compared are created starting from the same workflow definition and differ only slightly from their template (with a view $V_T =< N_T, E_T >$). Therefore the respective views V_1 and V_2 can be redefined as follows:

$$\begin{aligned}
V_1 &=< N_T \cup add.nodes_{V_1} \setminus delete.nodes_{V_1}, E_T \cup add.edges_{V_1} \setminus delete.edges_{V_1} > \\
V_2 &=< N_T \cup add.nodes_{V_2} \setminus delete.nodes_{V_2}, E_T \cup add.edges_{V_2} \setminus delete.edges_{V_2} >
\end{aligned} \tag{6}$$

Hereby the set $add.nodes_{V_1}$ defines nodes that should be added to the workflow definition in order to get the view V_1. The set of nodes $delete.nodes_{V_1}$ should be deleted from V_T. The sets $add.edges_{V_1}$ and $delete.edges_{V_1}$ have the same semantics but the objects to be altered are edges. The same consideration can be carried out for the view V_2. Now the sets $\hat{N_1}, \hat{E_1}, \hat{N_2}, \hat{E_2}$ can be redefined.

$$\begin{aligned}
\hat{N_1} &:= \{N_T \cup add.nodes_{V_1} \setminus delete.nodes_{V_1}\} \setminus \\
&\quad \{N_T \cup add.nodes_{V_2} \setminus delete.nodes_{V_2}\} \\
\hat{N_2} &:= \{N_T \cup add.nodes_{V_2} \setminus delete.nodes_{V_2}\} \setminus \\
&\quad \{N_T \cup add.nodes_{V_1} \setminus delete.nodes_{V_1}\} \\
\hat{E_1} &:= \{E_T \cup add.edges_{E_1} \setminus delete.edges_{E_1}\} \setminus \\
&\quad \{E_T \cup add.edges_{E_2} \setminus delete.edges_{E_2}\} \\
\hat{E_2} &:= \{E_T \cup add.edges_{E_2} \setminus delete.edges_{E_2}\} \setminus \\
&\quad \{E_T \cup add.edges_{E_1} \setminus delete.edges_{E_1}\}
\end{aligned} \tag{7}$$

Using results of the set theory the edit distance can be transformed to the following formula:

$$\begin{aligned}
&\delta(V_1, V_2) = |\hat{N_1}| + |\hat{E_1}| + |\hat{N_2}| + |\hat{E_2}| = \\
&\quad |\{delete.nodes_{V_1} \cup delete.nodes_{V_2}\} \setminus \{delete.nodes_{V_1} \cap delete.nodes_{V_2}\}| + \\
&\quad\quad |\{add.nodes_{V_1} \cup add.nodes_{V_2}\} \setminus \{add.nodes_{V_1} \cap add.nodes_{V_2}\}| + \\
&\quad\quad |\{delete.edges_{V_1} \cup delete.edges_{V_2}\} \setminus \{delete.edges_{V_1} \cap delete.edges_{V_2}\}| + \\
&\quad\quad\quad |\{add.edges_{V_1} \cup add.edges_{V_2}\} \setminus \{add.edges_{V_1} \cap add.edges_{V_2}\}|
\end{aligned} \tag{8}$$

Since the sets *add.nodes* and *del.nodes* become available with the construction of instances that starts from templates and since it normally has a low cardinality the computation time of the edit distance decreases significantly. The sets *add.nodes* and *del.nodes* can be understood as indexes.
Finally, the distance can be normalized and transformed to the compatible similarity measure with a range $[0, 1]$, e.g.:

$$sim(V_1, V_2) := 1 - \frac{\delta(V_1, V_2)}{|N_1| + |N_2| + |E_1| + |E_2|} \tag{9}$$

This similarity measure can be enriched by the weights in order to emphasize some types of edit operations.

3.3 Similarity Measure for Workflows with Control Flow Elements

The distance measure introduced in the previous section does not support flow elements, such as AND-split, AND-join, XOR-split, XOR-join, and so on. However, taking them into consideration improves the approximation of usability (see Sect. 4).

The consideration of the flow elements in the similarity function entails several challenges. Contrary to tasks, which are unique within workflow instances and which could be identified by unique names, control flow elements do not have unique names and often occur several times within an instance. Because of this circumstance the computation of an exact edit distance becomes computationally more expensive. Therefore we regarded several approximation methods and evaluated the usability of the result sets empirically.

Approximation Method 1. The first approach supports workflows containing arbitrary control flow elements. However, it doesn't take the semantics of the control flow elements into account while computing the similarity value. The main idea of this straightforward approach is to represent every control flow element through one or several edges within a view. For this purpose every two tasks which are directly connected through control flow elements will be transformed to two nodes and one edge between them in the view. The "direct connection" means that there is a path in the workflows' structure connecting these tasks and this path does not contain any further tasks (but one or more control flow elements between them are allowed). E.g. regard two paths $(T_1, AND-split, T_2)$ and $(T_1, AND-split, T_3)$ within workflow instance $T_1 \rightarrow \begin{matrix} \nearrow T_2 \\ \searrow T_3 \end{matrix}$. The tasks T_1, T_2, and T_3 will be converted to nodes N_{T_1}, N_{T_2}, and N_{T_3} in each respective view. The control flow element will be substituted through two edges $e_1 = (N_{T_1}, N_{T_2})$ and $e_2 = (N_{T_1}, N_{T_3})$. The similarity assessment can then be carried out in the same way as presented in Sect. 3.2).

Approximation Method 2. The second approach is an extension of the first one. Also here every control flow element will be represented through one or several edges within a view. The difference is that every edge here is labelled by names of substituted elements. In order to realize this, a view on workflow instances will be extended to the following one:

$$View =< N, E, name_N, name_E > \tag{10}$$

While $name_N$ is a function providing names (or labels) for nodes, $name_E$ does the same for edges. For two tasks T_1 and T_2 which are directly connected through some path $p = (Task_1, CFElement_1, \ldots, CFElement_n, Task_2)$ the function $name_E(e) = name_E((n_{T_1}, n_{T_2}))$ provides an ordered set of the elements' names: $name(CFElement_1), \ldots, name(CFElement_n)$. For example, consider the workflow instance introduced by the description of approximation method 1. The tasks T_1 and T_2 are directly connected by the path $p = (T_1, AND-split, T_2)$. For the edge $e = (n_{T_1}, n_{T_2})$ the function $name_E$ provides the value "$AND-split$". Now consider two tasks T_1 and T_2 which are directly connected by the path $p = (T_1, AND-split, XOR-split, AND-split, T_2)$. For that setup the function $name_E(e)$ provides the value "$AND-split, XOR-split, AND-split$".

The last thing to do is to redefine the equality of edges. Two edges $e_1 \in E_1$ and $e_2 \in E_2$ are defined to be equal if and only if $name(predecessor(e_1)) = name(predecessor(e_2))$ and $name(successor(e_1)) = name(successor(e_2))$ and $name_E(e_1) = name_E(e_2)$.

Using this extended model the similarity computation can be executed according to the approach presented in Sect. 3.2.

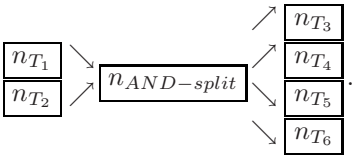
Approximation Method 3. The idea of this approximation method is to model the control flow elements of the type "sequence" as edges and other control flow elements (abbreviated with $\neg sequence$) as nodes. The only restriction is that for every $\neg sequence$-control flow element type (e.g. "$AND-split$") only one node

will be introduced in the view, and this is independent from the real number of the same elements that occurred in a workflow instance. Thus, for all pairs of workflow elements e_1 and e_2, with e_2 being a direct successor of e_1, the following components will be introduced in the view:

- nodes n_{e_1}, $n_{successor(e_2)}$ and edge $e = (n_{e_1}, n_{successor(e_2)})$ if element e_1 is a task and e_2 is a control flow element of a type "sequence".
- nodes n_{e_1}, $n_{type(e_2)}$ and edge $e = (n_{e_1}, n_{type(e_2)})$ if element e_1 is a task and e_2 is a $\neg$*sequence*-flow element.
- nodes $n_{type(e_1)}$, n_{e_2} and edge $e = (n_{type(e_1)}, n_{e_2})$ if element e_2 is a task and e_1 is a $\neg$*sequence*-flow element.
- nodes $n_{type(e_1)}$, $n_{type(e_2)}$ and edge $e = (n_{type(e_1)}, n_{type(e_2)})$ if the both elements are $\neg$*sequence*-flow elements.

Here, the name of every node $n \in N$ representing a $\neg$*sequence*-flow element is set to the element type: $name_N(n) = type(n)$.

For example, the following two parts of one workflow instance $T_1 \rightarrow \begin{vmatrix} \rightarrow T_2 \\ \rightarrow T_3 \end{vmatrix}$ and $T_4 \rightarrow \begin{vmatrix} \rightarrow T_5 \\ \rightarrow T_6 \end{vmatrix}$ will be transformed to the following nodes and edges within a view:



Also in this case the similarity computation can be carried out according to the approach presented in Sect. 3.2).

This approximation method could be further improved by counting the recurrent edges within a view. This can be achieved by using bags of edges instead of sets of edges. All operations on sets should be then replaced through operations on bags.

We have selected the approximation methods 1 and 3 for our empirical evaluation in order to get first insights whether and to what extent the results differ. In future, further experiments are required as well as a further extension of the described methods. For instance the control flow elements could be identified unambiguously by means of a naming function using their succeeding workflow elements.

4 Formative Evaluation

We did an experimental evaluation of the approximation methods 1 and 3. The test case base consists of 37 workflow instances from the chip design domain. They are derived from real change request documents of our industrial partner Silicon Image GmbH (formerly sci-worx). We presented each of the cases as a

query to the remainder of the case base according to the leave-one-out approach. 35 of them have an empirically best matching case (EBMC) from the remainder of the case base. The EBMC has been selected by a human expert. As a quality criterion for the evaluation, we investigated whether the empirically best matching case was in the 10 most similar cases according to approximation methods 1 and 3. Method 3 is implemented with the bag approach that we sketched above.

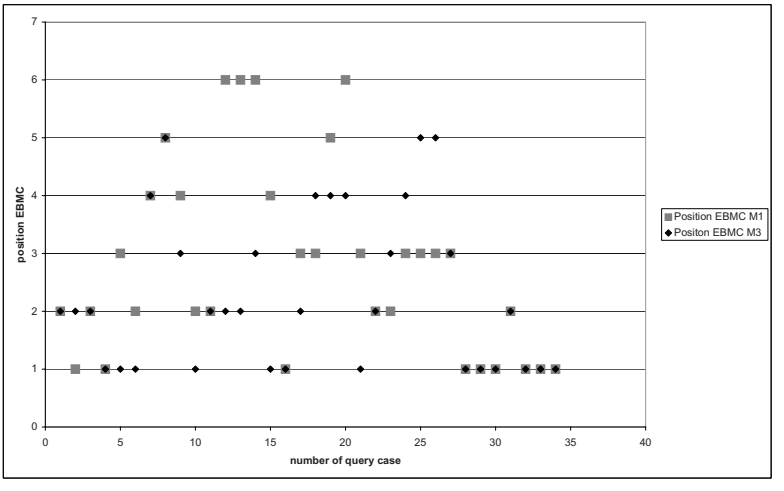


Fig. 6. Position of the empirically best matching case (EBMC) in the retrieval results

Both methods gave excellent results (compare Figures 6 – 7). For 34 of the queries, the EBMC was under the 10 most similar cases for both methods. For 21 of those, the EBMC was among the three most similar cases for both methods. Fig. 6 shows the positions of the particular EBMC's in the retrieval result lists. The squared dots stand for the results of method 1 and the diamonds for those of method 3. For example, for the case number five (x-axis) used as query the EBMC achieved position 3 (y-axis) for method 1 and the best position (position 1) for method 3. The expected position of the EBMC in a result set is with 2.91 for method 1 worse than for method 3 with 2.38. In 17 cases, the two methods gave the identical retrieval results. In 6 cases, method 1 achieved a better result and in 12 cases, method 3 was empirically more successful. In two of these cases of those, method 3 was significantly better; the empirically best matching case had a difference of 4 positions in the lists of most similar cases.

Figure 7 shows the frequency distribution of the positions of the EBMC's. Method 3 achieved better results than method 1, as the density of the distribution is higher for the better positions (the lower part of the distribution).

The representation according to method 1 required less nodes and edges for the same workflow instances. On average, this saved about a third of the size of the graph that was required by method 3.

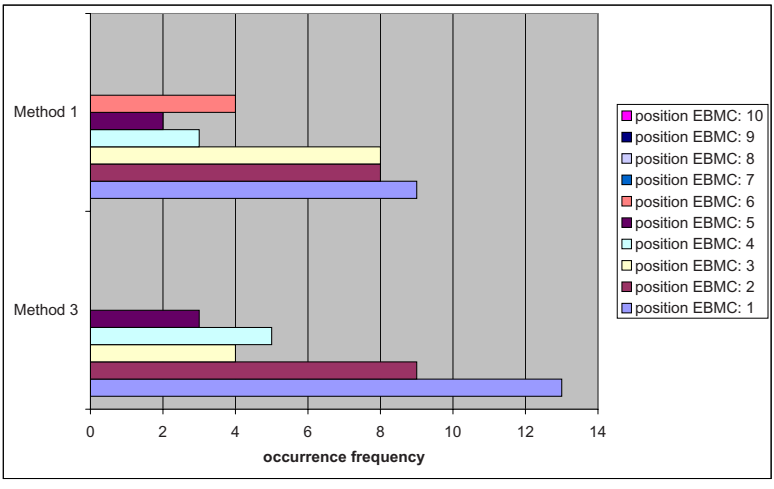


Fig. 7. Frequency distribution of the positions of the EBMC's

5 Conclusion

Handling the increasing dynamics of the market by means of agile workflow technology can be supported by CBR successfully. Our results have shown that the experience with the adaptation of ongoing workflows can be represented appropriately by the graph-based structure. Our new retrieval approach gave excellent experimental results showing that it provides a good approximation of the utility for the user. In addition, the experiments have clarified that it is worth-while to consider the control flow elements of the workflows explicitely within the similarity measure. The implementation seems to be computationally efficient due to our first experiments. The approximation graphs representing the agile workflows for retrieval purposes can be derived automatically from the process data and are available for further machine processing in future. We believe that our approach is suitable for developing a semi-automatic adaptation of workflows as well as for learning optimal weights for the distance measure, for instance by means of neural networks.

As next steps, we will conduct further experiments with approximation method 2 as well as with a more general distance model for agile workflows. Furthermore, we are going to do research on the employment of AI planning methods, for instance hierarchical planning [19], for semi-automatic, interactive adaptation of agile workflows.

Acknowledgements

The authors acknowledge the Federal Ministry for Education and Science (BMBF) for funding parts of our work under grant number 01M3075. We

acknowledge the assistance we have received from our industrial partners in the chip design company Silicon Image GmbH (formerly sci-worx).

References

1. Herrmann, T.: Lernendes Workflow. In: Herrmann, T., Scheer, A.-W., Weber, H. (eds.) Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Management-Systemen, pp. 143–154. Physica-Verlag, Heidelberg (2001)
2. Weber, B., Wild, W.: Towards the Agile Management of Business Processes. In: Althoff, K.-D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T.R. (eds.) WM 2005. LNCS (LNAI), vol. 3782, pp. 409–419. Springer, Heidelberg (2005)
3. Bunke, H., Messmer, B.T.: Similarity Measures for Structured Representations. In: Wess, S., Richter, M., Althoff, K.-D. (eds.) Topics in Case-Based Reasoning. LNCS, vol. 837, pp. 106–118. Springer, Heidelberg (1994)
4. van Elst, L., Aschoff, F.R., Bernardi, A., Maus, H., Schwarz, S.: Weakly-structured Workflows for Knowledge-intensive Tasks: An Experimental Evaluation. In: 12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, pp. 340–345. IEEE Computer Society, Los Alamitos (2003)
5. Freßmann, A., Maximini, R., Sauer, T.: Towards Collaborative Agent-Based Knowledge Support for Time-Critical and Business-Critical Processes. In: Althoff, K.-D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T.R. (eds.) WM 2005. LNCS (LNAI), vol. 3782, pp. 420–430. Springer, Heidelberg (2005)
6. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 434–448. Springer, Heidelberg (2004)
7. Luo, Z., Sheth, A., Kochut, K., Arpinar, B.: Exception Handling for Conflict Resolution in Cross-Organizational Workflows. Distributed and Parallel Databases 13(3), 271–306 (2003)
8. Schimm, G., van der Aalst, W.M.P., van Dongen, B., Herbst, J.: Workflow Mining: a Survey of Issues and Approaches. Data and Knowledge Engineering. 47(2), 237–267 (2003)
9. Reichert, M., Rinderle, S., Dadam, P.: ADEPT Workflow Management System: Flexible Support For Enterprise-wide Business Processes (Tool Presentation). In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 370–379. Springer, Heidelberg (2003)
10. Minor, M., Koldehoff, A., Schmalen, D., Bergmann, R.: Configurable Contexts for Experience Management. In: Gronau, N. (ed.) 4th Conference on Professional Knowledge Management - Experiences and Visions. Potsdam, University of Potsdam, GITO-Verlag, Berlin vol. 2. pp. 119–126 (2007)
11. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)
12. Bergmann, R. (ed.): Experience Management. LNCS (LNAI), vol. 2432. Springer, Heidelberg (2002)
13. Babai, L., Erdös, P., Selkow, S.M.: Random Graph Isomorphism. SIAM Journal of Computation 9, 628–635 (1980)
14. Babai, L., Kucera, L.: Canonical Labelling of Graphs in Linear Average Time. In: Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science, pp. 39–46 (1979)

15. Ullman, J.R.: An Algorithm for Subgraph Isomorphism. Journal of the Association for Computing Machinery. 23(1), 31–42 (1976)
16. Brandstädt, A.: Graphen und Algorithmen. Teubner, Stuttgart (1994)
17. Mehlhorn, K.: Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness. Springer, Heidelberg (1984)
18. Wagner, K., Bodendiek, R.: Graphentheorie I. BI-Wissenschaftsverlag, Mannheim (1989)
19. Nau, D.S., Cao, Y., Lotem, A., Muñoz-Avila, H.: SHOP: Simple Hierarchical Ordered Planner. In: Dean, T. (ed.) IJCAI 99. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden, July 31 - August 6, 1999, vol. 2, pp. 968–975. 1450 pages, Morgan Kaufmann, San Francisco (1999)