On Predicting Impacts of Customizations to Standard Business Processes

Pietro Mazzoleni, Aubrey Rembert, Rama Akkiraju, and Rong (Emily) Liu

IBM T.J. Watson research Center

Abstract. Adopting standard business processes and then customizing them to suit specific business requirements is a common business practice. However, often, organizations don't fully know the impact of their customizations until after processes are implemented. In this paper, we present an algorithm for predicting the impact of customizations made to standard business processes by leveraging a repository of similar customizations made to the same standard processes. For a customized process whose impact needs to be predicted, similar impact trees are located in a repository using the notion of *impact nodes*. The algorithm returns a ranked list of impacts predicted for the customizations.

Keywords: Process Re-engineering, Process Comparison, Best practices, Tree Comparison.

1 Introduction

In this paper, we consider the problem of predicting the impact of a given set of customizations to a standard business process. We look at the problem from an IT services providers perspective. IT services providers typically implement the desired customizations to standard processes offered by ERP vendors for many companies. In our solution, we assume that IT service providers maintain a repository of anonymized customizations, each with its associated impact. We predict the impact of changes done on a standard process by comparing a query customization with all customizations (with associated impacts) available in the repository. The set of retrieved customizations are ranked based on closeness to the query customization.

2 Process Difference and Impact Representations

In this section, we describe delta trees and impact trees, which are both derivations of process structure trees. A Process Structure Tree (PST) is a hierarchical representation of business process. PSTs are used to represent both standard and customized processes because there are established algorithms for detecting differences between trees [1]. The output of our PST difference detection algorithm is a delta tree. Impact trees are delta trees that have been annotated with impact information.

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 557–562, 2010.

2.1 Delta Tree

A delta tree is a PST difference representation that is annotated with customization operations. To create delta trees, we only consider the customization operations *insert* and *delete*. We do not consider the *move* and *relabel* operations because their effects can be replicated with combinations of delete and insert operations. *Customization tags* are labels on leaf nodes of a delta tree that represent customization operations (i.e. whether a node has been inserted or deleted), and *customization summary tags* are labels on interior nodes which summarize the changes that were made to its descendents. The algorithm for constructing a delta tree has three phases.

The first phase is a modification of the algorithm by Küster et. al. [7], which determines the differences between two PSTs and represents those differences with a Joint PST (JPST). The modification decomposes the move portion of the algorithm into a series of insertions and deletions. The second phase involves reducing the JPST by finding the *delta root*. The delta root is the least common uncustomized ancestor of all the nodes in the JPST that have a customization tag. Let ℓ be the least common ancestor of the two customization tagged nodes that are the farthest distance apart in JPST, J. Thus, ℓ is also the ancestor of all the customization tagged nodes in that Joint PST. There are, however, situations when ℓ could itself be a customization tagged node, and thus not the delta root. When this happens, the delta root is the parent of ℓ in the Joint PST. Once the delta root, ℓ, is found the subtree J/ℓ becomes the delta tree. The reduction procedure continues by finding all of the untagged nodes that are not ancestors of customization tagged nodes and removing them from the delta tree. The procedure recursively removes all of the untagged leaves from the delta tree. The last phase in constructing a delta tree is adding *customization summary tags* (CSTs) to each interior node. The CST of an interior node, x, denoted by $CST(x)$, can take on one of three different values:"+", "-", and "M". Let x be an interior node in a delta tree and v be a descendant of x.

Definition 1 (Insertion Summary Tag). *If there exist at least one descendent of x with a customization tag of $''+''$, and no descendant of x with a customization tag of $''-''$, then $CST(x) = '' +''$.*

Definition 2 (Deletion Summary Tag). *If there exist at least one descendent of x with a customization tag of $''-''$, and no descendant of x with a customization tag of "+", then $CST(x) = '' -''$.*

Definition 3 (Mixed Summary Tag). *If there is at least one descendant of x with a customization tag of "+", and at least one descendant of x with a customization tag of "-", then $CST(x) =$ "M''.*

2.2 Impact Trees

An *impact tree* is a delta tree that has user-specified nodes called *impact nodes*. An impact node characterizes the effect the set of change operations defined by

the customization tagged descendents have on a business. There is at least one impact node in an impact tree.

If a query delta tree matches an impact tree in the repository, then the impact associated with the impact nodes is the impact predicted for the query.

Definition 4. *Delta (Sub)Tree Compatibility. Let r be a node in impact tree $\hat{R}$, and X be the set of customization tagged descendents of r. Additionally, let q be a node in delta tree $\hat{Q}$ and Y be the set of customization tagged descendents of q. Subtree $\hat{R}/r$ is* incompatible *with subtree $\hat{Q}/q$, if $X \not\subseteq Y$. Otherwise, $\hat{R}/r$ is* compatible *with $\hat{Q}/q$.*

It should be noted that compatibility is directional. Therefore, if $\hat{R}/r$ is compatible with $\hat{Q}/q$, it does not imply that $\hat{Q}/q$ is compatible with $\hat{R}/r$. Based on the customization summary tags on delta trees, we can determine whether or not a subtree of an impact tree is compatible with a subtree of a delta tree. Given two trees R and Q, let r be in R and q and a be in Q. If a is an ancestor of q in Q and $id(a) = id(r)$, then a is an *r-matched ancestor* of q.

Property 1. Let $\hat{Q}$ and $\hat{R}$ be a query delta tree and an impact tree in the repository, respectively, created from the same standard process. Additionally, let q and r be interior nodes in $\hat{Q}$ and $\hat{R}$, respectively. Subtree $\hat{R}/r$ is compatible with $\hat{Q}/q$, if q does not have an r-matched ancestor and $CST(r) = CST(q)$ or q does not have an r-matched anscestor and $CST(q) = M$.

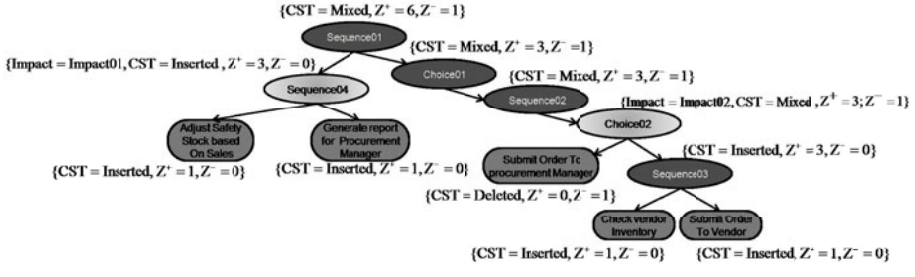

Fig. 1. Impact Tree Representation - Customized Process

3 Impact Prediction Algorithm

Our notion of tree matching is, as such, not new and is grounded in the methods noted in literature. But, the innovative rubber-banding of sub-trees/nodes as impact nodes for the customized processes in the repository is what differentiates our approach. This enables us to apply conditions to prune the search space by recognizing unfruitful paths. This allows us to speed up the performance. Our match starts with the basic query delta tree on the left hand side and the impact trees from a repository on the right hand side.

Algorithm 1. ImpactTreeTraversal($\hat{Q}$, $\hat{R}$)

```
r ← Root node of R̂ ;
q ← Root node of Q̂ ;
matched ← FALSE ;
if id(r) == id(q) then
    matched ← TRUE ;
if r is an impact node then
    ImpactTreeSearch(Q̂, R̂);
for each child node r_j ∈ children(R̂) do
    if matched == FALSE then
        ImpactTreeTraversal(Q̂, R̂/r_j);
```

The algorithm to predict impacts for a delta tree $\hat{Q}$ consists of three steps to be repeated for all impact trees in the repository.

Step 1 described in Algorithm 1 performs a pre-order traversal of all impact nodes r in $\hat{R}$, where $\hat{R}$ is an impact tree in the repository. For each impact node, $ImpactTreeSearch()$ (to be described in Step 2) traverses $\hat{Q}$ searching for a subtree matching the selected impact node in $\hat{R}/r$. If such matching subtree is found, the corresponding subtree, $\hat{Q}/q$, is annotated with the impacts associated with r. The algorithm terminates when there are no more impact nodes to be searched in $\hat{R}$.

In *Step 2*, ImpactTreeSearch(), which is illustrated in Algorithm 2, does a pre-order traversal of $\hat{Q}$ searching for a node q_i in $\hat{Q}$ such that $\hat{Q}/q_i$ and $\hat{R}/r_j$ are a *compatible* . Step 2 primarily checks the compatibility of the sub-trees and hands over the actual tree comparison to Step 3. The recursive comparison of $\hat{Q}/q_i$ and $\hat{R}/r_j$ will stop under two conditions: (1) all the nodes in $\hat{Q}$ have been searched and no compatible subtree was found, or (2) $\hat{R}/r_j$ is *incompatible* with $\hat{Q}/q_i$.

In *Step 3*, DeltaTreeSimilarityMatch() is invoked to compare compatible trees. The algorithm traverses the query and impact tree sub-trees to find similarity. When matching leaf nodes, only the nodes with the same id are matched. Let m be the number of leaf nodes in $\hat{Q}/q_i$ and let n be the number of leaf nodes in $\hat{R}/r_j$. Let k denote the number of *matched leaf nodes* between $\hat{Q}/q_i$ and $\hat{R}/r_j$. Then, the similarity match score M between these two sub-trees is:

$$\mathrm{M}(\hat{R}/r_j,\hat{Q}/q_i) = \min(k/m,\, k/n)$$

Once all the matching is complete, repository customizations are ranked based on the overall similarity score attributed to the parent query node. The impact lists associated with each of the ranked repository customizations are the predicted impacts of the desired customizations in the query process.

Algorithm 2. ImpactTreeSearch($\hat{Q}$, $\hat{R}$)

```
q ← Root of Q̂;
r ← Root of R̂;
if CST(q) == CST(r) then
    if id(q) == id(r) then
        DeltaTreeSimilarityMatch(q, r);
        return();
    else
        for each node q_i ∈ children(Q̂/q) do
            ImpactTreeSearch(Q̂/q_i, R̂);
            return();
if CST(q) == "M" then
    for each node q_i ∈ children(Q̂/q) do
        ImpactTreeSearch(Q̂/q_i, R̂);
else
    return();
```

4 Related Work

A large number of publications have addressed the problem of process matching, impact analysis, and tree comparison. However, they differ in domain, goal, medium, or approach.

We are not the first to leverage the PST representation for detecting the differences between process models. As example, the paper by Küsteret. al. [7] detects the differences between PSTs and represents those differences in a JPST, which is similar to our notion of delta trees. However, their work is concerned with obtaining a change log when one is not available. The paper by Eshuis and Grefen [3], which proposes a heuristic approach for matching BPEL processes represented in Program Structure Tree [5,6], is similar but, once again, the goal of their work (to find executable services exposing a certain behavior) is different than ours. The work of Dijkman et. al. [2] compares four approaches to the process similarity problem using a graph-based representation of process models. Their work is similar to ours in that they explore the process similarity problem. However, our approach differs in that we use a tree-based representation. The authors of Provop [4] present an approach for managing process variants by constraining the variations to select change operators such as insert, delete, move and modify. Te main motivation for our work is to improve the overall discoverability of variations on processes and also to predict the impact of these variations.

5 Conclusion

In this paper we have presented an algorithm to predict the impact of customizations made to standard business processes. We assumed that (a) all customizations

are to the same standard business processes and (b) a repository of such customizations exists. For a given customized process whose impact needs to be predicted, we searched the repository of existing customizations to find similar customizations for that prediction. The algorithm presented takes advantage of two characteristics of our problem domain (a) the set of operations to customize a standard process is constrained, and (b) not all changes result in impact worthy of consideration. Together, these characteristics limit the scope and speeds up the runtime of the algorithm.

For future research, we would like to explore if given a desired business impact to be achieved, could we predict the customizations that need to be made to a standard process by analyzing customizations with similar impacts in the repository. This could help IT service providers in assessing what needs to be done to a standard process to meet customer requirements.

References

1. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science 337(1-3), 217–239 (2005)
2. Dijkman, R., Dumas, M., Garcıa-Banuelos, L.: Graph matching algorithms for business process model similarity search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) Business Process Management. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
3. Eshuis, R., Grefen, P.: Structural matching of BPEL processes. In: Fifth European Conference on Web Services, ECOWS 2007, pp. 171–180 (2007)
4. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process lifecycle. In: ICEIS 2008 (2008)
5. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation, pp. 171–185. ACM, New York (1994)
6. Johnson, R.C.: Efficient program analysis using dependence flow graphs (1995)
7. Küster, J.M., 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)