A Framework for Querying Graph-Based Business Process Models

Sherif Sakr
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia
ssakr@cse.unsw.edu.au

Ahmed Awad
Hasso-Plattner-Institute
University of Potsdam
Potsdam, Germany
ahmed.awad@hpi.uni-potsdam.de

ABSTRACT

We present a framework for querying and reusing graph-based business process models. The framework is based on a new visual query language for business processes called BPMN-Q. The language addresses processes definitions and extends the standard BPMN visual notations for modeling business processes for its concrete syntax. BPMN-Q is used to query process models by matching a process model graph to a query graph. Moreover, the reusing framework is enhanced with a semantic query expander component. This component provides the users with the flexibility to get not only the perfectly matched process models to their queries but also the models with high similarity. The query engine of the framework is built on top of traditional RDBMS. A novel decomposition-based and selectivity-aware relational processing mechanism is employed to achieve an efficient and scalable performance for graph-based BPMN-Q queries.

Categories and Subject Descriptors

H.2 [**Database Management**]: Query languages; H.1.m [**Information Systems**]: Value of information; D.2.8 [**Software Engineering**]: Software libraries

General Terms

Management - Standardization - Verification

Keywords

Querying Business Process - BPMN- Process Models

1. INTRODUCTION

To understand, communicate upon, or reengineer working procedures, companies document their daily routines in the form of business process models. Business process modeling is a complex, time consuming and error prone task. Model design requires determining the activities that need to be performed, ordering of their execution, handling exceptional cases that might occur, etc. In many cases, variants of process models need to be created in response to special business situations. For instance, there could several insurance claim handling process. One process is designed for people with special working environment conditions, another for people over seventy years, etc. With the rapid growth in the number of process models developed by different process designers, providing business process designers with a framework for reusing previously designed business process models and for making the best use of them is of great practical value. Therefore, in order to simplify and improve the business process modeling task, process models need to be highly reusable, favoring process flexibility and minimizing designs made from scratch. Moreover, reusing can effectively increase the quality and the maturity of the newly developed process models. Reusing of process models implies the need for querying a process repository to find a suitable previous work that can be the base for a new design. Hence, the need for an intuitive, easy-to-use and expressive query language of process models is very important.

Recently, BPMN has been considered as the *defacto* standard for process modeling. The BPMN-Q query language is a visual language to query repositories of process models which extends BPMN notations with very few additional constructs to serve its querying purpose [1, 3]. BPMN-Q allows expressing structural queries and specifies proceedings of determining whether a given process model is structurally similar to a query. The expressive power of BPMN-Q allows the construction of more complex queries that are more than just a path lookup. In principle, a BPMN-Q query is considered to be a graph which is going to be matched with process graph(s).

Graph data structures have been widely used to model many complex structured and schemaless data such as: XML documents, social networks, chemical compounds and *business process models*. Relational database management systems (RDBMSs) have repeatedly been shown to be highly efficient and scalable in hosting types of data which have formerly not been anticipated to live inside relational databases such as complex objects, spatio-temporal data and XML data. In addition, RDBMSs have shown their ability to handle vast amounts of data very efficiently using their powerful indexing mechanisms. In this work we utilize the powerful features of the relational infrastructure to implement efficient mechanisms for processing graph-based business process queries.

The goal of this demonstration is to present a novel, efficient and scalable framework to support business process designers to achieve an effective modeling task by querying and reusing existing process models. In particular, we summarize the main strengths of our demonstrated system as follows:

1. The framework is based on a novel, intuitive and visual query language for business process models, BPMN-Q. It allows users to define their business process models and their queries using a very similar set of notations.
2. The framework is enhanced with a semantic query expansion component which employs an ontological dimension in the query matching process and tackles the problem of applying different terminologies when modeling processes. Ontology construction does not assume a priori semantic tagging or semantic description of process models.

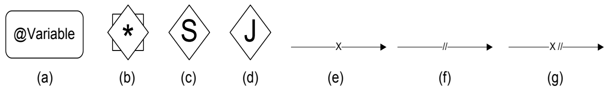


Figure 1: BPMN-Q Elements.

3. To achieve an efficient and scalable performance, the SQL-backend query processor makes use of the robust indexing infrastructure available by RDBMS and uses a novel decomposition-based and selectivity-aware translation mechanism of BPMN-Q graph-based queries into efficient SQL scripts.
4. The framework architecture is designed in a very flexible front-end/back-end fashion. On the front-end, it uses BPMN for modeling business processes, BPMN-Q for querying business processes and standard SQL relational processor as its back-end. However, the framework can be easily adapted to other modeling notations such as: EPCs, UML ADs.

2. BPMN-Q: A VISUAL QUERY LANGUAGE FOR BUSINESS PROCESSES

BPMN-Q [1] is a visual language that uses the BPMN notations as its concrete syntax. It is used to query business process models by matching a process model graph to a query graph. BPMN-Q extends the set of notations of BPMN with seven new elements. Some of these elements are flow objects, the others are for connectivity to serve its querying purpose. These elements are shown in Figure 1 and are described as follows:

(a) **Variable Node**: it resembles an activity but is distinguished by the @ sign in the beginning of the label. It is used to indicate unknown activities in a query.
(b) **Generic**: this indicates an unknown node in a process. It could evaluate to any node type.
(c) **Generic Split**: refers to any type of split gateways.
(d) **Generic Join**: refers to any type of join gateways.
(e) **Negative Sequence Flow**: states that two nodes A and B are not directly related by sequence flow.
(f) **Path**: states that there must be a path from A to B. A query usually returns all paths.
(g) **Negative Path**: states that there is not any path between two nodes A and B.

A BPMN-Q query is represented as a business process diagram (*graph*) that might contain additional query elements that will be substituted with BPMN elements during its processing. The result of such a graphical query is given by a sub-graph of the original process model. Figure 2 is a simplified loan handling process using BPMN notations. An example query and its match are shown in Figure 3. When matching a process graph (Figure 2) to the query in Figure 3(a), the result of the path edge is the sub-graph of the matching process in which the two nodes along with nodes in between are contained (Figure 3(b)).

An interesting usage scenario of BPMN-Q is to find similarities between process models on both structural and semantical basis [3]. For example, a query with a path from activity “Receive purchase request” to activity “Archive request” would be created by a business designer to lookup situations of handling purchase requests. For the basic query processor of BPMN-Q, it looks for process models having activity labels strictly matching those in the query. Thus, process models having activities on the form “Get purchase order”, “Process purchase request”, etc., will not be inspected by the query processor, though they are *semantically* relevant to the query. To overcome this limitation, the basic query processor was extended by a semantic expansion layer [3]. In that layer, information retrieval techniques are employed to analyze labels of activities in process models from a semantical point of view.

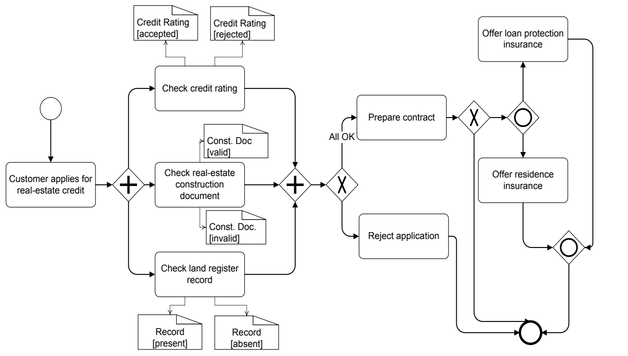
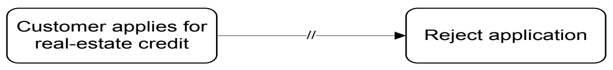
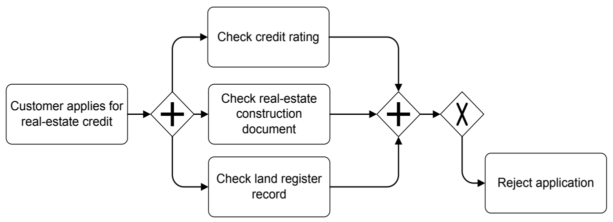


Figure 2: Sample Banking Process Model (in BPMN Notation)



(a) Sample BPMN-Q Query with Path edge


Customer applies for real-estate credit
Check credit rating
Check real-estate construction document
Check land register record
Reject application

(b) Match of the Query

Figure 3: A Sample BPMN-Q Path Query and its Match

The semantic expansion is light weight as it derives the semantic similarity without any prerequisite annotation of activities done by a human. Rather, a vector space model [10] with knowledge of WordNet[1] ontology is used on the words in the labels of activities to derive the similarity. With this expansion, models as discussed above are now relevant to the query. Of course, such an expansion adds to the complexity of the query processing. To control this complexity, the user is asked to determine a search *threshold* that controls the search depth and thus the time taken to process queries.

3. FRAMEWORK ARCHITECTURE

The framework of our implementation of querying and reusing business process models is designed in using very flexible *front-end/back-end* architecture. In the back-end, the repository of the business process graph models are stored and indexed in the RDBMS using a *fixed* relational scheme. In this repository we can store business process models defined using BPMN notations or models which are defined using any other notations (BPEL, EPC or UML) after applying the required transformations [12]. In the front-end, querying the business process repository is achieved through the use of a visual editor for the BPMN-Q query language. In the middle, the relational query processor evaluates the BPMN-Q queries and their semantic expansion (if required) over the relational scheme of the business process repository. This design allows us to easily port the implementation to any RDBMS or any visual BPMN-Q query editor. Figure 4 shows the framework architecture with the following main components:

[1]http://wordnet.princeton.edu/

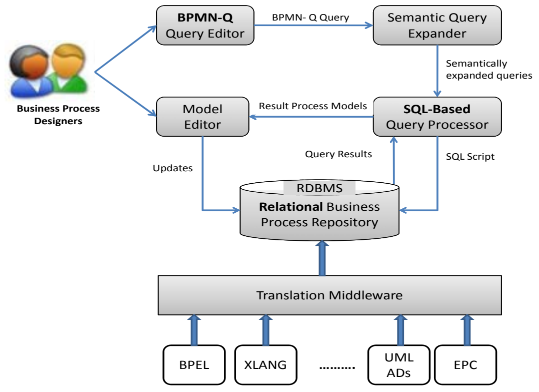


Figure 4: A framework architecture.

- **Translation Middleware**: translates business process definitions from specific languages syntax to the repository's internal representation. This Middleware facilitates the ability to unify the query interface against different process definition languages.
- **Repository**: is a central relational database that stores an abstract uniform representation of the enterprise process models. Conceptually, business process models are represented by graph data structures where events, activities and gateways are represented by graph nodes and the sequence flow between any two nodes is represented by an edge. On the physical level, we use a fixed-mapping storage scheme to map these graph data structure into a tabular relational representation. More details of the relational schema of the business process repository will be discussed in Section 4.
- **Query Editor**: is a visual *web-based* editor where the users can compose their queries using the BPMN-Q notations. The task of the query editor ends with the passing of the composed BPMN-Q query graph to the semantic query expander component before being shipped into the back-end SQL-based query processor.
- **Semantic Query Expander**: This component is responsible for employing an ontological dimension in the query matching process. It analyzes the labels of activities in process models from a semantical point of view and derives a list of semantically similar activities. In this way, the query processor is able to retrieve not only the exact matching process models but also the *relevant* process models.
- **Query Processor**: receives the BPMN-Q query graphs, translates them into SQL scripts which are then shipped into the backend RDBMs. The resulting process models are then received, *verified* and the relevant models are then passed to the Model Editor component for displaying purposes. More details of this component will be discussed in Section 4.
- **Model Editor**: displays the results returned by the query processor. Results can be changed by the user and then stored back in the repository, or can be reissued as new queries.

4. SQL-BASED EVALUATION FOR GRAPH-BASED QUERIES OF BPMN-Q

Retrieving related graphs matching a query graph from a large graph database is a key performance issue in any graph-based application. It is apparent that the performance of any of these applications is directly dependent on the efficiency of the graph indexing and query processing mechanisms. The query processing of BPMN-Q queries goes beyond the traditional sub-graph query processing in two ways. First, it does not treat all nodes of the graph repository or graph query in the same way. Each node has its own type and characteristics. Second, the connections (*edges*) between the nodes of the subgraph query are not always *simple* or *direct* connections that can be evaluated using the intuitive retrieval mechanisms. However, these query edges can represent more complex and recursive types of connections (*paths, negative paths* and *negative connections*) between the nodes of the business process graph models. Therefore, efficient *filtering* and *verification* techniques need to be employed to deal with these extra challenges posed by the semantics of the query constructs of BPMN-Q.

The business process repository uses a *fixed-mapping* storage scheme to store the graph structures of the process models. This relational storage scheme is described as follow:
BPModel(ModelID,ModelName,ModelDescription).
BPElements(ModelID,ElementID,ElementName,ElementType).
BPSequenceFlows(ModelID,SElementID,DElementID).
An obvious aspect in the SQL-based evaluation of BPMN-Q graph-based queries is the huge cost which may result from the large number of join operations which are required to be performed between the encoding relations. To overcome this problem, we exploit an observation from previous works which is that the size of the intermediate results dramatically affect the overall evaluation performance of SQL scripts [7, 6, 11]. Hence, we use an effective and efficient pruning strategy to filter out as many as possible of the false positives graphs that are guaranteed to not exist in the final results first before passing the candidate result set to an optional verification process. Therefore, we keep statistical information about the less frequently existing nodes and edges in the graph database in the form of simple Markov Tables. In our context, we are only interested in label and edge information with low frequency. Hence, we summarize these Markov tables by deleting high-frequency tuples up to a certain defined threshold, $freq$. This statistical information is then used to influence the decision of relational query optimizers by selectivity annotations of the translated query predicates to make the right decision regarding selecting the most efficient join order and the cheapest execution plan Moreover, we carefully exploit the fact that the number of distinct vertices and edges labels are usually far less than the number of vertices and edges respectively. Therefore, we try to achieve the maximum performance improvement for our relation execution plans by utilizing the powerful *partitioned B-trees* indexing mechanism of the relational databases to reduce the access costs of the secondary storage to the minimum [5].

BPMN-Q queries with large number of elements and sequence flow cannot be evaluated with one-step SQL evaluation or *view-based* mechanisms. They generate SQL queries that are too long and too complex and can not be executed by the back-end RDBMS. Therefore, we use a decomposition mechanism to divide this large and complex SQL translation query into a sequence of intermediate queries before evaluating the final results [9]. However, applying this decomposition mechanism blindly may lead to inefficient execution plans with very large, non-required and expensive intermediate results. Therefore, we reuse our statistical summaries to perform an effective selectivity-aware decomposition process.

Depending on the BPMN-Q query structure, the set of resulting process models may contain too few (*sometimes empty*) or too many result instances (*could be the whole repository for very simple generic queries*). Both cases are not helpful, useful or practical for the process designers. On the case of the too few results, the system tries to automatically find semantically similar process models using the semantic query expander component. The decision that the result set is small or not is based on a user defined threshold parameter. On the case of the too many results, it is unpractical to overwhelm the user with a very large set of process models. There-

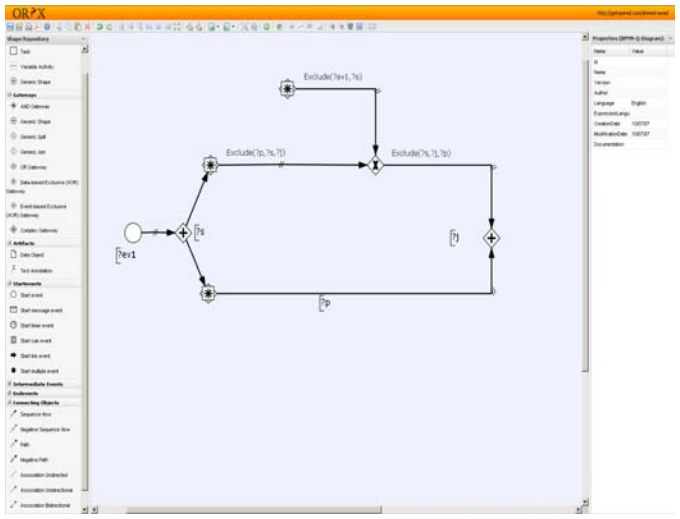


Figure 5: Screenshot of BPMN-Q query editor.

fore, in order to effectively deal with this situation and improve the query response time we apply two main techniques: 1) We use a user defined parameter to specify the maximum size of the query result set. The system uses this parameter to stop verifying the filtered process models when the specified maximum limit is reached. 2) The system divides the set of resulting process models into a number of chunks. Each chunk consists of N models where N is a user-defined parameter. To improve the systems response time, the system starts to return the results back to the users after determining the first positively verified N models.

5. DEMONSTRATION

The demo will show that the business process modeling task can be very interactive and efficient using intuitive, easy-to-use and expressive query language and efficient subgraph query processing techniques. Moreover, we will demonstrate that our framework can improve the quality and the maturity of the process modeling task by reusing the higher level business knowledge which are previously developed by business experts. The framework will be demonstrated using a very large real-world dataset collected from the online business process modeling repository, ORYX[2]. The dataset of business process model covers many application domains. The query engine of our demonstrated system is built on top of the IBM DB2 RDBMS as the database-backend. Sample BPMN-Q queries will be ready to run, but users can visually edit and design their own ad-hoc queries using *web-based* visual query editor as well (see Figure 5 for a snapshot[3]). The BPMN-Q queries will be evaluated by the relational query processor to *retrieve, filter* and *verify* the structurally matching process models and their semantically related models. The end users will be able to view and update the resulting process models through the Model Editor component. We will also demonstrate how our framework can provide effective solutions for many important use cases such as:

- **Compliance checking:** Compliance rules originate from different sources and are changing over time. The constantly changing nature of rules (e.g. due to changes in policies) calls for the checking of business processes each time a rule is added or changed. In case of manual auditing, a considerable amount of time is consumed in identifying the set of process models affected by each rule that may lead to the failure to meet the deadline for declaring compliance. Automated approaches for compliance checking can be achieved by expressing the checking rules as queries using a visual query language [2].
- **Detecting anomalies:** Even structurally sound process models can suffer from anomalies. For instance, an activity may have a precondition to read a data object in a specific state. If that data object does not reach the required state during execution of the process, the process will stall in deadlock at that point. Querying mechanisms can provide a remedy to this dilemma, as it allows the formulation of queries that find common model anomalies [4].
- **Discovery of frequent process patterns/anti-patterns:** Modularization has been always a principle of good software design. It helps to localize the effect of software updates and control redundancy. Querying techniques can be used to query the definition of process models and extract the frequent patterns. These discovered frequent patterns can be then moved to sub processes and replace their occurrences by calls to these sub processes. In addition the discovery of anti-patterns is very important to facilitate the ability to control the occurrence of unwanted process behavior [8].

Besides the framework demonstration, we will discuss about the design choices that we have made on defining the constructs of the query language, the indexing and query processing techniques. In addition, performance evaluation on the quality of similarity search of process models with respect to user studies and on search efficiency over a comprehensive dataset will be exhibited.

[2]http://oryx-project.org/backend/poem/repository
[3]Please visit http://bpmnq.sourceforge.net/ for live demo

Acknowledgments

The authors would like to thank Prof. Athman Bouguettaya for his valuable comments on this work.

6. REFERENCES

[1] A. Awad. BPMN-Q: A language to query business processes. In *EMISA*, 2007.
[2] A. Awad, G. Decker, and M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM*, 2008.
[3] A. Awad, A. Polyvyanyy, and M. Weske. Semantic querying of business process models. In *EDOC*, 2008.
[4] A. Awad and F. Puhlmann. Structural Detection of Deadlocks in Business Process Models. In *BIS*, 2008.
[5] G. Graefe. Sorting And Indexing With Partitioned B-Trees. In *CIDR*, 2003.
[6] T. Grust, M. Mayr, J. Rittinger, S. Sakr, and J. Teubner. A SQL:1999 Code Generator for the Pathfinder XQuery Compiler. In *SIGMOD*, 2007.
[7] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *VLDB*, 2004.
[8] R. Laue and A. Awad. Visualization of business process modeling anti patterns. In *VFfP*, 2009.
[9] S. Sakr. GraphREL: A decomposition-based and selectivity-aware relational framework for processing sub-graph queries. In *DASFAA*, 2009.
[10] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11), 1975.
[11] J. Teubner, T. Grust, S. Maneth, and S. Sakr. Dependable Cardinality Forecats for XQuery. In *VLDB*, 2008.
[12] Matthias Weidlich, Gero Decker, Alexander Großkopf, and Mathias Weske. BPEL to BPMN: The myth of a straight-forward mapping. In *OTM*, 2008.