The ICoP Framework: Identification of Correspondences between Process Models

Matthias Weidlich[1], Remco Dijkman[2], Jan Mendling[3]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
matthias.weidlich@hpi.uni-potsdam.de
[2] Eindhoven University of Technology, The Netherlands
r.m.dijkman@tue.nl
[3] Humboldt-Universität zu Berlin, Germany
jan.mendling@wiwi.hu-berlin.de

Abstract. Business process models can be compared, for example, to determine their consistency. Any comparison between process models relies on a mapping that identifies which activity in one model corresponds to which activity in another. Tools that generate such mappings are called matchers. This paper presents the ICoP framework, which can be used to develop such matchers. It consists of an architecture and re-usable matcher components. The framework enables the creation of matchers from the re-usable components and, if desired, newly developed components. It focuses on matchers that also detect complex correspondences between groups of activities, where existing matchers focus on 1:1 correspondences. We evaluate the framework by applying it to find matches in process models from practice. We show that the framework can be used to develop matchers in a flexible and adaptable manner and that the resulting matchers can identify a significant number of complex correspondences.

1 Introduction

Organisations compare business process models to identify operational commonalities and differences. Such comparisons are, for example, necessary when organisations merge and need to determine and resolve the differences between their operations, and when an organisation needs to check whether its operations conform to an company-wide or industry-wide standard. The first step when comparing business process models is always to determine which activities in one business process model correspond to which activities in the other. This step is called the *matching* step and can be supported by tools that are called *matchers*. This paper presents a framework that can be used to develop such matchers.

A major challenge for matchers is that business process models often do not use the same level of detail and the same words to describe activities. For example, there is a problem with level of detail if one business process model contains an activity 'Check Invoice', whereas the other describes the same activity using a sequence of 'Verify Customer Data', 'Decide on Correctness of Data', and 'Approve Invoice'. This problem relates to refinement and meronymy. There is a problem with the words used to describe an activity if there are two labels 'Check Invoice' and 'Verify Bill' that point to the same activity, although using very different words. These are problems of synonymy and homonymy. This problem of heterogeneous representation and description is of striking relevance in practice [1, 2], because organisations usually do not align the way in which they describe their business processes. It also is the major reason why the comparison of related processes requires an extensive amount of manual preprocessing.

Therefore, this paper presents automated assistance for this 'preprocessing' step, by proposing a re-usable framework for identifying correspondences between activities in one process and equivalent activities in a similar process, while taking into account that equivalent activities may be modelled at different levels of granularity, have different labels, and have different control-flow relations to other activities. The framework is specifically tailored to also deal with complex 1:n matches (i.e., each activity can be matched to an arbitrary number of other activities), where existing matchers focus on elementary 1:1 matches (i.e., each activity can be mapped to at most one other activity). The framework consists of an architecture and a set of re-usable matcher components. Matchers can be developed in the framework by composing them from existing components and, if desired, newly developed ones. We present matchers that we implemented within the framework, and we evaluate them using models from practice. Our contribution is a framework with re-usable components to automatically find both 1:1 and 1:n matches between activities from similar business processes.

While this contribution is significant to the process model matching field, it also has implications for schema matching in general [3]. The research area of schema matching covers a broad set of techniques, including structural analysis and natural language processing to automatically identify the elements of one schema (which are activities of a process model in our case) that match to those of a second schema. Unfortunately, there has been a predominant focus on 1:1 matches in the schema matching community, such that *'1:n and n:m mappings [..] are currently hardly treated at all'* [3]. Although there are notable exceptions, like iMAP [4], these techniques cannot be applied in our context as they have been tailored for data models and partially also use the extension of a database. We further discuss this issue when reviewing related work in section 5.

The paper is structured as follows. Section 2 describes the background of the problem by an example of process models with activity correspondences. Section 3 introduces the ICoP framework for developing matchers. It describes both the framework itself and the techniques that are implemented as components of the framework. Section 4 presents the evaluation of the framework. Section 5 assesses our contribution in the light of related work. Section 6 concludes.

2 Background

The relevance of finding matches in pairs of corresponding process models has been described before [1, 2]. In this section, we aim to illustrate the problem and

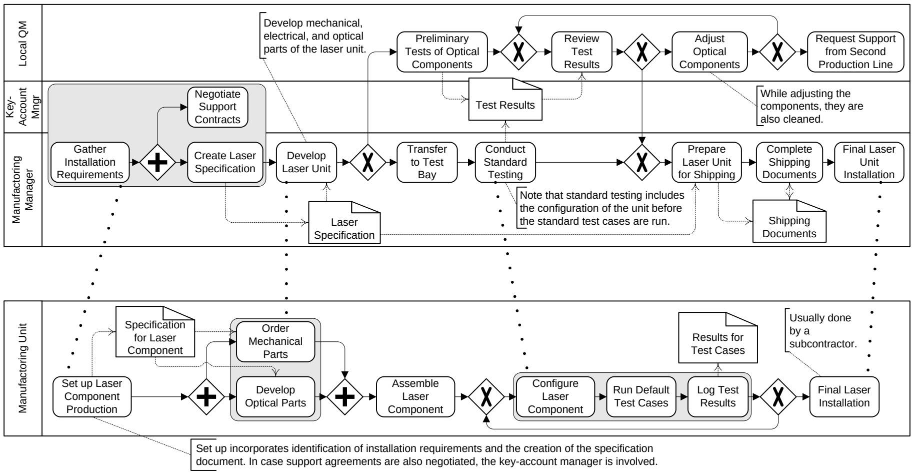

Local QM
Key-Account Mngr
Manufactoring Manager
Develop mechanical, electrical, and optical parts of the laser unit.
Preliminary Tests of Optical Components
Review Test Results
Adjust Optical Components
Request Support from Second Production Line
Negotiate Support Contracts
Test Results
While adjusting the components, they are also cleaned.
Gather Installation Requirements
Create Laser Specification
Develop Laser Unit
Transfer to Test Bay
Conduct Standard Testing
Prepare Laser Unit for Shipping
Complete Shipping Documents
Final Laser Unit Installation
Laser Specification
Note that standard testing includes the configuration of the unit before the standard test cases are run.
Shipping Documents
Manufactoring Unit
Specification for Laser Component
Order Mechanical Parts
Develop Optical Parts
Set up Laser Component Production
Assemble Laser Component
Configure Laser Component
Run Default Test Cases
Log Test Results
Results for Test Cases
Usually done by a subcontractor.
Final Laser Installation
Set up incorporates identification of installation requirements and the creation of the specification document. In case support agreements are also negotiated, the key-account manager is involved.

Fig. 1. Two BPMN process models with one 1:1 and three 1:n correspondences

the underlying combinatorial challenges using a pair of example processes. We also introduce terminology that we will use throughout the paper.

Figure 1 shows the operations of a company that produces lasers. The upper model depicts the process in a way that hand-overs between different departments can easily be traced. The lower process focusses more on the operational details as being conducted in the manufacturing unit. Roughly speaking, both processes describe that first the component production is set up, then the laser units are developed, assembled, tested, prepared for shipment, and finally installed at the sight of the customer. There are *matches* of different complexity between the activities of the two models. In the general case, a match refers to a correspondence, which is an element of the powerset of activities of a first model times the powerset of activities of a second model. Thus, a match is denoted by a tuple (A_1, A_2) of two sets of activities. A match (A_1, A_2) is called *elementary match*, if $|A_1| = |A_2| = 1$. An example are the 'Final Laser Unit Installation' (FLUI) and the 'Final Laser Installation' (FLI) activities. We say that FLUI matches or corresponds to FLI. This match pair can be easily identified due to the syntactic similarity of the labels and the same position at the end of the processes.

Figure 1 also highlights three complex matches by dotted lines. A match (A_1, A_2) is called *complex* if at least one activity set in the pair contains more than one element, i.e., $|A_1| > 1$ or $|A_2| > 1$. The first complex match on the left-hand side shows that 'Set up Laser Component Production' of the lower model describes the same activities as the set of 'Gather Installation Requirements', 'Negotiate Support Contract', and 'Create Laser Specification' in the upper model, but at a different level of granularity. Thus, it is a 1:3 match as there is one activity corresponding to three activities in the upper model.

The goal of automatic process model matching is to find all those matches that are meaningful. Usually, there are conditions that a collection of meaningful

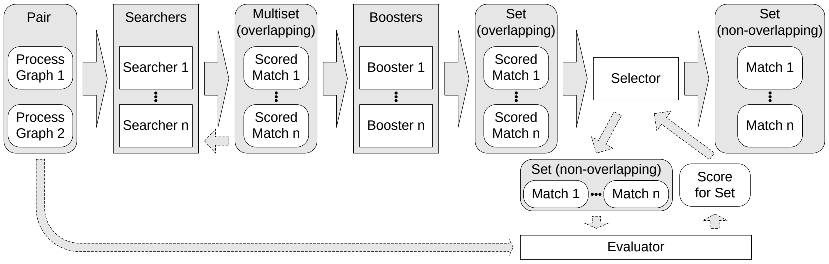


Fig. 2. The architecture of the ICoP framework

matches has to obey, for instance, that the matches do not *overlap*. That is, for every pair of matches (A_1, A_2) and (A_3, A_4) in the mapping it holds $A_1 \cap A_3 = \emptyset$ and $A_2 \cap A_4 = \emptyset$. We call this collection of matches a *mapping*. From a structural point of view, a mapping is an element of the powerset of matches.

Identification of complex matches imposes serious challenges. For the case of 1:1 matches, it might be feasible to analyse the whole set of potential matches, which corresponds to the Cartesian product of activities of two process models. In contrast, the possibility of 1:n matches increases the size of the set of potential matches significantly. For two process models with n and m activities, respectively, the number of 1:x matches is given by $n * \binom{m}{x} + m * \binom{n}{x}$. Here, the binomial coefficient $\binom{n}{x}$ defines the number of x-element subsets of an n-element set. For instance, the two process models depicted in Fig. 1 consist of 13 and 8 activities, respectively, which, in turn, yields 104 potential 1:1 matches. However, considering the possibility of 1:2 and 1:3 matches in addition increases the set of potential matches by $13 * \binom{8}{2} + 13 * \binom{8}{3} = 1092$ matches for one direction, and $8 * \binom{13}{2} + 8 * \binom{13}{3} = 2912$ matches for the other direction. One might argue that for the case of process models, the value of x might be bound by some threshold instead of the number of nodes of the respective model. However, our experience shows that this threshold must not be set too low (we encountered up to 1:9 matches), such that the amount of possible combinations still hinders any attempts to analyse the whole set of potential matches. These observations call for adequate search heuristics and an architecture that evaluates efficiently.

3 The ICoP Framework

This section introduces the ICoP framework for automatic derivation of matches between two process models. Section 3.1 discusses the overall architecture. We elaborate on the four types of components, namely searchers, boosters, evaluators, and selectors, in detail and present exemplary realisations in Sections 3.2 to 3.5.

3.1 Architecture

The overall architecture of the ICoP framework stems from the observation that the number of (1:n) matches between two business process models is potentially large and therefore it is not feasible to explore all possible matches exhaustively. Instead, the ICoP framework proposes a multi-step approach, which is illustrated in Fig. 2. Given two process models, *searchers* extract potential matches based on different similarity metrics and heuristics for the selection of activities. The result of the search stage is a multiset of matches due to the possibility of multiple searchers identifying the same potential matches. Each match is assigned a *match score*, which results from the scoring function implemented by the searcher to select potential matches. Note that a searcher may use the knowledge about potential matches that have been identified by other searchers already.

After completion of the search stage, the scored potential matches are conveyed to *boosters*. These components 'boost' the matches that are returned by the searchers using certain heuristics in order to aggregate matches, remove matches, or adapt the score of a match. As part of the boosting stage the *multiset* of potential matches is transformed into a *set* of potential matches by aggregating matches that have been identified by multiple searchers.

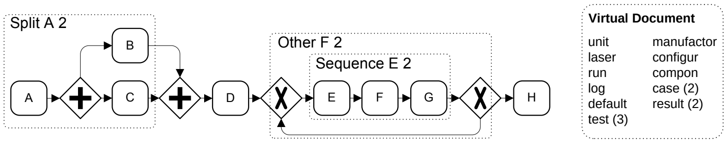
Subsequently, a *selector* builds up the actual mapping from the set of potential matches. That is, it selects the best matches from the set of potential matches, in such a way that the constraint that matches are not overlapping is satisfied. The selection of the best matches can be guided by two kinds of scores. On the one hand, the individual match scores of the potential matches can be exploited. On the other hand, an *evaluator* can be utilised, which assigns a single *mapping score* to a mapping. An evaluator may use knowledge about the original process models to compute this score. The selection is an iterative process. In each iteration the selector selects a set of matches, the evaluator computes the score for this set, upon which the selector either decides to modify the set of matches and continue the selection, or to complete the selection. Once the selection procedure completes, the selector produces the final mapping between elements of the process models.

3.2 Match Searchers

Searchers identify potential 1:1 and 1:n matches between two process models along with a score that indicates the quality of the match. Such a match is denoted by (A_1, A_2, s) with s being the match score based on the similarity of the matched activities. Therefore, we will also refer to the similarity score of a match. The similarity of a match can be determined based on various aspects, including the labels or descriptions of the matched activities and structural or behavioural relations between those activities.

We now introduce four searchers that have been implemented in the ICoP framework. Although a similarity score for the activity labels is at the core of all searchers, they incorporate similarity metrics for different aspects of labelling.

Similar Label Searcher. The purpose of this searcher is to identify straightforward 1:1 matches based on a high syntactic similarity of activity labels. It computes the Cartesian product of activities of two process models and selects all pairs of activities for which the string edit similarity of their labels is above a given threshold. For two strings s_1 and s_2 the string edit similarity is defined as $sim(s_1, s_2) = 1 - \frac{ed(s_1,s_2)}{\max(|s_1|,|s_2|)}$ with $ed(s_1, s_2)$ as the string edit distance, i.e., the



(a) (b)

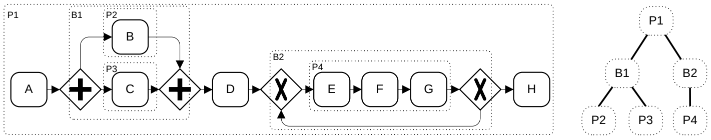
Fig. 3. (a) groups of activities for the lower model of Fig. 1, (b) virtual document for 'Sequence E 2'

minimal number of atomic character operations (insert, delete, update) needed to transform one string into another [5]. As the string edit similarity is a rather strict criterion (different orders of words and linguistic phenomena such as synonymy are neglected), the potential matches are identified with high confidence, such that the initial score for these matches is set to one by the searcher. For the scenario illustrated in Fig. 1 and a threshold of 0.8, for instance, the searcher identifies the match between the activities describing the installation of the laser. Clearly, the runtime complexity of this searcher depends solely on the number of activities of the respective process models.

Distance Doc Searcher. This searcher follows a two step approach in order to identify potential 1:n matches. First, activities of both process models are grouped heuristically. Second, the similarity between such a group of activities in one model and all single activities in the other model is assessed. Assuming that it is more likely that activities that are closer to each other should be in the same group, we use the graph distance to group activities. The graph distance between two activities is the number of edges on the shortest path from one activity to the other. Given a base activity and a distance, we look for four types of groups:

- Sequences, which are determined by a base activity and the activities on a directed path of the given length (distance) from the base activity.
- Splits, which are determined by a base activity and the activities that can be reached from the base activity and that are within the given distance. The base activity can or cannot be considered in such groups, depending on whether the choice leading to the split is or is not modelled explicitly.
- Joins, which are determined by a base activity and the activities from which the base activity can be reached and that are within the given distance. The base activity can or cannot be considered in such groups.
- Others, which are the groups that consist of all activities that are within the given distance of a base activity (not considering the direction of edges). In contrast to sequences, these activities are not necessarily on a path.

For the lower model of Fig. 1, Fig. 3(a) shows examples of a group that is a sequence with base 'E' and distance 2, a split with base A and distance 2, and an 'other' with base F and distance 2. The distance doc searcher identifies all groups of activities in a process model by taking each of the activities as a basis and creating each possible type of group for each possible graph distance value.



(a) (b)

Fig. 4. (a) RPST fragments for the lower model of Fig. 1, (b) the corresponding RPST

A maximum distance value can be set as a parameter. The groups are collected in a set to avoid duplication.

Once all groups of activities have been identified, the notion of ‘virtual documents’ is used to score their similarity. Virtual documents were introduced for aligning ontologies [6]. A virtual document of a node consists of the words from all textual information that is related to that node. Given two virtual documents, their similarity can be calculated based on their distance in a vector space, in which the dimensions are the terms that appear in the documents and the values for the dimensions are computed using term frequency [7]. In our setting, a virtual document for an activity consists of the terms that are derived from the activity label and, if this information is available, the labels of the roles that are authorized to perform the activity, the assigned input and output data, and a textual description of the activity. For a group of activities, the virtual document is derived by joining the documents of the respective nodes. Note that the creation of virtual documents includes a normalization of terms, filtering of stop-words, and term stemming [8]. Fig. 3(b) illustrates the terms of the virtual document for the group ‘Sequence E 2’. These terms originate from the activity labels, the label of the associated role, and the data output of one activity, while we also applied stop-word filtering and term stemming. Note that the runtime complexity of this searcher is heavily influenced by one parameter besides size and structure of the models. The maximal graph distance value for grouping activities might increase complexity significantly.

Fragment Doc Searcher. This searcher is similar to the distance doc searcher, except that it relies on the Refined Process Structure Tree (RPST) [9] for grouping activities. The RPST parses a process model into a hierarchy of fragments with a single entry node and a single exit node. Fig. 4(a) depicts these fragments for the lower process model in Fig. 1. The fragments are defined in such a way that they do not overlap. Consequently, they form a tree-structure, as illustrated in Fig. 4(b) (cf., [9]). We leverage the hierarchy of fragments in order to select the groups of activities that will be considered by the searcher. Starting with the leaf fragments of the RPST, the tree is traversed upwards up to a height that is given as a parameter. All traversed fragments are considered by the searcher that creates two virtual documents for each fragment, one containing all activities of the fragment and one containing all activities except those that are boundary nodes of the fragment. As in case of the Distance Doc Searcher, the similarity of the virtual documents is assessed using a vector space approach.

For the example in Fig. 4 and a height threshold of two, the searcher considers the fragments B1, B2, P2, P3, and P4. The runtime complexity of this searcher mainly depends, besides size and structure of the models, on the height up to which the RPST is traversed upwards for identifying groups of activities.

Wrapup Searcher. This searcher resembles the Similar Label Searcher, as it also aims at deriving potential 1:1 matches by analysing the string edit similarity for the labels of a pair of activities. In contrast to the Similar Label Searcher, however, not the whole Cartesian product of activities of two process models is considered. In fact, solely activities that have not been addressed in potential matches retrieved by other searchers are taken into account. Obviously, the Wrapup Searcher has to be run after all other searchers. In addition, the threshold for the similarity of two activity labels is typically set to a lower value than for the Similar Label Searcher.

3.3 Match Boosters

After potential 1:1 and 1:n matches have been identified, the multiset of scored matches is propagated to a set of boosters. We implemented the following four boosters as part of the ICoP framework.

Cardinality Booster. This booster reduces the *multiset* of potential matches to a *set* by aggregating the similarity scores for potential matches that associate the same (sets of) activities to each other. Two matches (A_1, A_2, s_1) and (A_3, A_4, s_2) with $A_1 = A_3$ and $A_2 = A_4$ are replaced by a match (A_1, A_2, s_a), such that $s_a = s_1 + (1 - s_1) * s_2$ is the first score increased by the second relative to its current value. Note that the operation is symmetric and can iteratively be applied if more than two scores need to be aggregated.

Subsumption Booster. The idea behind this matcher is that a 1:n match (A_1, A_2, s_1) might subsume another match (A_3, A_4, s_2), such that $A_1 = A_3$ and $A_4 \subset A_2$. As similarity scoring based on vector spaces tends to favour documents consisting of a small number of terms, the subsumed matches will have higher initial similarity scores on average. To countervail this effect, we boost a 1:n match, if it subsumes other matches. If the match (A_1, A_2, s_1) subsumes the match (A_3, A_4, s_2), its similarity score is increased relative to the current value, such that $s_1 := s_1 + w_s * (1 - s_1) * s_2$ with $w_s \in [0..1]$ as a weighting factor.

Tree Depth Ratio Booster. In contrast to the aforementioned boosters, this booster considers solely a single match and boosts the score of a match, if its activities show a certain structural property that is evaluated based on the RPST. Given a match (A_1, A_2, s_1), we determine the least common ancestors (LCA) of A_1 and A_2 in the RPST of the respective process model, denoted by $lca(A_1)$ and $lca(A_2)$. Further on, let $maxDepth_1$ and $maxDepth_2$ be the maximal depths of a fragment in the RPSTs of the two process models. Based thereon, we determine two ratios by relating the depth of the LCA to the maximal depth of the tree, i.e., $r_1 = \frac{lca(A_1)}{maxDepth_1}$ and $r_2 = \frac{lca(A_2)}{maxDepth_2}$, with $r_1, r_2 \in [0..1]$. Boosting the match takes places, if the average of the two ratios is above a threshold, which indicates that both LCAs are relatively low in the tree. That, in turn, can be interpreted as a hint for a good quality of the match. In this case, the similarity score of the match is increased according to the average of the two ratios, relative to the current similarity score, i.e., $s_1 := s_1 + (1 - s_1) * w_r * 0.5 * (r_1 + r_2)$ with $w_r \in [0..1]$ as a weighting factor.

Distance Ratio Booster. This booster also considers solely single matches. Here, the structural property that is evaluated relates to the graph distance as mentioned above. For each of the sets of activities A_1 and A_2 of a match (A_1, A_2, s_1), we determine the maximal distance between two activities of this set. That is, for each activity we compute the distance from or to all other activities and select the minimal distance. Then, the maximal value of all these minimal distances is chosen, denoted by $maxDistance(A_1)$ and $maxDistance(A_2)$, respectively. Furthermore, let $maxDistance_1$ and $maxDistance_2$ be the maximal distances that can be observed between two activities that are connected by a path in the two process models. Again, we define two ratios $r_1 = 1 - \frac{maxDistance(A_1)}{maxDistance_1}$ and $r_2 = 1 - \frac{maxDistance(A_2)}{maxDistance_2}$, with $r_1, r_2 \in [0..1]$. If the average of both ratios is above a threshold, the similarity score of the match is increased according, i.e., $s_1 := s_1 + (1 - s_1) * w_d * 0.5 * (r_1 + r_2)$. Here, $w_d \in [0..1]$ is the weighting factor.

3.4 Mapping Selectors

Once the match similarities have been adapted by the match boosters, a selector extracts a mapping from the set of scored potential matches. As mentioned above, any mapping has to satisfy the constraint of non-overlapping matches. While selectors follow different notions of quality for a mapping, they have in common that the derivation of the optimal mapping (w.r.t. to the chosen quality criterion) is a computationally hard problem. Therefore, our selectors follow a greedy or 1-look-ahead strategy. Experiments in a similar context revealed that results obtained by a greedy matching strategy are close to those obtained with an exhaustive strategy [10]. The ICoP framework consists of the following selectors.

Match Similarity Selector. This selector selects a non-overlapping mapping solely based on the similarity scores assigned to potential matches. The match with the highest score is selected (in case of multiple matches with an equal score one is chosen arbitrarily) and the set of potential matches is reduced by the matches that are overlapping with the selected match. The mapping is constructed iteratively until the highest score for a potential match is below a given threshold. Besides this greedy strategy, we also implemented a 1-look-ahead strategy, which optimizes the score for the succeeding iteration in case of multiple matches with equal scores.

Mapping Similarity Selector. This selector neglects the scores assigned to potential matches and relies solely on the score for a (partial) mapping as provided by an evaluator. The matches to build the mapping are iteratively selected such that in each step the match leading to the maximal score for the mapping is selected. In case of multiple matches meeting this requirement, one is chosen arbitrarily. The procedure terminates once the mapping score cannot be increased any further. Again, we implemented this greedy strategy and a 1-look-ahead variant that selects the match that leads to the maximal mapping score in the succeeding iteration.

Combined Selector. This selector uses both, match scores and mapping scores as provided by an evaluator. In a first step, the highest match score is determined. All potential matches having assigned this score are then selected for the mapping (in case this is not possible due to overlapping matches, selection is randomised). In a second step, the mapping is iteratively extended with the match that maximises a combined score that is built from its individual match score and the mapping score as computed by the evaluator. Here, both scores are combined in a weighted way (e.g., the mapping score might have a bigger impact than the match score). The procedure terminates if the combined score cannot be increased any further. Again, the second step of the selector has been implemented as a greedy and as a 1-look-ahead strategy.

3.5 Mapping Evaluators

A mapping selector can use a mapping evaluator to score mappings. Given a (partial) mapping, the evaluators return a single score for the quality of the mapping. Of course, different notions of quality can be considered. In the ICoP framework, we implemented the two following evaluators.

Graph Edit Distance Evaluator. This evaluator scores a given mapping based on the graph edit distance of the two original process models that is induced by the mapped activities. For a pair of graphs and a mapping between their nodes, the graph edit distance defines the minimal number of atomic graph operations (substitute node, insert/delete node, (un)grouping nodes, substitute edge, insert/delete edge) needed to transform one graph into another [11]. For these operations, only mapped nodes are considered to be substituted and potentially grouped. For example, to transform the upper process to the lower process in Fig. 1, considering the mapping that is represented in the figure, operations that have to be performed include: substituting 'Final Laser Unit Installation' by 'Final Laser Installation'; grouping 'Order Mechanical Parts' and 'Develop Mechanical Parts'; and inserting 'Assemble Laser Component'. Optionally, operations can be weighted instead of just counted (e.g.: a node insertion can count for 0.8 instead of 1). The graph edit distance can be leveraged to define a similarity score according to [12]. This score is computed as 1 minus the fraction of the given mapping and a hypothetical ideal mapping. The ideal mapping matches each node and each edge in one process with a node or an edge in the other process with a quality of 1.0. The similarity score defines the quality of the mapping according to the Graph Edit Distance Evaluator.

Path Relation Evaluator. The evaluator scores a given mapping based on whether the path relations are preserved for the activities of a pair of matches of the mapping. For a pair of matches $M_1 = (A_1, A_2, s_1)$ and $M_2 = (A_3, A_4, s_2)$, we derive the number of preserved path relations $pre(M_1, M_2) = |\{(a_1, a_2, a_3, a_4) \in (A_1 \times A_2 \times A_3 \times A_4) \mid path(a_1, a_3) \Leftrightarrow path(a_2, a_4)\}|$ with $path(x, y)$ being a predicate that denotes the existence of a path from activity x to activity y. Then, the evaluation score for the pair of matches M_1 and M_2 is defined as $s(M_1, M_2) = \frac{pre(M_1,M_2)}{|A_1 \times A_2 \times A_3 \times A_4|}$. Based thereon, the score for the mapping is computed by iterating over the Cartesian product of matches and computing the average of their scores.

4 Evaluation

We evaluate the matchers by comparing the matches that they discover to matches that business process analysts found in a collection of 20 pairs of process models. 3 pairs are taken from a merger in a large electronics manufacturing company. Each of these pairs represents two processes that have to be merged. 17 pairs are taken from municipalities. Each of these pairs represents a standard process [13] and an implementation of this standard process by a municipality. Each process model from the collection has, on average, 31.1 nodes, with a minimum of 9 nodes and a maximum of 81 nodes for a single process model. The average number of arcs pointing into or out of a single node is 1.2 and the average number of words in the label of a single node is 2.8.

For the 20 process model pairs, process analysts determined a total of 520 matched activity pairs. Of these 520 pairs, 221 were part of a complex match. However, the distribution of these complex matches in our model collection shows a high variation. For instance, for 3 out of the 20 model pairs (the model pairs from the merger), more than 90% of the activity pairs relate to complex matches. In turn, 6 model pairs contain solely elementary 1:1 matches.

We evaluate the performance of the matchers in terms of precision and recall. The precision is the fraction of found activity matches that that is correct (i.e., that is also found by process analysts). The recall is the fraction of correct activity matches that is found. The F-Score combines precision and recall in one value. We also compute the Overall score, an alternative metric that has specifically been developed for measuring the quality of schema matches [14]. Note that all metrics are based on activity pairs. Thus, complex matches are split up into activity pairs, e.g., $(\{a, b\}, \{x\})$ yields two activity pairs $(\{a\}, \{x\})$ and $(\{b\}, \{x\})$.

$\mathcal{C}_h$ = Activity pairs identified by human observer
$\mathcal{CC}_h \subseteq \mathcal{C}_h$ = Activity pairs that are part of a complex match
$\mathcal{CE}_h \subseteq \mathcal{C}_h$ = Activity pairs that are elementary matches ($\mathcal{CC}_h$ and $\mathcal{CE}_h$ partition $\mathcal{C}_h$)

$\mathcal{C}_m$, $\mathcal{CC}_m$, and $\mathcal{CE}_m$ are analogously defined as the sets of activity pairs, complex matches, and elementary matches identified by matcher m.

$$\begin{array}{llll} \textit{precision} & = |\mathcal{C}_m \cap \mathcal{C}_h| / |\mathcal{C}_m| & \textit{recall} & = |\mathcal{C}_m \cap \mathcal{C}_h| / |\mathcal{C}_h| \\ \textit{recall-elementary} & = |\mathcal{CE}_m \cap \mathcal{CE}_h| / |\mathcal{CE}_h| & \textit{recall-complex} & = |\mathcal{CC}_m \cap \mathcal{CC}_h| / |\mathcal{CC}_h| \end{array}$$

$$\begin{array}{ll} \textit{F-score} & = 2 \cdot (\textit{precision} \cdot \textit{recall}) / (\textit{precision} + \textit{recall}) \\ \textit{Overall} & = \textit{recall} \cdot (2 - 1/\textit{precision}) \end{array}$$

For our evaluation, we created five matchers within the ICoP framework.

Baseline Matcher This matcher represents the greedy graph matcher presented in [12] and consists of a Wrapup Searcher, a Graph Edit Distance Evaluator, and a Mapping Similarity Selector. This matcher identifies solely elementary matches and achieves high precision and recall in doing so. Therefore, we use it as a baseline benchmark for our framework, which focuses on improving results with respect to complex matches.

Matcher A This matcher consists of all searchers (cf., Section 3.2) and a Look Ahead Match Similarity Selector. Thus, it demonstrates the pure performance of our searchers.

Matcher B This matcher extends Matcher A by incorporating all four match boosters introduced in Section 3.3. Therefore, this matcher shows the impact of the boosters on the matching process.

Matcher C This matcher consists of all searchers, but in contrast to matcher A, it takes the evaluation of (partial) mappings into account when selecting a mapping. That is, it relies on the Path Relation Evaluator, while a Look Ahead Combined Selector is used to demonstrate its effect.

Matcher D This matcher consists of all searchers and evaluates partial mappings, but uses another mapping evaluator than matcher C, i.e., a Graph Edit Distance Evaluator.

Note that matchers A and B, as well as C and D, are very similar. The former rely solely on the scores assigned to matches in order to build up the mapping, whereas the latter use a combined approach that also considers the scores derived by evaluating a (partial) mapping.

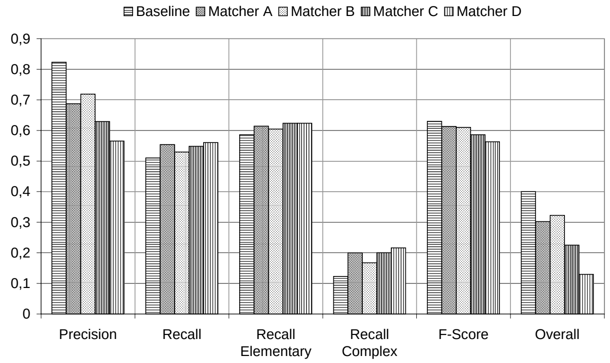


Fig. 5. Metrics derived for the whole model collection

Fig. 5 depicts the results of applying these five matchers to the set of 20 model pairs, by showing the precision, recall, F-Score, and Overall as defined above for one specific configuration of each matcher that maximises the F-score for the whole model collection. In addition to presenting the recall as a whole, it presents the recall for the complex 1:n matches and the elementary 1:1 matches separately. Note that, although the baseline matcher is not meant to detect complex matches, it does return some, due the fact that some activity pairs that it identifies as 1:1 matches are actually part of a complex match.

The results show that the architecture works and, while it has a more adaptable modular setup, produces the same results as were obtained by the more rigid matcher that we developed in prior work [12]. The results also show that the architecture can be used to produce matchers that detect complex 1:n matches and that their recall is better than that of our baseline matcher. Unfortunately, the complex matchers improve recall at the expense of precision, leading to an F-Score that is slightly lower than that of the baseline matcher. The comparison of the results for matchers A and B reveals that the application of match boosters increases the precision, at the cost of decreased recall. Similarly, the Path Relation Evaluator in matcher C leads to a better precision than the Graph Edit Distance Evaluator in matcher D, which, again, is traded for recall to a certain extent.

There is a large difference between the results with respect to the use-case from which they were derived. As indicated 3 model pairs originated from a comparison in a merger, while 17 originated from a comparison of standard processes to their implementations. The results for the merger model pairs are much worse than the results for the standard process comparison pairs. With an F-Score of around 0.3 the results for the merger model pairs are so bad that we conclude that the matchers cannot be used for this use-case. Qualitative analysis of the results shows that the reason for the bad performance of the matchers for this use-case mainly stems from the fact that the activity labels of matched tasks are very different. This leads to the conclusion that, in order for the current matchers to work, there must be some level of similarity between activity labels of similar activities; a pre-condition that is met in the standard process implementation use-case, but not in the merger use-case.

5 Related Work

Our work can be related to two main areas of research, namely process model similarity and database schema matchers that aim at finding complex matches.

Work on process model similarity can be traced back to formal notions of behavioural equivalence (see [15]) and integration of process models (see [16–20]). These works assume that correspondences between process models have been identified, but do not discuss how these correspondences can be found. Recent research in this area provides promising solutions to automatically match activities in pairs of process models, mainly as a vehicle to automatically calculate a similarity value for the models altogether. Structural and semantic information is used in [21], behavioural information in [22], and graph-edit distance techniques are used as well [10]. All these approaches only identify 1:1 matches between activities, just as many approaches from schema matching do [3, 7]. With our ICoP framework we address the need to automatically propose complex matches between process model activities, which has been identified in [1].

One of the few database schema matching approaches that consider complex matches is the iMAP system [4]. iMAP inspired our work, as it introduces the idea of searching the space of potential complex matches using a set of searchers implementing different search heuristics. Subsequently, the similarity of target entities and potential matches is analysed in order to select the final mapping. In contrast to our work, iMAP searchers exploit the value distribution of instance data and also take domain knowledge (e.g., domain constraints) into account. Another fundamental difference is the search result, as iMAP searchers derive (linguistic, numeric, structural) mapping expression. While these expressions are of crucial importance for relating database entities to each other, they might be derived automatically in case of process model correspondences. Given a match between one activity in one process model and multiple activities in a second model, the model structure can be leveraged to decide whether the former corresponds to the conjunction, disjunction, or another logical combination of the latter. Similarly, the approach presented by Xu and Embley [23] relies on the discovery of characteristics for the instance data, the application of domain ontologies that describe expected data values, and further external knowledge (i.e., WordNet relations). Other work that aims at finding complex matches uses correlation mining techniques. The DCM framework [24] proposes to mine web query interfaces in order to identify grouping attributes, i.e., attributes that tend to be co-occurring in web interfaces. This knowledge is exploited to mine negative correlations between groups of attributes, which, in turn, hint at potential complex matches. Note that there are other matchers, e.g., Cupid [25], that retrieve complex matches by just applying a static similarity threshold for the selection of matches. Given a similarity matrix for all model elements, various match combinations for a single element might show similarity values above the threshold, such that complex matches are created. However, such an approach does not hint at strategies that are used to identify complex matches, as it assumes this knowledge to be already encoded in the similarity matrix.

We summarize that the few existing approaches for finding complex matches extensively rely on instance data and external knowledge. The former is not always available for process models, while the latter raises the question of how to utilize external knowledge for a dedicated domain. Nonetheless, the use of instance data and external knowledge are promising directions for future work.

6 Conclusion

This paper presents the ICoP framework, which provides a flexible and adaptable architecture for the implementation of matchers, by splitting up matchers into searcher, booster, evaluator and selector components. Also, it enables the development of matchers that detect complex 1:n matches, where existing matchers focus on detecting elementary 1:1 matches. Experimental results show that the framework can reproduce results of existing matchers by composing them from separate searcher, booster, evaluator, and selector components. The results also highlight that we are able to identify a significant number of complex 1:n matches. While this demonstrates the potential of our framework for improving matching results, we also explicated that, compared to existing 1:1 matchers, the increase in recall is often traded for a decrease in precision. Finally, the experiments show that a minimal level of label similarity for similar activities is required for the matchers to produce acceptable results. This level is met in case matches are determined between standard processes and their implementations, but not for the use-case in which matches are determined between processes that must be merged.

We aim at addressing this phenomenon in future work. In order to counteract the decrease in precision when considering complex matches, we plan to integrate the usage of external knowledge into the ICoP framework. The usefulness of applying such knowledge in general, and WordNet in particular, has been demonstrated by the aforementioned approaches for matching data schemas [4, 23]. It is worth to mention that external knowledge for the domain of business processes is also available in terms of several reference models such as the MIT Process Handbook. Moreover, we aim at extending the framework towards n:m matches. That requires new heuristics to select groups of activities for analysis, as the combinatoric problem is increased even further for this kind of matches.

References

1. Dijkman, R.: A Classification of Differences between Similar Business Processes. In: Proceedings of IEEE EDOC. (2007) 37–50
2. Dijkman, R.: Diagnosing differences between business process models. In: Proceedings of BPM. (2008) 261–277
3. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal **10**(4) (2001) 334–350
4. Dhamankar, R., Lee, Y., Doana, A., Halevy, A., Domingos, P.: imap: Discovering complex semantic matches between database schemas. In: SIGMOD (2004) 383—394
5. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**(8) (1966) 707–710
6. Qu, Y., Hu, W., Cheng, G.: Constructing virtual documents for ontology matching. In Carr, L. et al., eds.: WWW, ACM (2006) 23–31
7. Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag (2007)
8. Porter, M.F.: An algorithm for suffix stripping. Program **14**(3) (1980) 130–137
9. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: BPM. LNCS 5240. (2008) 100–115
10. Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In BPM. LNCS 5701. (2009) 48–63
11. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters **18**(8) (1997) 689–694
12. Dijkman, R., Dumas, M., García-Bañuelos, L., Käärik, R.: Aligning business process models. In: Proceedings of IEEE EDOC. (2009) 45–53
13. Documentair structuurplan, http://www.model-dsp.nl/ (20 February 2009).
14. Do, H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In Chaudhri, A. et al., eds.: Web, Web-Services, & DB Syst. LNCS 2593 (2002) 221–237
15. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica **37**(4/5) (2001) 229–327
16. Preuner, G., Conrad, S., Schrefl, M.: View integration of behavior in object-oriented databases. Data & Knowledge Engineering **36**(2) (2001) 153–183
17. Basten, T., Aalst, W.: Inheritance of Behavior. Journal of Logic and Algebraic Programming **47**(2) (2001) 47–145
18. Grossmann, G., Ren, Y., Schrefl, M., Stumptner, M.: Behavior based integration of composite business processes. In BPM. LNCS 3649 (2005) 186–204
19. Pankratius, V., Stucky, W.: A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In Hartmann, S., Stumptner, M., eds.: APCCM. CRPIT 43, Austral. Comp. Soc. (2005)
20. Mendling, J., Simon, C.: Business Process Design by View Integration. In Eder, J., Dustdar, S., eds.: Proceedings of BPM Workshops. LNCS 4103 (2006) 55–64
21. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In APCCM. CRPIT 67, Austral. Comp. Soc. (2007) 71–80
22. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring similarity between business process models. In: CAiSE. LNCS 5074 (2008) 450–464
23. Xu, L., Embley, D.W.: Discovering direct and indirect matches for schema elements. In: DASFAA, IEEE Computer Society (2003) 39–46
24. He, B., Chang, K.: Automatic complex schema matching across web query interfaces: A correlation mining approach. ACM Trans. Database Syst. **31**(1) (2006) 346–395
25. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In Apers, P. et al., eds.: VLDB (2001) 49–58