published in: Proceedings of the Sixth East-European Conference on Advances in Databases and Information Systems (ADBIS 2002), Springer Verlag, LNCS 2435, pp 326-339

A Meta Model for Structured Workflows Supporting Workflow Transformations

Johann Eder and Wolfgang Gruber

Department of Informatics-Systems,
Univ. Klagenfurt, A-9020 Klagenfurt, Austria
{eder,gruber}@isys.uni-klu.ac.at

Abstract. Workflows are based on different modelling concepts and are described in different representation models. In this paper we present a meta model for block structured workflow models in the form of classical nested control structure representation as well as the frequently used graph representations. We support reuse of elementary and complex activities in several workflow definitions, and the separation of workflow specification from (expanded) workflow models. Furthermore, we provide a set of equivalence transformations which allow to map workflows between different representations and to change the positions of control elements without changing the semantics of the workflow.

1 Introduction

Workflow management systems (WFMSs) improve business processes by automating tasks, getting the right information to the right place for a specific job function, and integrating information in the enterprise [6, 9, 7, 1]. Here, we concentrate on the primary aspects of a process model [12], the control structures defining the way a WFMS would order and schedule workflow tasks. We do not cover other aspects like data dependencies, actors, or organizational models.

Numerous workflow models have been developed, based on different modelling concepts (e.g. Petri Net variants, precedence graph models, precedence graphs with control nodes, state charts, control structure based models) and on different representation models (programming language style text based models, simple graphical flow models, structured graphs, etc.). Transformations between representations can be difficult (e.g. the graphical design tools for the control structure oriented workflow definition language WDL of the workflow system Panta Rhei had to be based on graph grammars to ensure expressiveness equality between text based and graphical notation [4]).

In this paper, we consider in particular transformations, which do not change the semantics of the workflow. Such transformation operations may be applied to a process model SWF to transform it into SWF' such that SWF and SWF' still maintain underlying structural relationship with each other.

Equivalence transformations are frequently needed for workflow improvements, workflow evolution, organizational changes, and for time management in

```
WFS1:WF-Spec      A1:Activity      A2:Activity      A3,A4,A5:Activity
 sequence          sequence         conditional      elementary
  O1:A1             O2:A2            O5:A4          end
 end                O3:A3            O6:A5
end                 O4:A2           end
                   end              end
                  end
```

Fig. 1. Workflow specification example (control structure)

workflow systems [2, 13]. E.g. for time management it is important to compute due dates for all activities. The algorithms for that are typically more efficient in graph based representations. Equivalence transformations are e.g. necessary for generating timed workflow graphs [3]. For other purposes (e.g. transactional workflows [5]), control structure based representations are preferred.

The main contributions of this paper are: we present a workflow meta model for capturing structured workflows. This meta model supports hierarchical composition of complex activities. Activities, both elementary and complex activities can be used in several workflow definitions, definition of complex activities. We present a notion for equivalence of workflow models and introduce a series of basic transformations preserving the semantics of the workflows.

2 Workflow Models

2.1 Structured workflow definition

A workflow is a collection of *activities, agents,* and *dependencies* between activities. Activities correspond to individual steps in a business process, agents (software systems or humans) are responsible for the enactment of activities, and dependencies determine the execution sequence of activities and the data flow between them. In this paper, we concentrate on the activities and the control dependencies between the activities.

We assume that workflows are *well structured*. A well-structured workflow consists of m sequential activities, $T_1 \dots T_m$. Each activity T_i is either elementary, i.e., it cannot be decomposed any further, or complex. A complex activity consists of n_i parallel, sequential, conditional or alternative sub-activities ${T_i}^1, \dots, {T_i}^{n_i}$, each of which is either elementary or complex. Typically, well structured workflows are generated by workflow languages with the usual control structures which adhere to a structured programming style (e.g. Panta Rhei [4]).

Fig. 1 shows an example of a workflow definition. The control structures define complex activities. Within a complex activity a particular activity may appear several times. To distinguish between those appearances, we introduce the notion of occurrences [5, 10]. An occurrence is associated with an activity and represents the place where an activity is used in the specification of a complex activity. Each occurrence, therefore, has different predecessors, and successors.

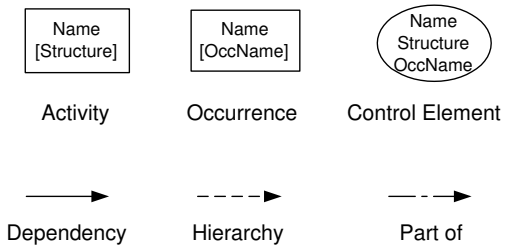

Fig. 2. Graphical elements

The distinction between an activity and its (multiple) occurrence(s) is important for reuseability, i.e. an activity is defined once and it is used several times in workflow definitions. Also for maintenance, it is only necessary to change an activity once, and all its occurrences are changed too. This allows that new workflows can easily be composed using predefined activities. Such a composition is also called a *workflow specification*.

2.2 Workflow graphs

Structured Workflows can also be represented by *structured workflow graphs*, where nodes represent activities or control elements and edges correspond to dependencies between nodes. An *and-split* node refers to a control element having several immediate successors, all of which are executed in parallel. An *and-join* node refers to a control element that is executed after all of its immediate predecessors finish execution. An *or-split* node refers to a control element whose immediate successor is determined by evaluating some boolean expression (conditional) or by choice (alternative). An *or-join* node refers to a control element that joins all the branches after an or-split.

The graph representation of a workflow can be structured in a similar way as workflow definitions in text based workflow definitions in block-structured workflow languages. Directed edges stand for dependencies, hierarchical relations, and part-of relations between nodes. Figure 2 shows the graphical elements.

Activity- and occurrence nodes are represented by a rectangle, in which the name of the representing activity or occurrence is indicated. An activity node features in addition the structure (only in complex activities) of the representing activity. The *structure* indicates the control structure ('seq', 'par', 'cond' or 'alt') of a complex activity. At the model level, nodes feature the name of the related specification occurrence that is encapsulated between round brackets. Control elements are represented by a circle in the graph, in which the name and the structure of the control element is indicated. Furthermore, any existing predicate of a node will be depicted between angle brackets below the graphical element.

Figure 3 shows the workflow defined above in graph notation.

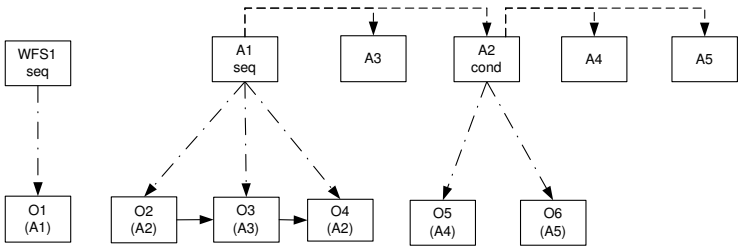

Fig. 3. Workflow specification graph example

A workflow graph is *strictly structured*, if each split node is associated with exactly one join node and vice versa and each path in the workflow graph originating in a split node leads to its corresponding join node.

For the purpose of allowing more transformations (see section 4) and the separation of workflow instance types in the workflow model we also offer a less strict notion. Here a split node may be associated with several join nodes, however, a join node corresponds to exactly one split node. Each path originating in a split node has to lead to an associated join node. Such graphs are results of equivalence transformations necessary e.g. for time management.

Both representations of workflows can be freely mixed in our approach which we call *hybrid graph* or *graph-based* workflow representations. It is possible to use graph based structures as complex activities, and use structured composite activities in graph based representations on the other hand.

2.3 Workflow model

In the workflow specification, the concept of occurrence helps to distinguish between several referrals to the same activity within a complex activity. When a complex activity is used several times within a workflow, we also have to distinguish between the different appearances of occurrences. Therefore, a model is required that corresponds to the specification, so that for the definition of the dependencies between activities, an occurrence of an activity in a workflow has to be aware of its process context. Therefore, we transform the design information (specification) contained in the meta model into a tree-like structure. In such a tree-like structure different appearances of the same activity are unambiguously distinguished, such that we can define the dependencies between activities on basis of these occurrences. We call these items *model elements*, and the workflow consisting of model elements the *workflow model*.

Fig. 4 shows the workflow model for the workflow specification in Fig. 1. In the following example, the model elements $M2$ and $M4$ have their own contexts with $M5$ and $M6$, respectively $M7$ and $M8$ and they are built up like a tree. Fig. 5 shows the workflow model in graph notation, in the upper half the full unflattened model and in the lower half the full flattened model.


```
WFM1:WF-Model    M1:ModelElem    M2:ModelElem    M4:ModelElem    M3,M5,M6,M7,M8:
 sequence         sequence        conditional     conditional     ModelElem
  M1:O1            M2:O2           M5:O5           M7:O5            elementary
 end              M3:O3           M6:O6           M8:O6           end
end               M4:O4          end             end
                 end             end             end
                end
```

Fig. 4. Workflow model control structure example

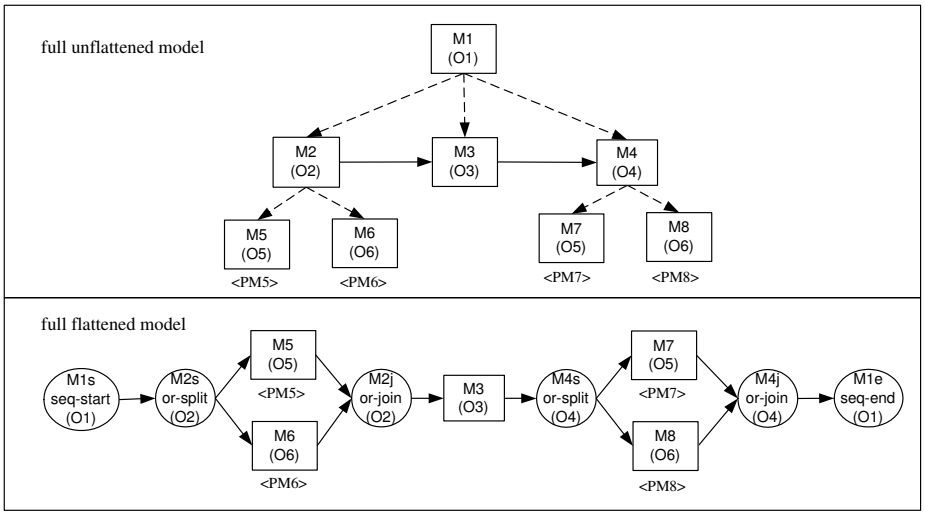

Fig. 5. Workflow model graph example

3 Workflow meta model

Structure and characteristics of a workflow can be sufficiently described by a workflow meta model. In this paper, we use UML as meta-modelling language. The meta model shown in Fig. 6 gives a general description of the static scheme aspects (the build time aspects) of workflows. The meta model presented in Fig. 6 is adopted to the purpose of this paper and does therefore not contain all necessary components of a workflow meta model. We briefly discuss the elements of the meta-model. The important concepts have already be described with examples in the previous section.

Workflows and activities A *Workflow* consists of activities which are either (external) workflows, elementary or complex activities. Complex activities are composed of other activities, represented as (activity-) occurrence in the composition of a complex activity. The type of a complex activities describes its control structure (*seq* for a sequential, *par* or *and* for a parallel, *cond* or *or* for a

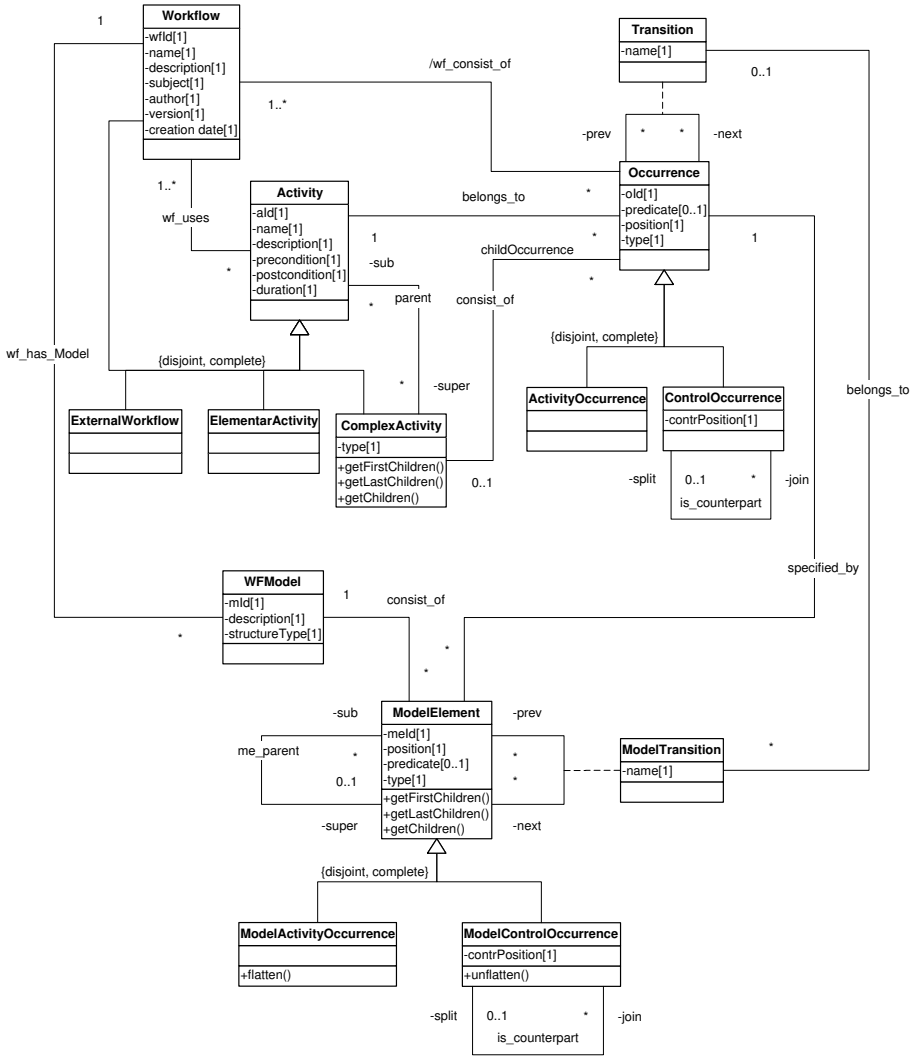

Fig. 6. Workflow Metamodel

conditional, or *alt* for an alternative activity). We also register in which *parent*-activities an activity appears.

Occurrences As outlined above, the notion of occurrence is central in our meta model. The attribute *predicate* represents the condition for child occurrences of conditional activities and for occurrences that follow an or-split. The attribute *position* indicates the processing position within the scope of the complex activity (values: 'start', 'between', 'split', 'join', 'end' or 'start/end'). We distinguish activity and control occurrences. Each occurrence *belongs_to* exactly one activity. The association class *Transition* models the predecessor and successor for each child occurrence of a sequence activity. Every child occurrence of conditional activities and every occurrence that follows an or-split is associated with a predicate. The class *ComplexActivity* has the methods *getFirstChildren* (returns the child occurrences with the value 'start' or 'start/end' in *position*), *getLastChildren* (returns the child occurrences of the complex activity with the value 'end' or 'start/end'), and *getChildren* (returns all child occurrences of the complex activity). A *ControlOccurrence* represents a control element (split or join). *cntrPosition* distinguishes between split- and join control elements. The association *is_counterpart* represents which join closes which split.

Workflow Model: The attribute *structureType* in the class *WFModel* indicates, whether the workflow is a strict block-structured workflow or a hybrid workflow. A model consists of *ModelElements* which are *specified_by* exactly one object of the class *Occurrence* - either an activity or a control occurrence. Model elements can have a *ModelTransition*.

A model occurrence of a complex activity can be represented through a split- and a join control element(control occurrences), if the model occurrence is flattened. The method *unflatten* which builds up composition hierarchies is the inverse method to *flatten* of the class *ModelActivityOccurrence*. The association *is_counterpart* of the class *ModelControlOccurrence* associates related join and split control elements.

4 Workflow Transformations

Workflow transformations are operations on a workflow *SWF* resulting in a different workflow *SWF'*. Each workflow transformation deals with a certain aspect of the workflow (e.g. move splits or joins, eliminate a hierarchy level). In the following we provide a set of transformations, which do not change the semantics of the workflow according to the definition of equivalence given below. Complex transformations can be established on this basic set of transformations by repeated application. Transformations are feasible in both directions, i.e. from *SWF* to *SWF'* and vice versa from *SWF'* to *SWF*.

4.1 Workflow instance type

Due to conditionals not all instances of a workflow processes the same activities. We classify workflow instances into workflow instance types according to the

actual executed activities. Similar to [11], a *workflow instance type* refers to (a set of) workflow instances that contain exactly the same activities, i.e., for each or-split node in the workflow graph, the same successor node is chosen; resp. for each conditional complex activity the same child-activity is selected. Therefore, a workflow instance type is a submodel of a workflow where each or-split has exactly one successor; resp. each conditional or alternative complex activity has exactly one subactivity.

4.2 Equivalence of workflows

Workflows are equivalent, if they execute the same tasks in exactly the same order. Therefore, the equivalence of correct workflows ($WF1 \equiv WF2$) is based on equivalent sets of workflow instance types.

Equivalent workflows: Two workflows are equivalent, if their sets of instance types are equivalent. Two instance type sets are equivalent if and only if for each element of one set there is an equivalent element in the other set.

Equivalent workflow instance types: Two workflow instance types are equivalent, if they consist of occurrences of the same (elementary) activities with identical execution order. The position of or-splits and or-joins in instance types is irrelevant since an or-split has only one successor in an instance type. Fig. 7 shows the instance types of the above workflow in Fig. 4.

4.3 Flatten/Unflatten

The operation *flatten* eliminates a level of the composition hierarchy in a model by substituting an occurrence of a complex activity by its child occurrences and two control elements (split and join element). Between the split control element and every child occurrence, a dependency is inserted, so that the split element is the predecessor of the child occurrences. Also between the last child occurrence(s) and the join control element, a dependency is inserted, so that the join element is the successor of the child occurrence. Fig. 8 shows an example of such a transformation. Here, applying the transformation *flatten* in the workflow model SWF on occurrence $M1$ with the child occurrences $M2$ and $M3$, results in the workflow *SWF'*, where $M1$ is replaced by the split $S1$ and the join $J1$. $S1$ is the predecessor of $M2$ and $M3$, and $J1$ is the successor of $M2$ and $M3$. Applying the operation flatten repeatedly on a workflow model so that no further hierarchy can eliminated, is called *total flatten*(see Fig.5). The inverse function to flatten is called *unflatten*.

4.4 Moving Joins

Moving Joins means changing the topological position of a join control element (and-, or-, alt-join). This transformation separates the intrinsic instance types contained in a workflow model. Some of the following transformations require node duplication. In some cases moving a join element makes it necessary to move the corresponding split element as well.


```
IT1:InstanceT   M1:ModelElem   M2:ModelElem   M4:ModelElem   M3,M5,M7:
 sequence        sequence       conditional    conditional   ModelElem
  M1:O1           M2:O2          M5:O5          M7:O5         elementary
 end             M3:O3          end            end          end
end              M4:O4         end            end
                 end
                end

IT2:InstanceT   M1:ModelElem   M2:ModelElem   M4:ModelElem   M3,M5,M8:
 sequence        sequence       conditional    conditional   ModelElem
  M1:O1           M2:O2          M5:O5          M8:O6         elementary
 end             M3:O3          end            end          end
end              M4:O4         end            end
                 end
                end

IT3:InstanceT   M1:ModelElem   M2:ModelElem   M4:ModelElem   M3,M6,M7:
 sequence        sequence       conditional    conditional   ModelElem
  M1:O1           M2:O2          M6:O6          M7:O5         elementary
 end             M3:O3          end            end          end
end              M4:O4         end            end
                 end
                end

IT4:InstanceT   M1:ModelElem   M2:ModelElem   M4:ModelElem   M3,M6,M8:
 sequence        sequence       conditional    conditional   ModelElem
  M1:O1           M2:O2          M6:O6          M8:O6         elementary
 end             M3:O3          end            end          end
end              M4:O4         end            end
                 end
                end
```

Fig. 7. Workflow instance types

Join moving over activity: A workflow SWF with an or- resp. alt-join $J1$ followed by activity occurrence $M3$, can be transformed to workflow SWF' through node duplication, so that the join $J1$ is delayed after $M3$ as shown in Fig. 9. Here, $M3$ will be replaced by its duplicates $M31$ and $M32$, so that $J1$ is the successor of $M31$ and $M32$, and $M1$ is the predecessor of $M31$ and $M2$ is the predecessor of $M32$. This transformation, and all of the following, can be applied to structures with any number of paths.

Moving join over join: A workflow SWF with a nested or-structure (i.e. within an or-structure with the split $S1$ and the corresponding join $J1$ there is another or-structure with the split $S2$ and the corresponding join $J2$), the inner join $J2$ can be moved behind the outer join $J1$, which requires also to move the corresponding split element $S2$ and to adjust the predicates according to the changed sequence of $S1$ and $S2$ by conjunction or disjunction. This change means that the inner or-structure is put over the outer. An example of this transformation in the workflow SWF' is shown in Fig. 10.

Moving or-join over alt-join: For a workflow SWF with a nested alt/or-structure, i.e. within an alt-structure with the split $S1$ and the join $J1$ there is an or-structure with the split $S2$ and the join $J2$, the inner join $J2$ can be

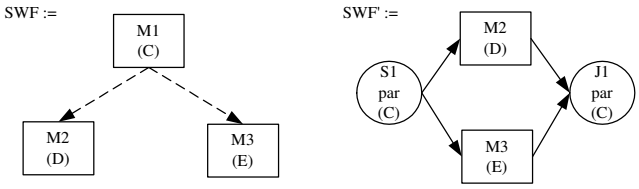

Fig. 8. Flatten

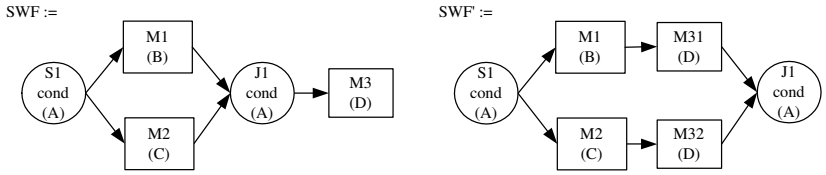

Fig. 9. Join moving over activity

moved behind the outer join $J1$. This also requires to move the corresponding split element $S2$ and to duplicate control elements and occurrences and adjust the predicates. This change means that the inner or-structure is put over the outer. An example of this transformation is given in Fig. 11.

Join coalescing: In a workflow SWF with a nested or-structure, i.e. within an or-structure with the split $S1$ and the join $J1$ there is an or-structure with the split $S2$ and the join $J2$, $J2$ can be coalesced with $J1$, which requires also to coalesce the corresponding split element $S2$ and $S1$. This change means that two or-structures are replaced by a single one. The predicates must be adapted. An example for this transformation is given in Fig. 12. This transformation is similar to the structurally equivalent transformations represented in [12] considering the differences in the workflow models.

Moving join over and-join (Unfold): The *unfold* transformation produces a graph based structure which is no longer strictly structured and requires *multiple sequential successors*, which means that a node, except split, could have more than one sequential successor in the workflow definition, however, in each instance type every node except and-splits has only one successor (the other successors of the definition are in other instance types).

An or-join $J2$ can be moved behind its immediately succeeding and-join $J1$, requiring duplication of control elements. The transformation is shown in Fig. 13 and Fig. 14. To move $J2$ behind $J1$ we place a copy of $J1$ behind every predecessor of $J2$, such that each of these copies of $J1$ has additionally the same predecessor as $J1$ except $J2$. A copy of $J2$ is inserted, such that it has the copies of $J1$ as predecessor and the successor of $J1$ as successor. Then $J1$ is deleted with all its successor- and predecessor dependencies. If $J2$ has no longer a successor,

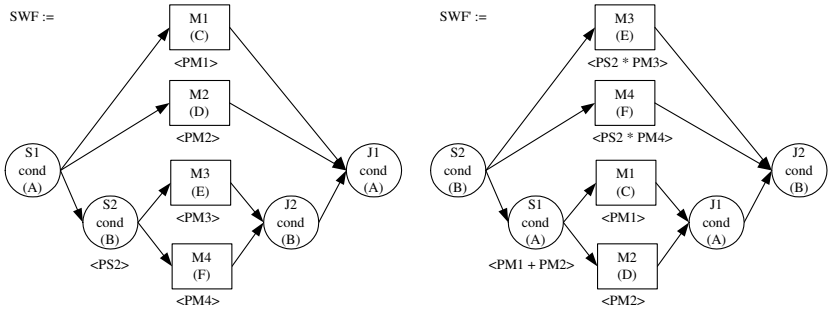

Fig. 10. Join moving over join

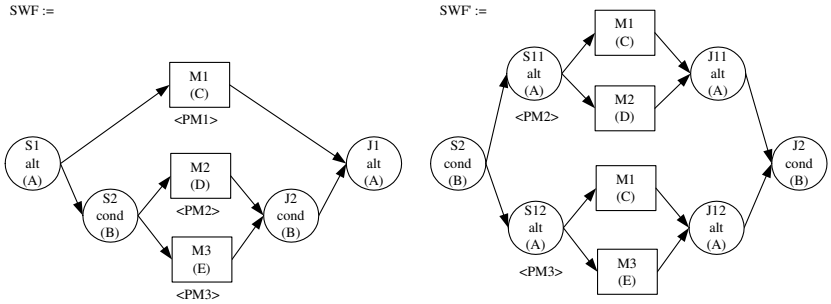

Fig. 11. OR-join moving over alt-join

it will also be deleted. *Partial unfold* as it is described in [3] is a combination of already described transformations.

4.5 Split Moving

Split Moving changes the position of a split control element. This transformation separates (moving splits towards start) or merges (moving splits towards end) the intrinsic instance types contained in a workflow model, in analogy to join moving. Not every split can be moved. Moving an alt-split is always possible. For an or-split it is necessary to consider data dependencies on the predicates. Another considerably aspect of or-split moving is, that the decision which path of an or-split is selected will be transferred forward, so that uncertainty based on or-splits will be reduced.

Moving split before activity A workflow SWF with an or- resp. alt-split $S1$ with activity occurrence $M1$ as predecessor, can be transformed in the workflow SWF' through node duplication, so that $S1$ is located before $M1$ (see Fig. 15). Here, $M1$ will be replaced by its duplicates $M11$ and $M12$, so that $S1$

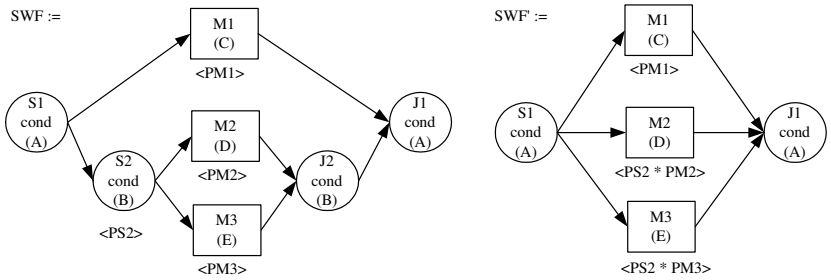

Fig. 12. Join coalescing

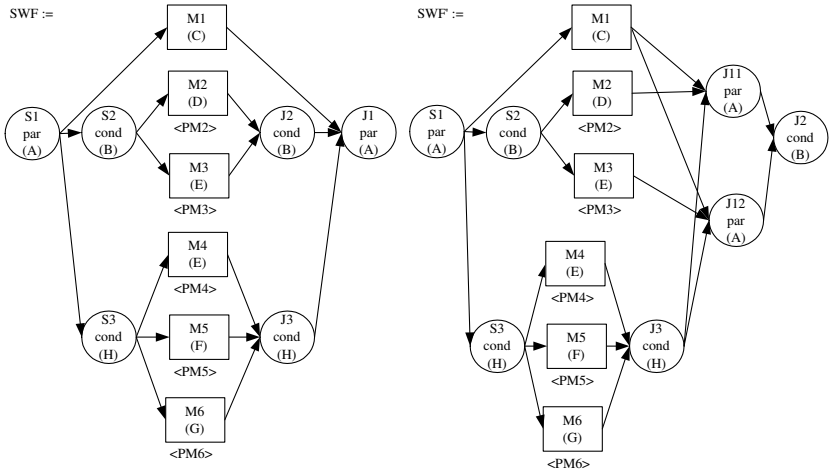

Fig. 13. Join moving over and-join (Unfold) - 1

is the predecessor of $M11$ and $M12$, and $M2$ is the successor of $M11$ and $M3$ is the successor of $M12$. Predicates are adjusted.

There are some more operations like moving and-join over or-join, introduced in [8], which - as we can show - is also an equivalence transformation. However, space limitations do not allow discussion of further transformations.

5 Related work

There is some work on workflow transformations reported in literature. In [13] various workflow patterns for different WFMS with different workflow models are catalogued. The alternative representations are employing different control elements and they are thought to be semantically equivalent, but there is no equivalence criterion nor are there any transformation rules.

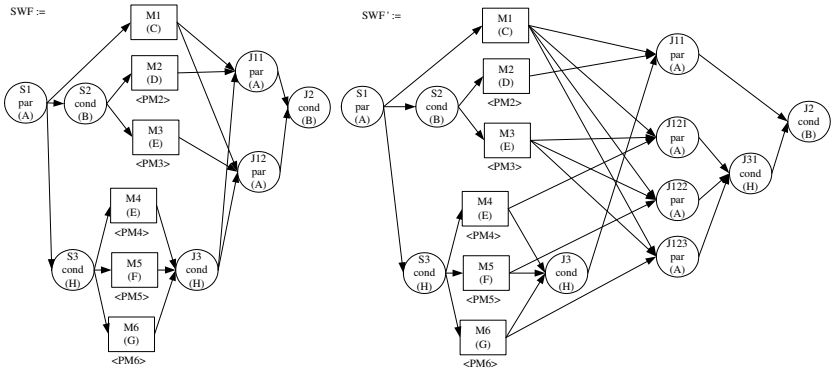

Fig. 14. Join moving over and-join (Unfold) - 2

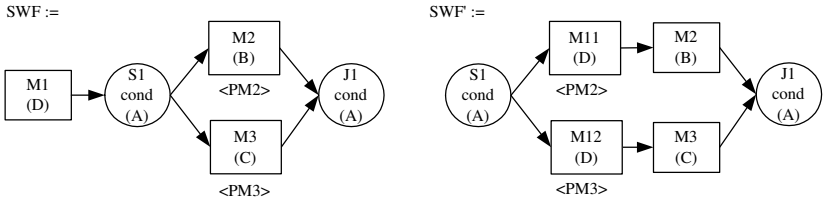

Fig. 15. Split moving before activity

Modelling structured workflows and transforming arbitrary models to structured models has been addressed in [8], based on the equivalence notion of bisimulation. In that paper, the authors investigate transformations based on several patterns and analyze in which situations transformations can be applied. One of the specified transformations is moving split-nodes, which is, in contrast to our work, considered as a non-equivalent transformation. The so called *overlapping structure*, which has been introduced in the context of workflow reduction for verification purposes, is adopted in our work and it is used by the transformation *Moving and-join over or-join* (omitted here due to space limitations).

Finally, in [12], three classes of transformation principles are identified to capture evolving changes of workflows during its lifetime. We are only focusing on the first class, namely on structurally equivalent transformations. In this work, the equivalence criterion (relationship) for structurally equivalent workflows is too restrictive, because the workflows must have identical sets of *execution nodes*, which implies that transformations using node duplication can't be applied. Considering the differences in the workflow models, we adopted eliminating of join-nodes as join coalescing with different semantics.

6 Conclusion

We presented a metamodel for workflow definition that supports control structure oriented as well as graph based representation of processes. Important aspects of this meta model are the elaborated hierarchical composition supporting re-use of activity definitions and the separation of specification and model level workflow descriptions. Through the notion of instance types we give and define the (abstract) semantics of process definitions which allows the definition of the equivalence of workflows. The main contribution of this work is the development of a set of basic schema transformation that maintain the semantics.

There are several applications for the presented methodology. It serves as sound basis for design tools. It enables analysts and designers to incrementally improve the quality of the model step by step. We can provide automatic support to achieve certain presentation characteristics of a workflow model. A model can be transformed to inspect it from different points of view. In particular a model suitable for conceptual comprehension can be transformed to a model better suited for implementation.

References

1. Work Group 1. Interface 1: Process definition interchange. *Workflow Management Coalition*, V 1.1 Final(WfMC-TC-1016-P), October 1999.
2. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. *Lecture Notes in Computer Science 1021*. Springer, 1995.
3. J. Eder, W. Gruber, and E. Panagos. Temporal modeling of workflows with conditional execution paths. *Lecture Notes in Computer Science 1873*. Springer, 2000.
4. J. Eder, H. Groiss, and W. Liebhart. The workflow management system panta rhei. In A. Dogac, et. al. (eds.), *Advances in Workflow Management Systems and Interoperability, Springer, 1997*.
5. J. Eder and W. Liebhart. The workflow activity model WAMO. In S. Laufmann, et.al, editors, *Cooperative Information Systems, 3rd Int. Conf., CoopIS, 1995*.
6. D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
7. D. Hollingsworth. The workflow reference model. *Workflow Management Coalition*, Issue 1.1(TC00-1003), January 1995.
8. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. *Lecture Notes in Computer Science 1789*. Springer, 1999.
9. P. Lawrence. *Workflow Handbook*. John Wiley and Sons, New York, 1997.
10. W. Liebhart. *Fehler- und Ausnahmebehandlung im Workflow Management*. PhD thesis, Universität Klagenfurt, 1998.
11. O. Marjanovic and M. E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems, KAIS*, vol 1. Springer, 1999.
12. W. Sadiq and M. E. Orlowska. On business process model transformations. *Lecture Notes in Computer Science 1920*. Springer, 2000.
13. W. M. P. van der Aalst, et.al. Advanced workflow patterns. *Lecture Notes in Computer Science 1901*. Springer, 2000.

