Difference-Preserving Process Merge

Kristof Böhmer and Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science
a1063026@unet.univie.ac.at, stefanie.rinderle-ma@univie.ac.at

Abstract. Providing merging techniques for business processes fosters the management and maintenance of (large) process model repositories. Contrary to existing approaches that focus on preserving behavior of all participating process models, this paper presents a merging technique that aims at preserving the difference between the participating process models by exploiting the existence of a common parent process, e.g., a reference or standard process model.

Keywords: Process Design, Process Merging.

1 Motivation

Nowadays, many companies face a multitude of different business process models or versions being in use simultaneously. As an example take the SAP reference process catalog containing more than 600 different process definitions [1]. Aside modeling and enacting these processes, their models have to be maintained and often adapted to reflect constantly changing market situations which can be the result of takeovers, acquisitions or amendments [2]. Hence, techniques to support users in keeping track and managing different process models or model versions are of great importance [3]. This has been recognized by literature, particularly resulting in techniques for process (model) merge [4, 5]: given a set of process models $\{P_1, \ldots, P_n\}$ existing techniques aim at preserving the behavior of each process model P_i within the merged model P_{merge}. An example for this approach is depicted in Figure 1. On the left side process models *Process 1* and *Process 2* are to be merged. The behavior-preserving merge produces the result depicted in the middle of the figure. Without presenting formal details on the process model due to space restrictions, the number of paths from the original models (1 or 2 respectively) has increased to 8 paths in P_{merge}. In some cases, even some additional executions paths might added which were not present in one of the original process models P_i.

Despite obvious advantages of this kind of merge techniques such as quickly giving an overview about the merged processes, they might create quite complex and hard to understand results [6] with increasing number of execution paths. This high number of execution paths might also necessitate that users have to configure the resulting process model for each use case.

This leads to the following questions:

1. Are there any other ways of merging process models?
2. How can process merging techniques be evaluated?

Y.T. Demey and H. Panetto (Eds.): OTM 2013 Workshops, LNCS 8186, pp. 718–721, 2013.

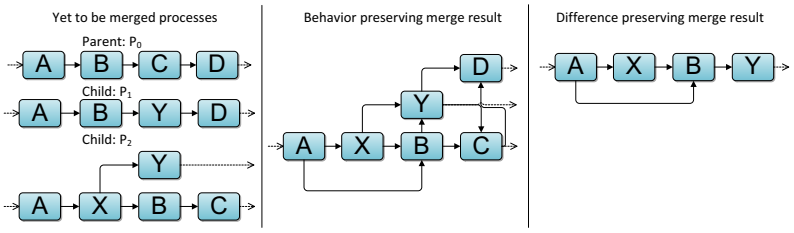

Fig. 1. Comparison of behavior and difference preserving merge

2 Difference-Preserving Merge

Following the above two questions, in this paper, we try to adopt the idea of merging software code (cf. e.g., [7]) in merging process models in a difference-preserving way. One assumption that has to be made is that the process models to be merged (*child processes*) all relate to an initial process (for now on called *parent process*), i.e., a child process has been derived from the parent process by applying a set of change operations, e.g., inserting or deleting nodes. An example is show in Figure 1 where P_1 and P_2 are children of P_0. P_1, for example, was derived from P_0 by deleting node C and inserting node Y. These differences are then to be preserved in the merge result.

Our presumption is that the difference-preserving merge will create much smaller, transparent, understandable and accessible results especially if the results should e.g. be merged again. Take result of difference-preserving merge as shown in Fig. 1 (right side): this model contains 4 instead of 6 elements and 2 instead of 8 paths when compared to the behavior-preserving merge result.

Approach: In the following we consider a set of three basic change operations which can be applied on the edges or nodes of a parent process to create a new child process. So we will tackle operations to add new edges and nodes using insert(edge(from, to)), and insert(node), to delete information by delete(node) or delete(edge) and to modify existing nodes e.g. by changing it's properties by calling modify(node, new data). Other high level operations like replace or move can be constructed using this basic operations. We also assume that all the applied operations will be used to create sound processes so that the later applied conflict resolution techniques only have to tackle merge related process conflicts.

Information gathering. The following steps will be executed sequentially to merge multiple processes. The first step will investigate the parent process in combination with the yet to be merged child processes. The behavior preserving merge ignores the parent process but we propose to use this additional information so that a so called *three way merge* can be applied [3]. It will be used to detect which elements (nodes and edges) have been deleted, added, or modified

to create one of the merged child processes. For each change operation an individual set including the affected elements (edges and nodes) will be created. The algorithms start by detecting the elements which have been deleted from the parent process. This information will then be stored at a set R. The set will be filled by comparing each element of the parent process with the child processes. If at least one child process does not contain the element it will be added to *R*. A similar approach will be used to detect newly generated elements. Therefore each element of the child processes will be compared with the parent process. If an element exists at a child but not at the parent it will be added to the set *N*. Also a set *M* containing the modified elements will be generated. For each element at the child processes it's companion at the parent process will be identified and, if any exists, their content will be compared. The element will be added to *M* if the compared content differs. If the element has been modified differently, at multiple child processes, the user has to decide which version should be taken. The last piece of information contains which elements have been transferred from the parent process to the child process without any modifications. So all the elements available at the parent process which are not present at the set *R* and *M* will be stored at a newly generated set *O*.

Creating the merged process. The second step will be used to merge the various changes\differences gathered from the parent and child processes. So the identified changes from all child process will be combined. Therefore the sets generated during the previouis step will be used. They are combined using the union operation to create $P_{merged} = (N \cup M \cup O)$. The set *R* will here be ignored because it has already been used to generate *O* so that all the detected delete operations will be preserved.

Optimizing the results. The third step will be used to enhance the result quality and will be applied onto P_{merge}. The first optimization will detect parts at the merged control flow where the merge created a new, not present at any child process, parallel control flow branch. Such a parallel execution can be problematic because it can cause concurrency issues like incorrect execution states, incomplete data or inappropriate calculations. On the other hand, it is also possible that it just cannot be realizable in day-to-day work. So if such a newly generated execution order (which was not present at the child processes) is detected the merger asks the user if she or he wants to reorder this e.g. to a sequential ordering by choosing between multiple auto generated alternative control flow recommendations.

The second optimization will try to find gaps at the control flow. Therefore each node n at P_{merge} will be checked if it has been correctly integrated, so that incoming and outgoing edges exists for this node. It also has to be taken into account that start and end nodes only need an incoming or outgoing edge. After a gap has been detected it will be closed by adding a newly generated edge. The end\start of the new edge will be n if incoming\outgoing edges are missing. The opposite side of the edge will be detected by analyzing the original control flow path which was used to integrate the node. At first the child processes will

be investigated, followed up by the parent process, if necessary. So the predecessors\successors, ordered by their distance to n, will be checked. The nearest node which is also available at P_{merge} will then be used as the start\end of the edge.

3 Evaluation and Conclusion

All algorithms and concepts behind the difference-preserving merge have been implemented as a proof of concept prototype. We experimented with different use cases of different complexity. Overall, we can conclude that the difference-preserving merge tends to excel the behavior-preserving merge in terms of simplicity of the produced merge results. This is advantageous for understandability, maintainability, and possible automation of the resulting process models. In turn, behavior-based approaches provide a complete overview on all participating models and can be produced with lower computational effort.

In future research, we will further investigate means to compare both approaches, preferably based on a real-world case study. Further on, we will work on "proving" that the approach is always difference-preserving. Another goal is to integrate the approach with existing work on calculating difference between process models such as [8].

References

1. Curran, T., Keller, G.: SAP R/3 Business Blueprint: Business-Engineering mit den R/3-Referenzprozessen. Addison-Wesley (1999)
2. Davenport, T.H., Short, J.E.: The New Industrial Engineering: Information Technology and Business Process Redesign. Sloan Mgmt. Review 31(4), 11–27 (1990)
3. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software Maintenance 22(6-7), 519–546 (2010)
4. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H.: Merging Event-Driven Process Chains. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 418–426. Springer, Heidelberg (2008)
5. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.: Merging Business Process Models. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 96–113. Springer, Heidelberg (2010)
6. Mendling, J., Reijers, H.A., Cardoso, J.: What Makes Process Models Understandable? In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 48–63. Springer, Heidelberg (2007)
7. Mens, T., Demeyer, S.: Software Evolution. Springer (2008)
8. Küster, J., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 244–260. Springer, Heidelberg (2008)