Querying Process Models Repositories by Aggregated Graph Search

Sherif Sakr[1], Ahmed Awad[2], and Matthias Kunze[3]

[1] National ICT Australia (NICTA) and University of New South Wales, Australia
ssakr@cse.unsw.edu.au
[2] Faculty of Computers and Information, Cairo University, Egypt
a.gaafar@fci-cu.edu.eg
[3] Hasso-Plattner-Institute, University of Potsdam, Germany
matthias.kunze@hpi.uni-potsdam.de

Abstract. Business process modeling is essential in any process improvement project. Yet, it is a time consuming and an error-prone step. With a rapidly increasing number of process models developed by different process designers, it becomes crucial for business process designers to reuse knowledge existing in model repositories, e.g., to find solutions for a recurring situation. Process model querying provides powerful means to address this situation. However, current approaches fail if no single process model satisfies all constraints of a query.

In this paper, we present a novel approach for querying business process models repositories, where a query is decomposed into several subqueries. Each subquery is then used to obtain matching fragments from process models stored in the repository. New process models are constructed from these fragments, which may originate from different process models. By this, several processes are assembled from matching fragments and presented to the process designer as a ranked list. The main advantage of our approach is that the designer does not need to specify the subqueries, as they are derived automatically.

Keywords: Business process design, Reuse, Querying business processes, Process model composition.

1 Introduction

Business Process Management (BPM) aims at the automated support and coordination of business in an integrated manner by capturing, implementing, controlling, and evaluating all activities taking place in an environment that defines the enterprise [1]. Business process modeling is an essential first step in the business process engineering chain, as they enable a better understanding of the organization's operations by facilitating communication between business analysts and IT experts.

In general, designing a new business process model is a tedious and error-prone task that requires identifying the activities that need to be performed, ordering of their execution, handling exceptional cases that can occur, etc. Therefore, in any organization, business process models represent a main source of business knowledge, typically scattered among several IT systems, business documents, and the minds of involved people. This knowledge is usually reused each time a process model is created or updated, however, in an *ad-hoc* and generally uncontrolled fashion. Thus, it is of great value to have

M. La Rosa and P. Soffer (Eds.): BPM 2012 Workshops, LNBIP 132, pp. 573–585, 2013.
© Springer-Verlag Berlin Heidelberg 2013

systematic, flexible, and effective mechanisms to query and reuse the available knowledge of process model repositories to reduce time and effort, and improve the quality newly designed business process model.

Business process repositories have been developed along with techniques to access models, and associate them with metadata [2,3]. While search and retrieval of process models are largely based on keyword and full text search, certain approaches to effectively *query* process models according to their semantics have been proposed recently [4,5,6]. Based on the same notion of a query that is formulated to search a process repositories, these approaches fail, if no single process model satisfies all constraints in the query, i.e., they return no result.

In this paper, we present a novel approach for querying business process models, where the answer of a query graph can be assembled of fragments from different process models, when a single process model can not satisfy all the query constraints. Here, business process designers are enabled to compose new process models by reusing multiple fragments from different process models. The main advantage of our approach is that the designer does not need to specify components of the query, which shall be mapped to fragments in matched process models. Instead, the query is decomposed automatically into subqueries and each subquery is matched against the process model repository to retrieve matching fragments. The retrieved fragments are then combined to provide answers matching the query in form of a ranked list, from which the designer can select.

We implemented a proof of concept of our approach top of existing software, namely the open modeling platform *Oryx* [2] and the *BPMN-Q* query language [4,7,8]. BPMN-Q is a visual query language that closely resembles BPMN and thus, facilitates formulating queries even for non-technical users and novices in a business domain. The benefits of this approach is that it enables designing new process models by reusing fragments from several existing process models, by automatic query decomposition and matching on a fine-granular level of process fragments; hence, effectively reducing time and effort, while improving the quality and maturity of newly designed processes.

The remainder of this paper is organized as follows. We lay out the basics of business process models and BPMN-Q in Section 2 before Section 3 introduces our approach of querying graph-based repositories by aggregated graph search. Section 4 describes the mechanism to decompose the process model query and to aggregate matching process model fragments to form the query answers. An architectural overview of the implementation is provided in Section 5. Related work is discussed in Section 6 before we conclude the paper in Section 7.

2 Preliminaries

This section formally introduces process modeling and querying, which form the groundwork for our approach.

2.1 Business Process Modeling

Currently, there is a number of business process modeling languages, e.g., BPMN, EPC, YAWL, and UML Activity Diagram. Despite the variance in their concrete syntax and expressiveness, they all share the common concepts of tasks, events, gateways (or routing nodes), artifacts, and resources, as well as relations between them, such as control

flow. Without loss of generality, we can abstract from particular node types as their execution semantics are not vital to structural query matching, which is rather based on the concept of a process model graph.

Definition 1 (Process Model). *A* process model P *is a connected graph* (N, E)*, where* N *is a non-empty set of control flow nodes and* $E \subseteq N \times N$ *a nonempty set of directed* control flow edges *where* $\bullet n$ *(*$n\bullet$*) stands for the set of immediate predecessor (successor) nodes of* $n \in N$*.*

A process model has exactly one start event $n_{start} \in N$ *with no incoming and at least one outgoing control flow edge, i.e.,* $|\bullet n_{start}| = 0 \wedge |n_{start} \bullet| \geq 1$*, and exactly one end event* $n_{end} \in N$ *with at least one incoming and no outgoing control flow edge, i.e.,* $|\bullet n_{end}| \geq 1 \wedge |n_{end} \bullet| = 0$*. Each other control flow node* $n \in N \setminus \{n_{start}, n_{end}\}$ *is on a path from* n_{start} *to* n_{end}*.*

A connected sub-graph of a process model is a *process model fragment*. We refer to a specific type of process model fragments that have a *single entry* node and a *single exit* node [9] as *process model components*.

Definition 2 (Process Model Component). *A connected subgraph* (N', E') *of a process model* (N, E)*, where* $N' \in N, E' \in E$*, is a process model component PC iff it has exactly one incoming boundary node* $n_{in} \in N'$*, i.e.,* $\bullet n_{in} \subseteq N \setminus N'$ *and one outgoing boundary node* $n_{out} \in N'$*, i.e.,* $n_{out}\bullet \subseteq N \setminus N'$*.*

2.2 Business Process Model Querying

Based on the definition of process models and process model components, we introduce the concept of process model queries as a means to obtain process components from a collection of business processes models by structurally matching a query to each of them. BPMN-Q is a visual process model query language designed to help business process designers access repositories of business process models [4]. The language supports querying the control flow aspects of business process models. Moreover, it introduces new *abstraction* concepts that are useful for various querying scenarios.

Definition 3 (BPMN-Q Query). A BPMN-Q query is a tuple
$Q = (QC, QCF, QP, isAnonymous)$ *where:*
- QC *is a finite set of* control flow nodes *in a query,*
- $QCF \subseteq QC \times QC$ *is the* control flow relation *between control nodes in a query,*
- $QP \subseteq QC \times QC$ *is the* path *relation between control nodes in a query,*
- $isAnonymous : QC \rightarrow \{true, false\}$ *is a function that determines whether control flow nodes of a query are anonymous.*

Matching Queries to Process Models. A BPMN-Q query is matched to a candidate process model via a set of refinements to the query. With each refinement nodes (edges) in a query are replaced with the corresponding nodes (edges) of the matching process model. If one node can have more than one possible replacement within the process model, a new, refined copy of the query is created for each possible replacement. We call the replacement a resolution of an element of the query. Fig. 1 shows a sample BPMN-Q query along with a match to a process model, highlighted in grey. The query represents a path edge which connects two nodes, A and D, and returns the set of nodes that could exist in between these two nodes in the matching process model.

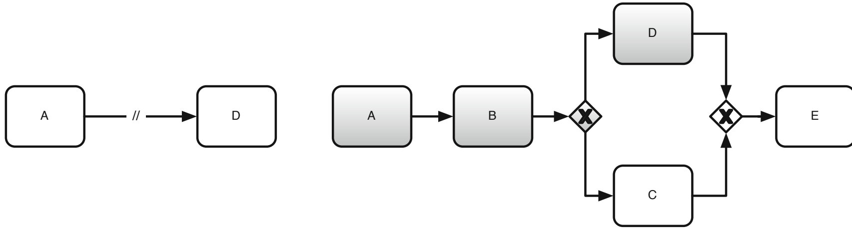


Fig. 1. An example BPMN-Q query with a match to a process model

Basically, the BPMN-Q query processor looks for exact matches of labels of activities in a query with those in the candidate process model. However, in practice, process modelers do not follow a strict naming scheme for activity labels. Thus, the query would find a small set of matching processes. To tackle this problem, we employed information retrieval techniques to automate the discovery of *semantically* similar activities [10]. The BPMN-Q query gets modified by substituting each of its activities with similar ones. With such a substitution step, new BPMN-Q query graphs are generated to constitute an expanded BPMN-Q query set.

Process components matching a query model will have a similarity score assigned ranging from 0 to 1. A similarity score of 1 indicates an exact match between the query and the process. Lower similarity scores indicate that a match was found between a semantically similar query and the process model. For more details about the BPMN-Q query language and its similarity matching mechanism, we refer the reader to [4,10,11].

3 Querying Process Models By Aggregated Search

The approach presented in this paper is based on the notion of *aggregated graph search* [12], where the answer of a process model query can be represented as an aggregation of process model fragments from multiple process models which are stored in the process model repository.

Definition 4 (Process Model Aggregated Search). *Given a process model query q and a process model repository $R = \{M_1, M_2, ..., M_n\}$, the problem of aggregated search of a process model query is to find a set of process models $S \subseteq R$ for which the joining of the matching process model fragments $F_{M_1}, F_{M_2}, ..., F_{M_k}$ from process models $M_1, M_2, ..., M_k \in S$ respectively, $F_{M_1} \bowtie F_{M_2} \bowtie ... \bowtie F_{M_k}$, leads to the answer of the process model query q.*

Fig. 2 shows a BPMN-Q query example which requires containment of an activity "*Check document*" that is immediately followed by an activity "*Verify customer record*" and two path edges from the latter activity to "*Assess risk*" and "*Open savings account*" respectively. Let us assume that the process model repository consists of the two process models which are shown in Fig. 3. Matching the BPMN-Q query to each process model separately fails to find any match. In particular, query evaluation against process model $P1$ fails because there is no path from activity "*Verify customer record*" to the "*Assess risk*" activity. Similarly, the query evaluation against process model $P2$ fails be-

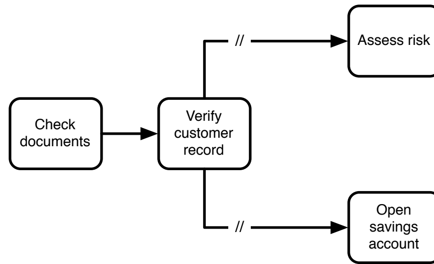


Fig. 2. An example BPMN-Q query

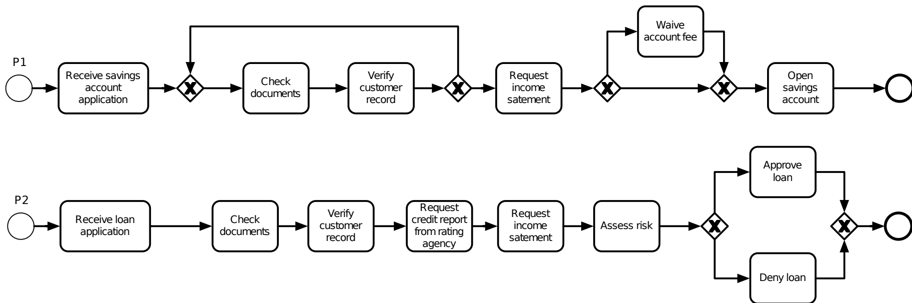


Fig. 3. Two process models to query

cause there is no path from activity "*Verify customer record*" to activity "*Open savings account*".

With our aggregation-based query approach, the query is not only matched collectively to each single process model, but the original BPMN-Q query is decomposed into subqueries which are matched individually against process models and the results of these sub-queries are aggregated to form the query answers.

4 Query Decomposition and Fragments Aggregation

This section introduces our approach towards above scenario, where a query is not met by a single process model, whereas subqueries can successfully be matched and returned fragments are aggregated into a newly designed process model.

4.1 BPMN-Q Query Decomposition

First, we focus on the decomposition of BPMN-Q query graphs. In particular, given an input BPMN-Q query q, we decompose the query graph into two sets of subqueries:

1. A set of static subqueries $StatQ$: where each static query represents a set of query nodes with static labels which are connected with direct flow edges.
2. A set of dynamic subqueries $DynaQ$: where each dynamic query contains at least one dynamic query element, e.g., anonymous node or path edge, in addition to a static join point with another subquery.

Algorithm 1. Decomposition Process of a BPMN-Q Query

```
Require: BPMN-Q Query: q,
Ensure: A Set of Static Sub-queries: StatQ
    A Set of Dynamic Sub-queries: DynaQ
    A Set of Join Points: JP
 1: SplitPoints:= IdentifyQuerySplitPoints(q);
 2: StatQ:= DetectConnectedStaticNodes(q);
 3: for all P ∈ SplitPoints do
 4:   if ∄(Q ∈ DynaQ | P ∈ Q) then
 5:     if P.Type == AnonymousActivity OR P.Type == GenericNode then
 6:       if P.HasIncomingEdges AND P.HasOutgoingEdges then
 7:         DQ1:= TraverseBackwardToFirstStaticPoint(P);
 8:         DQ2:= TraverseForwardToFirstStaticPoint(P);
 9:         DynaQ.Add(DQ1);
10:         DynaQ.Add(DQ2);
11:         JP.Add(P);
12:       else if P.HasIncomingEdges AND NOT P.HasOutgoingEdges then
13:         JoinPoint:= GetFirstStaticPointByBackwardTraversal(P);
14:         DQ:= TraverseBackwardToFirstStaticPoint(P);
15:         DynaQ.Add(DQ);
16:         JP.Add(JoinPoint);
17:       else if P.HasOutgoingEdges AND NOT P.HasIncomingEdges then
18:         JoinPoint:= GetFirstStaticPointByForwardTraversal(P);
19:         DQ:= TraverseForwardToFirstStaticPoint(P);
20:         DynaQ.Add(DQ);
21:         JP.Add(JoinPoint);
22:       end if
23:     else if P.Type == PathEdge then
24:       JoinPoint:= GetFirstStaticPointByBackwardTraversal(P);
25:       EndPoint:= GetFirstStaticPointByForwardTraversal(P);
26:       DQ:= SubGraph(JoinPoint, EndPoint);
27:       JP.Add(JoinPoint);
28:     end if
29:   end if
30: end for
31: for all Q ∈ DynaQ do
32:   if Q.HasNoStaticNodes then
33:     ExtendQuerySubgraphToIncludeStaticNode(Q);
34:     JP.Replace(Q.OriginalJoinPoint,Q.NewStaticNode);
35:   end if
36: end for
37: return StatQ, DynaQ, JP;
```

Algorithm 1 describes the steps of our BPMN-Q query decomposition mechanism. We start by identifying the set of *split points* of the input BPMN-Q query (Line 1). In particular, we specify the split points in a BPMN-Q query by the existence of any of the following BPMN-Q language constructs [4].

Anonymous Node. These nodes resemble activity nodes, but are distinguished by the (@) sign at the beginning of the node label. This query construct is used to allow usage of unknown activities in a query.

Path Edge. This query construct states that there must be a path from the source activity A to the destination activity B where the path edge is bound to all nodes and edges between the two nodes.

After determining the set of the split points, the set of decomposed static sub-queries ($StatQ$) is determined by identifying each set of nodes in the input query which have static labels and are connected by direct flow edges (Line 2). It should be noted that

static queries cannot contain any of the identified query split points. In addition, a static nodes of the input query cannot be included in more than one decomposed static query. The set of dynamic decomposed queries ($DynaQ$) is specified based on the identified query split points as follows:

- If the split node is of the type *anonymous node*, then the dynamic queries are specified according to the following conditions:
  - If the split node, SN, has *incoming edges and outgoing edges*, then *two* dynamic queries are constructed (Lines 6 to 11).

    The first dynamic query, $DQ1$, is constructed by traversing the query graph *backwardly* starting from the split point to the first node, $ST1$, with a static label or until no further nodes can be reached. The query subgraph that includes the split point SN and $ST1$ represents $DQ1$.

    Similarly, $DQ2$ is constructed by traversing the query graph *forwardly* to the first node, $ST2$, with a static label or until no further nodes can be reached. The query subgraph that includes the split point SN and $ST2$ represents $DQ2$. In this case, the split node (SN) represents the join point between $DQ1$ and $DQ2$.
  - If the split node, SN, has *only incoming edges* but no outgoing edges, then *one* dynamic query is constructed (Lines 12 to 16) by traversing the query graph *backwardly* starting from the split point to the first node, ST, with a static label. The query subgraph that includes the split point, SN, and ST represents DQ. The static node, ST, represents the join point between DQ and the (static or dynamic) query to which ST belongs.
  - If the split node, SN, has *only outgoing edges* but no incoming edges, then also *one* dynamic query is constructed (Lines 17 to 21) by traversing the query graph *forwardly* starting from the split point to the first node, ST, with a static label, to construct DQ. Also in this case, the static node, ST, represents the join point between DQ and the (static or dynamic) query to which ST belongs.
- If the split node is of the type *path edge*, then one dynamic query is specified (Lines 23 to 27) by traversing the query graph backwardly starting from the source node of the path edge to the first node, $ST1$, with a static label and then traversing the query graph forwardly starting from the destination node of the path edge to the first node, $ST2$, with a static label. The dynamic query, DQ, represents the subgraph between $ST1$ and $ST2$, where $ST1$ represents the join point between DQ and the query to which the node $ST1$ belongs.

The last step of our decomposition process is to verify that each dynamic sub-query has at least one node with a static label. It could occur that the dynamic query is generated with no static node if the traversal from the split point backwardly or forwardly stops by reaching a start or end node. If any sub-query fails to satisfy this condition, then it is expanded from its join point forwardly or backwardly until the first reachable static point and the join point is correspondingly adjusted (Lines 30 to 35). There will be no decomposition case if the input query does not have any node with a static label. It should be noted that each split point can be only included in one dynamic sub-query (Line 4).

Query execution starts by matching each query in the set of static sub-queries ($StatQ$) against the process model repository. In principle, the evaluation process of static sub-queries represent the traditional subgraph query matching problem, where exact or approximate means can be applied. The search process terminates if any of the decomposed static sub-queries has no match. Otherwise, query execution continues to evaluate each query in the set of dynamic sub-queries ($DynaQ$).

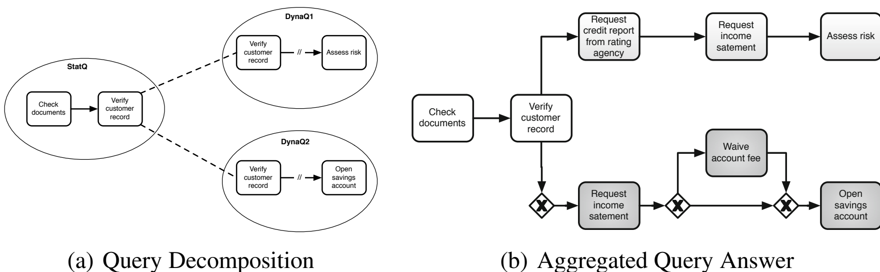
The results of both, static and dynamic, sub-queries, which may originate from different process models, are then joined to form the aggregated answer of the input query q. Fig. 4(a) illustrates the static and dynamic sub-queries from the decomposed BPMN-Q query of Fig. 2. Fig. 4(b) illustrates an aggregated query answer that combines process model fragments from the two process models which are presented in Fig. 3.

4.2 Aggregating the Process Model Fragments

The main task of a query processor is to evaluate the BPMN-Q queries of the decomposed static or dynamic sub-queries, which are discovered according to the decomposition process of Section 4.1, against the process model repository. For each BPMN-Q sub-query, a result set is returned that comprises *matched* process model components. These matched components could represent exact or similar matches for the query models, cf. Section 2.2. In case of similar matches, each matched process model component is then attached with its similarity score (SS), which is computed during the query evaluation process. In case of an exact match, the value of this similarity score is equal to 1 for each matched component of the result set [10].

From multiple matched components for each sub-query, which usually belong to different process models, follows that we can have several possible aggregation results that originate from distinct process models. Each potentially aggregated result needs to include exactly one component from the answer set of each sub-query. Clearly, it is inconvenient for process designers to go through this potentially very large list of aggregated models to select among them.

Therefore, the set of possible aggregated results are *ranked* according to various criteria, applying a ranking process that starts by initially ranking the matched process model components inside the answer set of each query based on their similarity scores. Then, it computes a ranking score for each possible aggregated model, by a number of



(a) Query Decomposition (b) Aggregated Query Answer

Fig. 4. BPMN-Q Query Decomposition and Aggregated Search for Business Process Models

elementary aggregation scores. In the following we describe two basic scores, further measures that also incorporate meta data are discussed in [13].

Combined Similarity Matching Score (CS). This score multiplies the respective similarity scores SS of each aggregated model component mc_i with regards to the matched sub-query

$$CS(ccp) = \prod_{i=1}^{n} SS(mc_i)$$

where n represents the number of the model components.

Homogeneity Score (HS). Model components that originate from the same process model shall be preferred over those that stem from different models to increase a result model's homogeneity and consistency. The pair homogeneity score, PHS, computes the homogeneity of each *unique pair* (mc_i, mc_j) of the model components by their origin:

$$PHS(mc_i, mc_j) = \begin{Bmatrix} 0 \text{ if the original models of the pair are different} \\ 1 \text{ if the original models of the pair are the same} \end{Bmatrix}$$

In general, the number of unique different pairs n is equal to $\frac{c(c-1)}{2}$ where c is the number of process model components. The homogeneity score of an aggregate model is then computed b¡

$$HS(ccp) = \frac{\sum_{i=1}^{n} PHS(udp_i)}{n}$$

where udp represent a unique pair of model components (mc_i, mc_j) and n represent the total number of unique different pairs.

The *final ranking score* of a candidate aggregated model is computed by the weighted sum of the elementary scores above,

$$FinalScore(ccp) = w_1 * CS(ccp) + w_2 * HS(ccp)$$

where w_i represents a weighting factor for a scoring element which can be configured and adjusted by the end-user, while $w_1 + w_2 = 1$. Initially, process designers can rely on a uniform regression parameter where all weighting factors have the same value, i.e., $w_i = 0.5$. With the continuous usage of the system, workload data can be gathered to generate significant training datasets that can be used as an input for a regression analysis process to deduce optimized weighting factors [14].

5 Framework Architecture

In this section, we describe the architecture of our implementation for the *aggregated graph search* framework for querying repositories of business process models, illustrated in Fig. 5, which consists of the following main components.

Process Model Repository. Instead of building our approach on top of a proprietary repository, it shall be connected to several, potentially disparate repositories, obtain and maintain process models stored remotely. Repositories do not only store models [15], but also a set of metadata, which can be used for aggregation and ranking.

Fig. 5. Framework Architecture

Query Processor & Process Model Indexes. The *query processor* evaluates queries received from the query interface [7]. It provides support for relaxation and refinement of user queries. In case the queries do not return sufficient results, the query processor is able to relax the query according to some similarity notions [10,16]. In order to further improve the searching, process models could be *indexed* upfront to expedite query evaluations [17].

Process Modeling and Querying Environment. provides the process designer with a user-friendly *modeling and querying interface* [2]. Users express their queries using the BPMN-Q language [4]. The *query decomposer* identifies the candidate decomposed sub-queries, cf. Section 4.1, and passes them to the *query processor*. The returned set of process model fragments for each query will then be combined and aggregated by the *fragment aggregator* and return a *ranked* list of aggregated process models as a result of the input query, cf. Section 4.2.

The *process modeling* environment of our framework is the *Oryx* editor, an extensible process modeling platform for research, designed to model and manage process models online [2]. The *Query interface* and *query processor* for BPMN-Q [4,7] components have been implemented as plugins to the Oryx editor and are able to run process model queries against the Oryx online process model repository. The *query decomposer* and *fragment aggregator* components are implemented as plugins to the Oryx editor that uses the BPMN-Q query processor to evaluate the results of each decomposed query and then returns the ranked aggregations to the end-user. Our architecture acknowledges the existence of a multitude of different process model repositories, which is the rationale behind the decoupling of a process modeling, querying, and aggregation components from a particular process model repository.

6 Related Work

Business process model search is a vivid topic among researchers and has attracted many solutions that can generally be distinguished in similarity search and process model querying [18]. An essential aspect of comparing processes is the alignment of process nodes [19]. That is, to discover relationships in one process model and map them to a second one, corresponding nodes must be discovered first. In most cases simple techniques such as string edit distance, n-grams, etc have been used. More complex techniques address one-to-many and many-to-many alignments to compensate for different levels of model granularity [19,20]. While we rely on simple, i.e., one-to-one, mappings to keep the paper concise, also complex mappings could be employed in our case.

With regards to to address process model similarity search, different structural techniques have been applied. For instance, in [16] the graph-edit-distance was used to assess how much two process graphs resemble each other; graph homomorphisms have been used to find models that embrace a given query model [21]. More sophisticated approaches address path resolution in graphs, i.e., if two nodes in a query model are connected by an edge, there must exist a path consisting of edges and nodes that connects correlating nodes in the stored models. Examples are BP-QL [5] which is restricted to BPEL and uses XML to formulate queries, IPM-QL that requires a custom XML representation for models and query, and BPMN-Q [4] where a visual query language that resembles the BPMN notation has been proposed for process model querying. The work presented in this paper, leverages BPMN-Q. Further querying approaches addressed behavior [22,23] or ontological information [6]. In principle, our approach is fully agnostic with respect to integrating and reusing any similarity matching technique for process models into the query processor component.

None of these works addressed decomposing a query into several fragments, querying stored process models with each of these fragments, and constructing of a new model from matches that originate from different models. In earlier work [13], we have presented an approach for reusing process model components based on the notion of a *partial process model* which consists of static and dynamic components. The static components represent the concrete aspects of the process model, while the dynamic components are BPMN-Q queries explicitly defined by a user. Upon search, each dynamic component is matched against models in the repository and returned fragments are to be embedded in the overall process, thus completing the static components.

Here, we develop our approach one step further, such that the specification of static and dynamic components are provided automatically without any user involvement. The idea of *aggregated graph search* has been introduced for the basic exact subgraph matching problem in [12], where the authors present a decomposition of labelled, directed graphs into a relational data schema and means to efficiently query this knowledge by means of SQL. However, the decomposition of a BPMN-Q query is more complex, as the graphs comprise advanced semantics, i.e., different node types must be distinguished during search along with the generic node type that is kind of a wild card; matching of nodes must address the anonymous activity. Further, BPMN-Q provides path-edges that may resolve to a path consisting of several edges in a model to be matched. Hence, more sophisticated means are required to decompose, store, and retrieve models, and aggregate matched fragments.

7 Conclusion

In this paper, we introduced a new approach for querying and reusing knowledge contained in business process model repositories. In this approach, the answer of a process model query can be represented as an assembly of different process model fragments from *multiple* process models, when a single process model can not satisfy all the query constraints. To achieve this, the query is automatically decomposed into several sub-queries and the results of each sub-query—fragments of matched process models in the repository—are aggregated to form a new process model that satisfies the query. A list of possible aggregations is ranked in order to provide the business process designer with the closest answers for his query.

This approach provides several benefits by reusing materialized business knowledge which is available in existing process model repositories. The reuse is not only on the level of a whole process model, but rather on a finer grained level, i.e., process model fragments. The approach *automatically* and *flexibly* collects components from different process models. Therefore, the approach can effectively reduce the time and effort of the business process modeling task. It can also effectively improve the quality and maturity of the newly developed business process models.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
2. Decker, G., Overdick, H., Weske, M.: Oryx – Sharing Conceptual Models on the Web. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 536–537. Springer, Heidelberg (2008)
3. Rosa, M.L., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: An advanced process model repository. Expert Syst. Appl. 38(6), 7029–7040 (2011)
4. Awad, A.: BPMN-Q: A Language to Query Business Processes. In: EMISA (2007)
5. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes with BP-QL. Inf. Syst. 33(6), 477–507 (2008)
6. Markovic, I.: Advanced querying and reasoning on business process models. In: BIS (2008)
7. Sakr, S., Awad, A.: A framework for querying graph-based business process models. In: WWW (2010)
8. Awad, A., Sakr, S.: Querying Graph-Based Repositories of Business Process Models. In: Yoshikawa, M., Meng, X., Yumoto, T., Ma, Q., Sun, L., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 6193, pp. 33–44. Springer, Heidelberg (2010)
9. 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)
10. Awad, A., Polyvyanyy, A., Weske, M.: Semantic Querying of Business Process Models. In: EDOC (2008)
11. Laue, R., Awad, A.: Visual suggestions for improvements in business process diagrams. J. Vis. Lang. Comput. 22(5), 385–399 (2011)
12. Le, T.-H., Elghazel, H., Hacid, M.-S.: A Relational-Based Approach for Aggregated Search in Graph Databases. In: Lee, S.-g., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012, Part I. LNCS, vol. 7238, pp. 33–47. Springer, Heidelberg (2012)

13. Awad, A., Sakr, S., Kunze, M., Weske, M.: Design by Selection: A Reuse-Based Approach for Business Process Modeling. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 332–345. Springer, Heidelberg (2011)
14. Hwang, C., Hong, D.H., Seok, K.: Support vector interval regression machine for crisp input and output data. Fuzzy Sets and Systems 157(8) (2006)
15. Bernstein, P., Dayal, U.: An overview of repository technology. In: VLDB (1994)
16. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
17. Sakr, S.: GraphREL: A Decomposition-Based and Selectivity-Aware Relational Framework for Processing Sub-graph Queries. In: Zhou, X., Yokota, H., Deng, K., Liu, Q. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 123–137. Springer, Heidelberg (2009)
18. Dijkman, R.M., Rosa, M.L., Reijers, H.A.: Managing large collections of business process models - current techniques and challenges. Computers in Industry 63(2), 91–97 (2012)
19. Weidlich, M., Dijkman, R., 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)
20. Dijkman, R., Dumas, M., Garcia-Banuelos, L., Kaarik, R.: Aligning Business Process Models. In: EDOC (2009)
21. Grigori, D., Corrales, J.C., Bouzeghoub, M.: Behavioral Matchmaking for Service Retrieval. In: ICWS (2006)
22. Jin, T., Wang, J., Wen, L.: Querying Business Process Models Based on Semantics. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part II. LNCS, vol. 6588, pp. 164–178. Springer, Heidelberg (2011)
23. Kunze, M., Weske, M.: Local Behavior Similarity. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Wrycza, S. (eds.) EMMSAD 2012 and BPMDS 2012. LNBIP, vol. 113, pp. 107–120. Springer, Heidelberg (2012)