1. INTRODUCTION

Workflow schema are constantly changing [6][7] driven by new customer requirements or business processes re-engineering. Schema evolution is less of a problem when no workflow is active, which could be achieved by waiting until all workflows are terminated, or by aborting active ones. These are however not viable solutions, because workflows can be of long duration and comprise considerable amounts of work. This results in a number of variants that may become problematic to manage especially when creating new instances of the schema. However, many of these variants can be merged together to reduce the number of templates to choose from while instantiating a new workflow instance. Merging can be accomplished only when two workflows are compatible or conformant. In this effort we aim at solving the problem of defining workflow conformance using OO concepts of inheritance and merging conformant workflows.

The rest of this paper is organized as follows. We first formulate the problem and then present our solution which is based on task inheritance. We finally present some conclusions and opportunities for future work.

2. Problem Formulation

Specifically the aim of this study was to:

1. Define what kind of change to a workflow can be considered as a safe change and when can two workflows be considered conformant.
2. Determine techniques to merge two conformant workflows, to give a workflow which acts as their superset. Thus, determine what can be considered as a safe merge, i.e. merging of workflows which will maintain the correctness of the system.
3. Finally to determine which instances of a given schema can be migrated in the midst of their execution to the merged superset without affecting the correctness of the system.

We illustrate the notion of conformance with an example first. The Figure 1 shows the original schema followed by two variants for a workflow that does insurance claims processing. Now variant 1 cannot be merged with the original because the original workflow allows for the lawyer and police verifications to proceed in parallel and makes a conclusion after they are over whereas Variant 1 might abort after police verification. Where as variant 2 may be merged with the original schema if we can provide an OR node after police and lawyer verification. That is the two instances can be merged if we can provide a mechanism which gives the user an option to skip the re-verification of customer information. The merged superset of variant 2 and the original is shown in Figure 4.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC ’06, April 23-27, 2006, Dijon, France.
Copyright 2006 ACM 1-59593-108-2/04/2006...$5.00.

1447
3. Our Approach – Schema Inheritance

We borrow from the notion of inheritance which is one of the cornerstones of object orientation and apply it to workflows. The concept can be applied to workflows in which case the class corresponds to workflow task and process definitions and object corresponds to a running instance. Some WFMS such as InConcert [3] allows for building workflow class hierarchies. In our approach we define four inheritance relations for workflow processes with inheritance implying substitutability in the usual sense.

1) Consider two workflow process definitions $x$ and $y$. If it is not possible to distinguish the behaviors of $x$ and $y$ when only tasks of $x$ that are also present in $y$ are executed, then $x$ is a subclass of $y$. This definition conforms to blocking tasks new to $x$. We term the resulting inheritance concept protocol inheritance.

2) Another mechanism would be to allow execution of new tasks but to consider only the affect of old ones. If it is not possible to distinguish the behaviors of $x$ and $y$ when arbitrary tasks of $x$ are executed, but when only the effects of tasks that are also present in $y$ are considered, then $x$ is a subclass of $y$. This inheritance notion is termed projection inheritance.

3) The two mechanisms (i.e. blocking and hiding) result in two orthogonal inheritance notions. Therefore, we also consider combinations of the two mechanisms. A workflow process definition is a subclass of another workflow process definition under protocol/projection inheritance if and only if both by hiding the new methods and by blocking the new methods one cannot detect any differences, i.e. it is a subclass under both protocol and projection inheritance.

4) A workflow process definition is a subclass of another workflow process definition under life-cycle inheritance if and only if by blocking some newly added tasks and by hiding some others one cannot distinguish between them. Life-cycle inheritance is more general than the other three inheritance relations.

We use the above definitions of substitutability to transform existing workflows in order to create workflows that are superset of two or more schemas. This deals with the control flow. We now turn our attention to the specific tasks comprising the workflow. We can use the concept of inheritance for nodes as well to decide whether two nodes can be merged together. To achieve this we start by assuming that there exists ontology of the possible kinds of nodes which can be present in a given workflow. So the type of nodes are fixed and each node typically consists of a set of attributes which can be considered as the work the node does or the action it takes when the node is activated. A node can also extend another node similar to the way in which class can extend another class in OO systems. Thus when a node $A$ extends another node $B$ it gets all the attributes and properties of node $B$ and can have further additional properties of its own. In this way we can merge two nodes $A$ and $B$ if:

- Both nodes $A$ and $B$ are of the same type, i.e. they have exactly similar set of attributes and properties.
- Node $A$ is a subclass of $B$ i.e. $B$ extends $A$. In this case the new node formed is of type $B$. Similarly, if $A$ is a super type of $B$ then the new node formed on merger of these nodes is of type $A$.

In case a new node $C$ is formed which extends both $A$ and $B$ then the new node $C$ can replace the two nodes $A$ and $B$ in the original workflow only if nodes $A$ and $B$ are not directly present together.

4. Conclusions

There exists numerous efforts [1][2][4][5][6] in the area of modifying control flow at run time to maintain correctness. We have mainly focused on the notions of conformance with a view to merging multiple schemas – these notions have led to strategies to transform existing schemas to act as the superset of two schemas. We have also used the notion of inheritance for merging tasks in WFMS. We have implemented these ideas in Bonita – an open source WF engine as a proof of concept. We are now working towards extending these concepts to include semantic equivalence of tasks beyond just control flow.

5. REFERENCES


