Pseudocode of the Algorithm for Matching Business Process Workflows Across Abstraction Levels

Moisés Castelo Branco¹, Javier Troya², Krzysztof Czarnecki¹, Jochen Küster³, and Hagen Völzer³

¹ Generative Software Development Laboratory, University of Waterloo, Canada
WWW home page: http://gsd.uwaterloo.ca
² Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
³ IBM Research Zurich, Switzerland

1 Introduction

In this document we present the pseudocode of the algorithm presented in the paper submitted to the MODELS 2012 Conference, entitled Matching Business Process Workflows Across Abstraction Levels. In our matching algorithm, we assume that the models to be matched represent the same process, but at different levels of abstraction. We also assume that, although the models are intended to be consistent, inconsistencies can occur during their evolution and the models to be matched may include some inconsistencies. The aim of the algorithm is to automatically identify a correspondence among the models. A correspondence is defined by a set of correspondence links. To achieve that, the algorithm actually deals with the PST representations of the models. Leaves in a PST represent model elements, while inner nodes represent SESE regions. In this document, when referring to the elements in a PST, we will use the term model element to refer to leaves, region to refer to SESE regions and node to refer to both of them indistinctly. Thus, a correspondence link establishes a relation between two nodes in both PSTs.

2 Algorithm’s Pseudocode

The algorithm receives as input the two business process models (Modela and Modelb) whose correspondence we want to establish and the threshold values, f and l, controlling node’s similarity. It produces, as result, the PST representations (PSTa and PSTb) of the corresponding models as well as the set of correspondence links (corr) established among the PSTs’ nodes. The objects of type CorrsLink are pairs of nodes.

The algorithm is summarized in Listing 1. First, it initializes the output variables: the set of correspondence links is an empty set at the beginning (line 2) and the variables representing the PSTs are initialized with the result of the function buildPST, which takes a process model as input and returns its
PST’s representation (lines 3 and 4). Then, correspondence links established by
attribute (first phase of our algorithm) are inserted in \( \text{corr} \) (line 5). It is done by
the procedure \texttt{matchPSTs.Attribute}, shown in Listing 2. Then, correspondence
links determined by structure (second phase of the algorithm) are inserted in
\( \text{corr} \) (line 6) by calling the procedure \texttt{matchPSTs.Structure}, shown in Listing
3. Finally, the algorithm returns the PST’s representations of the input models
and the set of correspondence links among their nodes (line 7).

### Listing 1 Process models matching procedure

```plaintext
1: procedure BPMNMatching(in Model\( _a \), Model\( _b \) : Model; in \( f, l : \text{Real} \); out PST\( _a \), PST\( _b \) : PST; out corr : CorrsLink[ ])  
2:     corr ← \{emptyset\};  
3:     PST\( _a \) ← buildPST(Model\( _a \));  
4:     PST\( _b \) ← buildPST(Model\( _b \));  
5:     corr ← matchPSTs.Attribute(PST\( _a \), PST\( _b \), f, l, corr);  
6:     corr ← matchPSTs.Structure(PST\( _a \), PST\( _b \), corr);  
7:     return PST\( _a \), PST\( _b \), corr;  
8: end procedure
```

The first phase of the algorithm matches the nodes by content similarity,
comparing names and types of nodes. When comparing two model elements, the
string resulting from the concatenation of their names and types is compared,
as explained in the paper. As for regions, the string produced from the concate-
nation of names and types of all their model elements is compared. This phase
is called \textit{attribute matching} and is shown in Listing 2.

This procedure receives as input the PSTs of the models and the threshold
values. In adds correspondence links established in this phase to the \( \text{corr} \)
variable. The declared variables are \texttt{link}, of type \texttt{CorrsLink} (pair of nodes), and
\texttt{maxStringSim}, \texttt{leavesComp} and \texttt{stringSim}, of type \texttt{Real}.

In this phase, the roots of the PSTs are matched by default (line 4). Then, the
algorithm performs a depth-first traversal in \( \text{PST}_a \) (line 5) in order to establish
correspondence links with \( \text{PST}_b \). If the node in \( \text{PST}_a \) is a model element, i.e. a
leaf (line 6), it traverses \( \text{PST}_b \) (line 7) in order to find a model element with the
same name and type. If it finds it (line 8), a correspondence link is established
among them and included in \( \text{corr} \) (line 9).

If the node in \( \text{PST}_a \) is a region (line 12), the algorithm traverses \( \text{PST}_b \) (line
14) looking for a region to which establish a correspondence link. It keeps in
\texttt{maxStringSim} the maximum string similarity found (0 at the beginning, line
13), in \texttt{leavesComp} the comparison result of both regions in terms of leaves
matching (line 15), as explained in the paper, and in \texttt{stringSim} their string
similarity (line 16). As long as \texttt{leaveComp} and \texttt{stringSim} are bigger than their
corresponding threshold values (line 17), the algorithm compares if the string
similarity of the two nodes (\texttt{stringSim}) is bigger than the maximum string
similarity (\texttt{maxStringSim}) kept until the moment (line 18). If it is, the latter
variable is updated with the new string similarity (line 19) and the pair of regions is stored in the variable link (line 20). If the current region in $PST_a$, $n_a$, was matched with any region $n_b$ in $PST_b$, i.e., if the $maxStringSim$ is bigger than 0 (line 24), then the correspondence link stored in link is added to the set of corresponding links (line 25).

Eventually, this procedure returns the set of correspondence links (line 29).

The second phase of the algorithm matches the remaining unmatched nodes by structural similarity, comparing neighborhood matches among their parents and siblings. This phase is called structure matching and is shown in Listing 3.
Listing 3 PSTs matching by structure

1: procedure matchPSTsStructure(in PSTa, PSTb : PST; in/out corr : CorrsLink[ ])  
2:   NodesPair[ ] np;  
3:   np ← getUnmatchedNodes(PSTa, PSTb, corr);  
4:   for all (na, nb) in np do  
5:     if areMatched(na.parent(PSTa), nb.parent(PSTb)) and  
         areMatched(na.lefTsibling(PSTa), nb.lefTsibling(PSTb)) or  
         areMatched(na.rightSibling(PSTa), nb.rightSibling(PSTb)) or  
         na.isFirstChild(PSTa) and nb.isFirstChild(PSTb) or  
         na.isLastChild(PSTa) and nb.isLastChild(PSTb)  
     then  
6:       corr ← addLink(na, nb);  
7:   end if  
8: end for  
9: return corr;  
10: end procedure

This procedure declares a set of node pairs, np, of type NodesPair (line 2), where it keeps all the pairs of nodes in PSTa and PSTb that have not been matched with any node. These pairs are retrieve by the method getUnmatchedNodes and stored in the variable np (line 3). For every pair of nodes, na and nb, belonging to np, if their parents are matched, and if at least one sibling (the left or right one) are matched or if none of the siblings match but both na and nb are the last or first node in the child list (line 5), then a correspondence link among the nodes is added to corr (line 6).

Finally, this phase returns the updated set of correspondence links (line 9).