Matching Business Process Workflows across Abstraction Levels

Moisés Castelo Branco[1], Javier Troya[2], Krzysztof Czarnecki[1], Jochen Küster[3], and Hagen Völzer[3]

[1] Generative Software Development Laboratory, University of Waterloo, Canada
{mcbranco,kczarnec}@gsd.uwaterloo.ca
http://gsd.uwaterloo.ca
[2] Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
javiertc@lcc.uma.es
[3] IBM Research Zurich, Switzerland
{JKU,HVO}@zurich.ibm.com

Abstract. In Business Process Modeling, several models are defined for the same system, supporting the transition from business requirements to IT implementations. Each of these models targets a different abstraction level and stakeholder perspective. In order to maintain consistency among these models, which has become a major challenge not only in this field, the correspondence between them has to be identified. A correspondence between process models establishes which activities in one model correspond to which activities in another model. This paper presents an algorithm for determining such correspondences. The algorithm is based on an empirical study of process models at a large company in the banking sector, which revealed frequent correspondence patterns between models spanning multiple abstraction levels. The algorithm has two phases, first establishing correspondences based on similarity of model element attributes such as types and names and then refining the result based on the structure of the models. Compared to previous work, our algorithm can recover complex correspondences relating whole process fragments rather than just individual activities. We evaluate the algorithm on 26 pairs of business-technical and technical-IT level models from four real-world projects, achieving overall precision of 93% and recall of 70%. Given the substantial recall and the high precision, the algorithm helps automating significant part of the correspondence recovery for such models.

Keywords: BPMN Matching, Consistency Management, Change Extraction.

1 Introduction

A growing number of enterprises use Model-Driven Engineering (MDE) based on Business Process Modeling (BPM) to automate their business processes. BPM

typically requires collaboration of many stakeholders, including Business Analysts, Systems Analysts, IT Architects and Developers. The distribution of responsibilities among these roles usually results in the creation of several models of the same business process, each residing at a different abstraction level. These models range from business-oriented ones, which are technology-independent and easily understandable by business people, to IT-oriented ones, constructed by taking into consideration technical facilities of existing systems. Specialized modeling languages have been developed to represent such models. One such language, standardized by the OMG, is Business Process Modeling and Notation (BPMN) [11].

A key challenge in BPM is maintaining the consistency among these different models. Maintaining consistency is important in order to ensure that business-level process models are implemented correctly by executable models, and that the business-level models reflect the implemented processes correctly, for example, for auditing purposes. Checking and maintaining consistency between a business-level model and its IT-level counterpart requires knowing the correspondences between the activities in the first model and the activities in the other one. Unfortunately, such correspondences are often missing in practice since the models are created using different tools or languages. For example, business-level models are often created using some variant of BPMN, either in a dedicated BPMN tool or in a diagramming tool such as Visio, and the IT-level models are often created in executable workflow language BPEL, targeting a specific process execution engine. With the introduction of BPMN 2.0, both business- and IT-level models can be expressed using the same language, improving tool interoperability across levels of abstraction. Nevertheless, organizations having existing business- and IT-level models still face the challenge of establishing the correspondence among these models. For example, IT personnel at the Bank of Northeast of Brazil (BNB), our industry partner, has faced this challenge as part of a regulatory compliance project. In the absence of adequate tool support, this task is very tedious and time consuming.

This paper presents a heuristic matching algorithm for determining such correspondences. The algorithm is based on an empirical study of process models at BNB [1]. The study revealed frequent correspondence patterns between models spanning multiple abstraction levels, including adding or modifying the information on individual activities and changing the models structure, for example, by behavioral refactoring and adding IT-specific tasks. Consequently, our algorithm has two phases, first establishing correspondences based on similarity of model element attributes such as types and names and then refining the result based on the structure of the models. Our algorithm can recover complex correspondences relating whole process fragments of one model to such fragments in the other model—a capability needed in practice [1]. Previous work on process models has focused on either one-to-one correspondences between activities (e.g., [3]) or one-to-many correspondences relating activities and process fragments ([17]).

We evaluate the algorithm on 26 pairs of business-technical and technical-IT level models from four real-world projects, achieving overall precision of 93%

and recall of 70%. Given the substantial recall and the high precision, the algorithm helps automating significant part of the correspondence recovery for models spanning multiple abstraction levels.

The remainder of the paper is structured as follows: Section 2 provides background on BPM and important concepts used throughout the paper. Section 3 describes the heuristic algorithm using a running example. Section 4 presents the results of the empirical evaluation including threats to the validity of the work. Section 5 discusses related work on process model matching. Finally, Sect. 6 summarizes and concludes the paper.

2 Background

2.1 BPMN, SESE, and PST

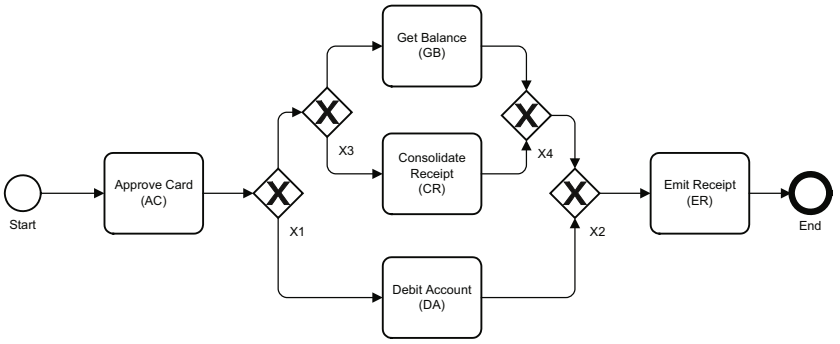
This paper assumes that the models to be matched are expressed in BPMN 2.0 [11]. BPMN 2.0 allows businesses to represent their internal business procedures in a graphical notation and communicate them in a standard way for both documentation and execution. Models expressed in other languages, such as BPMN 1.0 and BPEL, can be translated into BPMN 2.0 without adversely impacting the information used by our algorithm (cf. Sect. 4.1). BPMN inherits and combines elements from a number of previously proposed notations, including the Activity Diagrams component of the Unified Modeling Notation (UML).

Figure 1 shows two BPMN process models. We added shorter names in parentheses (e.g., *(AC)*) to later facilitate concisely representing correspondences between the models. The notation displays activities by rounded rectangles, events by circles, gateways by diamonds, and sequence flows by arrows. Each model has a start, usually modeled by a start event (e.g., *Customer inserts Card into ATM*), a flow of activities governed by decisions (e.g., *X1*), and an end point. A larger, realistic example is given elsewhere [1].

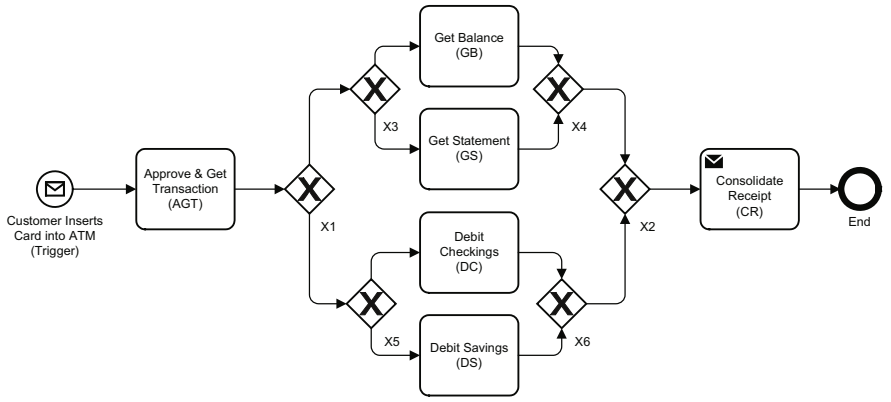
Any workflow graph (a BPMN process model in our case) can be uniquely decomposed into single-entry single-exit (SESE) regions [15]. Let $G = (N, E)$ be a workflow graph, where N is the set of nodes and E the set of edges. A SESE region $R = (N', E')$ is a nonempty subgraph of G, i.e., $N' \subseteq N$ and $E' = E \cap (N' \times N')$ such that there exist edges $e, e' \in E$ with $E \cap ((N \backslash N') \times N') = \{e\}$ and $E \cap (N' \times (N \backslash N')) = \{e'\}$; e and e' are called the *entry* and the *exit* edge of R, respectively. According to the formal definition, a SESE region is any region in the workflow graph that has a single entry at the beginning and a single exit at the end. In this way, an activity itself is a SESE region, and so is the whole workflow graph.

The Process Structure Tree (PST) for a BPMN process model is a tree representing the decomposition of the model into SESE regions [15], similar to the much older notion of a program structure tree [9]. Figure 2 shows the PSTs corresponding to the BPMN process models. There is a unique PST for each BPMN model. The root represents the whole process model since a process model is a SESE itself. Leaves represent model elements, i.e., activities, gateways and

events. Inner nodes represent SESE regions. In particular, the parent of a region R is the smallest region R' that contains R.



(a) Business Specification



(b) Technical Specification

Fig. 1. BPMN Models

2.2 Differences between Business and IT Process Models

Our target scenario involves matching business-level models specified by business analysts and the corresponding IT-level models implemented by IT specialists. IT specialists usually refine the original specification to meet technical requirements of the underlying IT infrastructure, such as invoking existing and new services, adding exception treatment, and changing the control flow to satisfy application protocols and optimize the execution. In previous work [1], we have studied over 70 models from BNB and interviewed their creators and maintainers, compiling a catalog of 11 recurrent patterns used to refine business-level models into IT-level models. These patterns include (i) adding or modifying properties of model

elements, such as changing the name or type of an activity or adding service call details, and (ii) changing the flow structure. The latter category includes behavioral refinement and refactoring and adding additional behavior, such as technical exception flow. An example from category (i) is the renaming and retyping of the empty start event *Start* (Fig. 1.a) into the message-driven event *Customer inserts card into ATM* (Fig. 1.b). An example from category (ii) is the refinement of the task *Debit Account* (Fig. 1.a) into the block consisting of the gateways *X5* and *X6* and two other tasks *Debit Checkings* and *Debit Savings* (Fig. 1.b). Examples of other patterns are given in the study [1].

3 Matching Algorithm

We assume that the models to be matched represent the same process, but at different levels of abstraction, as described in Section 2.2. We also assume that, although the models are intended to be consistent, inconsistencies can occur during their evolution. Thus, the models may include inconsistencies, such as order of activities switched during refinement or business-relevant activities added to the IT-level model but not reflected in the business-level model (see [1] for other examples).

The algorithm identifies a correspondence between two models residing at different abstraction levels. The algorithm operates on the PST representations of the models. As stated in Sect. 2, leaves in a PST represent *model elements*; inner nodes represent SESE regions, or *regions*, for short. The algorithm computes a *(model) correspondence*, which is a set of *correspondence links* among PST nodes; each link connects a single node in the PST of the first model with a single node of the PST of the other model. Thus, our algorithm is able to identify correspondence links of different cardinality with respect to model elements: *1:1* (link among two model elements or two regions with only one model element each), *1:n* (link between a region with one model element in the first PST and a region with more than one model elements in the second PST), and *m:n* (link between regions with more than one model element each).

Our algorithm has two phases: *attribute matching* and *structure matching*. The first phase deals with the search of correspondence links based on the attributes of model elements such as names and types; the second phase tries to find correspondence links based purely on the structures of the PSTs and the links established in the first phase. Note that the first phase also considers the structure of the PSTs since it matches both model elements and regions. The next subsection presents the similarity measures for model elements and regions. The following two subsections explain the two matching phases using the running example from Fig. 1. The pseudo-code of the algorithm is available at `http://gsd.uwaterloo.ca/matchingbpm`.

3.1 Matching Criteria for Model Elements and Regions

Our algorithm uses two attribute matching criteria for PSTs: one for matching individual model elements and another for matching regions. We adapted them

from previous work on matching source code represented as abstract syntax trees (ASTs) [6]. The original criteria use *bigram string similarity* to match the values of AST leaves and inner nodes. Fluri et al. [6] achieved better results for source code matching using Dice Coefficient with bigrams as string similarity compared to other measures such as the Levenshtein Distance [10]. In particular, the bigram-based similarity tolerates word re-orderings, which also occur in process refinement (e.g., ApproveCard vs. CardApproval).

We have adapted the original matching criteria by Fluri et al. to the process matching context, by using the information available in PSTs and refining the criteria based on experiments with sample models. In particular, we require exact matches for model elements and use bigram similarity only for inner nodes (regions). The reason is that process model elements have often relatively short names, and the names can be very similar, although representing completely different functions (e.g., ApproveCredit, ApproveContract, CreditAccount). The resulting criteria are as follows:

Matching criterion for model elements

$$match_e(n, m) \triangleq (type(n) = type(m)) \wedge (name(n) = name(m))$$

Matching criterion for regions

$$match_r(r, s) \triangleq (\frac{common(r, s)}{max(r, s)} \geq l) \wedge (sim_{2g}(value(r), value(s)) \geq f)$$

where

type returns the type of the model element as a numeric code, such as 0 for start event, 1 for task, 2 for exclusive gateway, etc.

name returns the name of the model element, for example: *Get Balance*, *Debit Savings*, etc.

sim$_{2g}$ calculates the bigram-based similarity of two strings [6]; it returns a numeric value between *0* and *1*, where *1* means that the strings are equal.

value returns the string formed by the concatenation of the names and types of all model elements of a region. Thus, similarity of names is emphasized, since types are short numeric codes and names are typically complete words.

common returns the number of pairs of model elements of the two regions that match exactly (i.e. $match_e$ is *true*).

max returns the maximum number of distinct pairs that could be matched (i.e., the number of all model elements in the smaller region).

f and ***l*** are thresholds controling the algorithm. We obtained the best results in our evaluation with *0.6* and *0.4*, respectively.

3.2 Attribute Matching

Let us explain the first phase by applying it to the PSTs in Fig. 2 obtained from the models in Fig. 1.

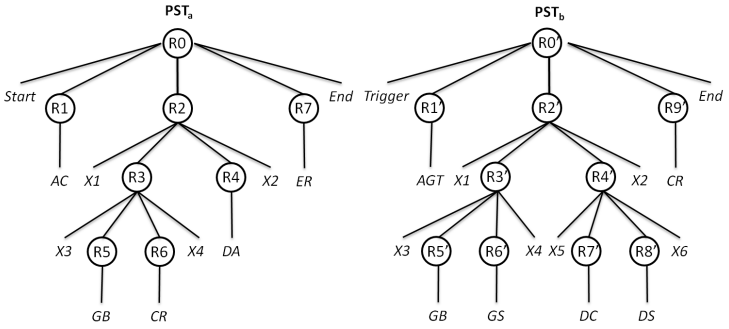


Fig. 2. PSTs representation of the business process models

First, the algorithm assumes that the roots of both PSTs correspond to each other. Then, the algorithm performs a depth-first traversal in one of the PSTs in order to establish correspondence links with the second PST. Starting with region *R1* in PST_a, it tries to find a corresponding region in PST_b. According to the matching criterion for regions (cf. Sect. 3.1), a necessary condition for a match is to satisfy the formula $\frac{common(R1,X)}{max(R1,X)} \geq l$ with any region X in PST_b. Since $R1$ has only one child (a model element), satisfying the formula requires finding a region in PST_b containing a model element with exactly the same name and type (matching criterion for model elements) as the activity *Approve Card*. Since there is none, the algorithm proceeds to region $R2$.

For $R2$, the algorithm finds $R2'$ in PST_b to satisfy the above formula ($\frac{5}{6} \geq 0.4$). The algorithm also checks that $sim_{2g}(value(R2), value(R2')) \geq f$ is satisfied. Assuming abbreviations, $value(R2)$ returns *X12X32GB1CR1X42DA1X22*; $value(R2')$ returns *X12X32GB1GS1X42X52DC1DS1X62X22*. Both strings have a similarity of around 0.65 (assuming full names). The algorithm then keeps on searching more matches for $R2$ in PST_b. The formula $\frac{common(R2,R3')}{max(R2,R3')} \geq l$ is also satisfied, returning $\frac{3}{4}$; however, the value obtained from the string comparison, 0.51, is smaller that f, so $R3$ is discarded as a match (see left figure in Fig. 3, where the top link is selected and the bottom one is discarded). No other region in PST_b satisfies the matching criterion with $R2$; however, if there were several matching regions in PST_b, the correspondence link would be established with the region with the highest string similarity to $R2$. If there are more than one region with the same highest string similarity to $R2$ (unlikely though, because copies are uncommon in process modeling), one of them is chosen arbitrarily.

The algorithm keeps traversing PST_a and establishes a correspondence link between $R3$ and $R3'$, since the string similarity value is 0.79 (right figure in

Fig. 3). $R5'$ in PST_b corresponds to $R5$, since the string similarity is 1. The same applies to $R6$ and $R9'$. There are no correspondence links for $R4$ and $R7$. Finally, the algorithm establishes correspondence links among model elements. In our example, correspondence links from $X1, X2, X3, X4, GB, CR$ and End in PST_a to the model elements with the same name in PST_b are created.

Figure 4 shows the complete set of correspondence links based on attribute matching, also indicating their model element cardinality. To avoid clutter, the links among model elements with the same name are not shown.

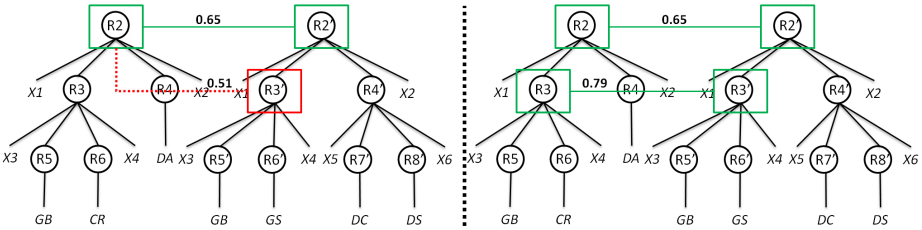


Fig. 3. Attribute matching phase step by step for $R2$ and $R3$

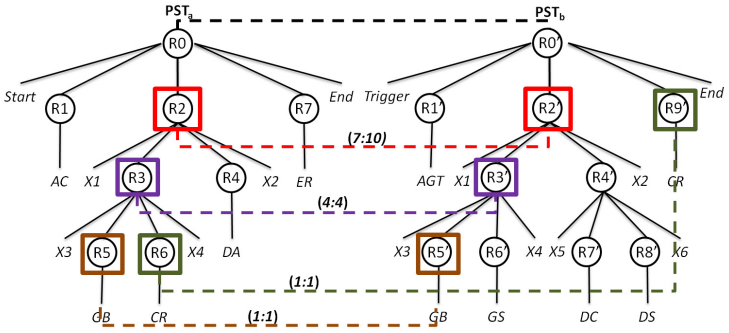


Fig. 4. Correspondence Links for the Attribute Matching Phase

3.3 Structure Matching

The second phase of the algorithm aims to match nodes that have not been matched in the first phase due to their different content. It does so by considering the location of the unmatched nodes in the PSTs and the correspondence links established so far. For example, consider regions $R4$ and $R4'$ in Fig. 4. Although they are dissimilar, it is likely, given the correspondence links so far, that they should be linked. The task of this procedure is to find such pairs of nodes and link them. The rule for finding node pairs to link is as follows. Let n_a and n_b be a pair of unmatched nodes. If the parents of n_a and n_b are linked, and if

at least one sibling (the left or right one) of n_a and n_b are linked, n_a and n_b should be linked, too. If none of the siblings are linked (possibly because they do not exist), we will also link the nodes if both n_a and n_b are the last or first node in the child list. According to these rules, the aforementioned regions $R4$ and $R4'$ should be matched, as their parents ($R2$ and $R2'$) and their left and right siblings ($R3$ with $R3'$ and $X2$ with $X2$) match. The same happens with $R1$ and $R1'$ since $R0$ matches with $R0'$ and $R2$ with $R2'$. This newly created correspondence link allows us to link the *Start* and *Trigger* events, too. All correspondence links established by both phases of the algorithm are shown in Fig. 5. As previously, correspondence links between model elements with the same name are not shown.

Complexity. Assume $n = max(|PST_a|, |PST_b|)$, where $|PST|$ is the number of regions. The cost of comparing the attributes of two regions is denoted by c and the cost of checking their structure similarity is s. The matching of all regions is in $O(n^2(c+s))$, that is, $O(n^2)$, since the algorithm compares each possible region pair.

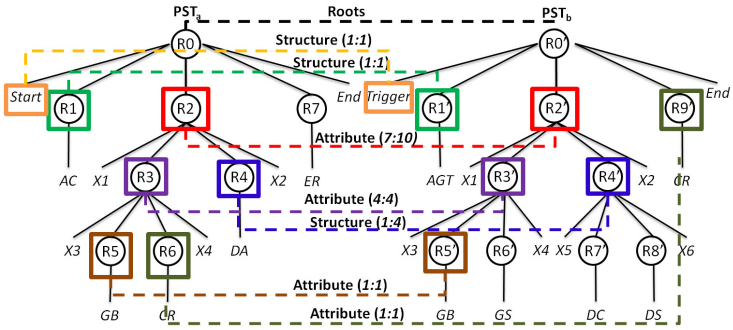


Fig. 5. Correspondence Links from Both Phases

4 Evaluation

We are interested in knowing the *precision* and *recall* of the presented algorithm when establishing correspondences among pairs of real-world process models across different levels of abstraction. Precision tells us whether the recovered correspondence links are correct and recall tells how large of a portion of the links the algorithm can recover. The following subsections present the methodology we have followed and the results.

4.1 Methodology

Objective. We want to evaluate the precision and recall of our algorithm. We define these measures for model correspondences (sets of correspondence links) between the PSTs. We refer to a model correspondence established by the domain experts as a *reference correspondence* (RC) and to a model correspondence established by our algorithm as a *computed correspondence* (CC). Given these sets, precision (P) and recall (R) are defined, respectively, as $P = \frac{|RC \bigcap CC|}{|CC|}$ and $R = \frac{|RC \bigcap CC|}{|RC|}$.

Subject Data. We used business process models taken from the Bank of Northeast of Brazil (BNB), a major financial institution in Brazil that is controlled by the federal government and oriented towards regional development. BNB has been using Business Process Modeling since 2007 in a development process based on the *Rational Unified Process*. The development process entails iterative and multi-staged model refinement, resulting in three types of process models (from higher to lower level of abstraction): business specifications, technical specifications, and executable processes. We had access to several projects developed as a result of this process and used them for evaluating the algorithm.

Table 1. BPM Projects

		Number of Models		
Project	Domain	Business	Technical	Implementation
P1	Customer Registration	2	2	2
P2	Credit Backoffice	6	6	6
P3	Credit Risk Assessment	2	2	2
P4	Procurement	3	3	3

We obtained four real BPM projects, containing 39 models in total. Table 1 shows, for each project, the number of models defined in each stage. Our target is to determine the correspondences between each corresponding pair of business and technical specifications and between the latter and executable implementations. Table 2 gives the total number of model elements for each level of abstraction.

Reference Correspondences. As reference correspondences, we use the correspondence links established manually by the domain experts (the bank's employees) who created and maintain the models. The reference correspondences in one of the projects was already established for auditing and regulatory compliance purposes, and reused here. The correspondences for the other projects were established as part of this research.

Table 2. Model Sizes

		Total Numbers		
		Tasks	Gateways	Events
P1	Business Spec.	59	38	25
	Technical Spec.	78	46	36
	Implementation	123	56	43
P2	Business Spec.	47	46	18
	Technical Spec.	95	48	23
	Implementation	107	52	31
P3	Business Spec.	17	8	6
	Technical Spec.	19	10	8
	Implementation	22	6	9
P4	Business Spec.	13	10	11
	Technical Spec.	18	12	15
	Implementation	25	14	17

Algorithm Implementation. We have implemented the algorithm in Java as an Eclipse feature, on top of the SOA Tools Platform BPMN Modeler [13]. Since the original models from BNB were created using IBM's WebSphere Process Modeler, we needed to recreate them to run our tool.

4.2 Results

Table 3 shows the results of our evaluation. We matched pairs of models at different levels of abstraction from each project. Concretely, we compared business and technical models, and the latter and IT implementation models. Column "Pair Type" indicates the type of models compared in each row. Column "Corresp - RC" gives the total number of correspondence links identified by the domain experts. Column "Corresp Type" shows the numbers obtained in each phase of the algorithm."Total" represents the net result of the two phases. Notice that the correspondence links do not overlap between the phases. Column "Correct" specifies the number and the cardinality of correspondence links that our algorithm was able to identify, in each phase, from those in the reference correspondence, including their cardinalities. Columns "FP" and "FN" give the number of false positives and negatives, respectively. False positives are those correspondence links that our algorithm finds but do not belong to the set of reference correspondence links. False negatives are those correspondence links included in the reference correspondence that our algorithm is unable to detect. In each phase, "FP" and "FN" are computed with respect to the complete reference. Finally, "Prec" gives precision, followed by column "Recall".

If we consider the correspondence links all together—as if they had been extracted from only one pair of models—we have 622 reference links found manually by the domain experts. Out of these 622, our algorithm was able to correctly identify 438, with 32 false positives and 184 false negatives, yielding overall recall of 70% and precision of 93% Among the reference links, 117 had cardinality type

Table 3. Correspondences among Models across Different Abstraction Levels. B: Business; T: Technical; IT: Information Technology; Corresp - RC: Reference Correspondence; FP: False Positives; FN: False Negatives; Prec: Precision.

Project	Pair Type	Corresp - RC	Corresp Type	Correct (*1:1*; *1:n*; *m:n*)	FP	FN	Prec	Recall
1	B–T	30	Attribute	16 (15;0;1)	0	14	100%	53%
			Structure	4 (1;2;1)	2	26	67%	13%
			Total	20 (16;2;2)	2	10	91%	67%
1	T–IT	42	Attribute	28 (26;0;2)	0	14	100%	67%
			Structure	3 (2;1;0)	2	39	60%	7%
			Total	31 (28;1;2)	2	11	94%	74%
2	B–T	138	Attribute	95 (90;0;5)	0	43	100%	69%
			Structure	8 (6;2;0)	4	130	67%	6%
			Total	103 (96;2;5)	4	35	96%	75%
2	T–IT	240	Attribute	136 (127;0;9)	0	104	100%	57%
			Structure	18 (10;5;3)	12	222	60%	8%
			Total	154 (137;5;12)	12	86	93%	64%
3	B–T	32	Attribute	22 (21;0;1)	0	10	100%	69%
			Structure	4 (4;0;0)	2	28	67%	13%
			Total	26 (25;0;1)	2	6	93%	81%
3	T–IT	44	Attribute	32 (32;0;0)	0	12	100%	72%
			Structure	2 (2;0;0)	5	42	29%	5%
			Total	34 (34;0;0)	5	10	87%	77%
4	B–T	42	Attribute	24 (23;0;1)	0	18	100%	57%
			Structure	6 (3;3;0)	3	36	67%	14%
			Total	30 (26;3;1)	3	12	91%	71%
4	T–IT	54	Attribute	36 (36;0;0)	0	18	100%	67%
			Structure	4 (2;1;1)	2	50	67%	7%
			Total	40 (38;1;1)	2	14	95%	74%

1:n and 89 had the cardinality type *m:n*. From these, the algorithm identified correctly 14 (12%) and 24 (27%), respectively.

The overall precision ranges between 87%-96%. None of the false positives is obtained in the attribute matching phase. This is very positive since a large portion of the reference correspondence links is recovered in this phase. The number of false positives in the structure matching phase is quite large compared to the number of reference correspondence links of purely structural nature. This would be a serious problem in a situation where models have many such purely structural correspondences. We identified two causes for having so many false positives. The principal cause is the presence of non-hierarchical refinement patterns [1]. For example, in one B-T pair of the Project 2, there is an activity in the business specification that corresponds to 3 activities in the technical specification. Each of the 3 activities belongs to a different region in the technical specification. The algorithm cannot identify such kind of correspondence; the second phase matched the region containing the business activity to an incorrect region in the technical specification. Another cause is matching nodes that are the last or first node in the child list. Although this is reasonable in many cases, it also leads to incorrect matches. For example, this rule produced a false positive in one T-IT pair of the Project 3. Unfortunately, without extra information (e.g., IDs or annotations) it is likely not possible to decide whether or not to match the regions in many of such cases.

The relatively high number of false negatives —20%-40%—is caused mainly also by the presence of non-hierarchical refinements, which occurred in all the projects. We believe that these numbers can be reduced by applying a pattern matching technique for describing and finding instances of well-known or organization-specific non-hierarchical refinements patterns, which we leave for future work.

4.3 Threats to Validity

This section summarizes the potential threats that may impact the internal and external validity [4] of the empirical results.

Threats to External Validity. A potential threat to external validity is that the models used in the evaluation may not be representative of those occurring in other realistic settings. While the models used here come from real-world projects, the algorithm should be tested additionally on models from other organizations and domains.

Threats to Internal Validity. The main threat is the re-modeling of the BNB's business process models to be processed by our tool. BNB applies IBM tools that use an extension of BPMN. Some features of the BNB models that are not covered in BPMN had to be omitted during the translation. This threat was minimized by checking with the domain experts that the BMPN models obtained after the simplification were largely equivalent to the original models.

5 Related Work

Matching of models is a standard topic in MDD. For example, UMLDiff is a prominent approach for matching UML models [19]. However, effective matching requires heuristics that are usually notation and application specific. Our work focuses on finding such heuristics for matching business process models across levels of abstraction. Discovery of effective heuristics usually requires studying the differences among such models. In this context, Dijkman [2] presents a classification of frequently occurring differences between similar business processes in general. Our previous study [1] provided an in-depth analysis of such differences between models targeting different levels of abstraction.

As in our approach, the work by Dijkman et al. [3] aims to realize business process models alignment based on lexical matching (similar to our attribute matching) and structural matching. They report recall of 60% and precision of 89% for their approach. However, their algorithm only captures 1:1 correspondences between model elements. Our algorithm also identifies correspondences between SESE regions, which is necessary for matching models at different levels of abstraction.

Weidlich et al. present ICOP in [17], a framework based on matchers to identify correspondences between process models. They represent the models using

Refined Process Structure Trees (RPSTs) [14,12] rather than PSTs. In RPSTs, regions can have more than one entry and more than one exit. The approach by Weidlich et al. deals both with 1:1 and 1:n matches. Our approach additionally relates regions to regions, which are examples of m:n matches. They report recall of 60% and precision of 80%.

Ehrig et al. [5] propose a set of similarity measures for process models, for example, in order to discover existing related process models in repositories. However, the approach does not establish fine grained correspondence links like in our approach. The authors do not discuss recall and precision of the approach.

Several related works deal with comparing process models (e.g., [7,8]), checking their consistency (e.g., [16]), and their synchronization (e.g., [18]). All these works assume that model correspondences have been previously established.

Table 4 summarizes our contribution in the light of the related works.

Table 4. Related BPM Matching Approaches. + : Feature Provided; – : Feature not Provided; NA : Not Available.

	Approach			
Feature	Weidlich et al. [17]	Dijkman et al. [3]	Ehrig et al. [5]	Our Approach
Match Activity Attributes	+	+	+	+
Match Model Structure	+	+	–	+
Match Activity-Activity (1:1)	+	+	+	+
Match Activity-SESE (1:n)	+	–	–	+
Match SESE-SESE (m:n)	–	–	–	+
Do not Require Model Element IDs	+	+	+	+
Support Activity Inserts and Deletes	+	+	+	+
Support Activity Moves	+	+	–	+
Support Activity Renaming	+	+	–	+
Support Activity Copies	+	+	–	–
Overall Precision	80%	89%	NA	93%
Overall Recall	60%	60%	NA	70%

6 Conclusions

We have presented an algorithm to automatically detect correspondences between BMPN process models across levels of abstraction. The algorithm operates over the PSTs of the input models in two phases. The first phase matches the PST nodes using region and model element matching criteria adapted from previous work on matching ASTs. The second phase establishes additional correspondences based on the position of the nodes in the PSTs.

We evaluated our algorithm on 26 pairs of business-technical and technical-IT level models from four real BPM projects, achieving overall precision of 93% and recall of 70%. Given the substantial recall and the high precision, the algorithm helps automating significant part of the correspondence recovery for such models.

The evaluation revealed that the algorithm is not able to detect certain kinds of complex correspondences. We believe that this limitation could be addressed in future work by extending the algorithm with an additional phase based on general and project-specific refinement patterns.

Acknowledgment. This work was partially supported by an IBM PhD CAS Fellow Scholarship and by Spanish Research Projects TIN2008-03107 and TIN2011-23795. We would like to thank the Bank of the Northeast of Brazil (Banco do Nordeste – BNB) for providing the case study as well as valuable requirements and feedback on the design of the algorithm.

References

1. Castelo Branco, M., Xiong, Y., Czarnecki, K., Küster, J., Völzer, H.: An Empirical Study on Consistency Management of Business and IT Process Models. Technical Report GSDLAB-TR 2012-03-22, Generative Software Development Laboratory, University of Waterloo, Waterloo (2012), http://gsd.uwaterloo.ca/reportstudybpm
2. Dijkman, R.: A Classification of Differences between Similar Business Processes. In: EDOC 2007, pp. 37–47. IEEE Computer Society, Washington, DC (2007)
3. Dijkman, R., Dumas, M., Garcia-Banuelos, L., Kaarik, R.: Aligning Business Process Models. In: EDOC 2009, pp. 45–53. IEEE (September 2009)
4. Easterbrook, S.M., Singer, J., Storey, M., Damian, D.: Selecting Empirical Methods for Software Engineering Research. In: Guide to Advanced Empirical Software Engineering, pp. 285–311. Springer (2007)
5. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: APCCM 2007, pp. 71–80. Australian Computer Society, Inc., Darlinghurst (2007)
6. Fluri, B., Wursch, M., Pinzger, M., Gall, H.: Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. IEEE Transactions on Software Engineering 33(11), 725–743 (2007)
7. Gerth, C., Luckey, M., Küster, J.M., Engels, G.: Detection of Semantically Equivalent Fragments for Business Process Model Change Management. In: SCC 2010, pp. 57–64. IEEE Computer Society, Washington, DC (2010)
8. Gerth, C., Luckey, M., Küster, J.M., Engels, G.: Detection of Semantically Equivalent Fragments for Business Process Model Change Management. Tech. Rep. IBM Research Report RZ 3767, IBM Research, Zurich, Switzerland (2010), http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/rz3767.pdf
9. Johnson, R., Pearson, D., Pingali, K.: The Program Structure Tree: Computing Control Regions in Linear Time. In: SIGPLAN Conference on Programming Language Design and Implementation (1994)
10. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
11. Object Management Group: Business Process Model and Notation (BPMN) Version 2.0, http://www.omg.org/spec/BPMN/2.0/
12. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Proceedings of the 7th International Conference on Web Services and Formal Methods, WS-FM 2010, pp. 25–41. Springer, Heidelberg (2011)
13. SOA Tools Platform: Eclipse BPMN Modeler, http://eclipse.org/projects/project.php?id=soa.bpmnmodeler
14. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008)

15. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
16. Weidlich, M., Dijkman, R., Weske, M.: Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 78–94. Springer, Heidelberg (2010)
17. Weidlich, M., Dijkman, R.M., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010)
18. Weidmann, M., Alvi, M., Koetter, F., Leymann, F., Renner, T., Schumm, D.: Business Process Change Management based on Process Model Synchronization of Multiple Abstraction Levels. In: Proceedings of SOCA 2011. IEEE Computer Society (2011)
19. Xing, Z., Stroulia, E.: Umldiff: an algorithm for object-oriented design differencing. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005, pp. 54–65. ACM, New York (2005), http://doi.acm.org/10.1145/1101908.1101919