BPMN-Q: A Language to Query Business Processes

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

Abstract: With the growing role business processes play in today's business life, they are being seen as an asset for the organization. With hundreds of process models developed by different process designers it would be helpful to look up the repository for models that could handle a similar situation before developing new ones. In this paper we introduce a new visual query language for business processes. The language addresses processes definitions. It extends BPMN for its abstract syntax as BPMN is a standard visual notation for modeling business processes. The overall architecture of the system in which the language can fit, in addition to the details of the query processing are also discussed.

1 Introduction

With the growing role business processes play in today's business life, business processes are being seen as an asset for the organization. With hundreds of process models developed by different process designers would be interested in looking up the repository for models that could handle a similar situation before inventing a new model. That is why *querying* a business process repository might be needed in such a situation. Querying of business processes can be divided into three stages, each of which has its own importance and its effect on way business is conducted:

1. Querying business process definition: this abstract model that shows the flow between activities, branching, joining, and data requirements. Querying at such level helps business analysts search for certain patterns within enterprise repository of business processes. This helps learn more about the way business is conducted, allows reusability of some business processes that might have been developed by others instead of duplication, and might help discover redundancy. Research under this category includes [VKL06, WLK06, BEKM06, LS06].

2. Querying running instances of business processes: querying here is almost a tool in the hand of administrator of a business process enactment engine to monitor the status of running processes, trace the progress of execution, make ad-hoc queries about a status of a certain process. Querying here is in flavor of Business Activity Monitoring BAM. This helps detect deadlocks, or unbalanced load on resources[MS04, BEMP07]

3. Querying execution history (logs) of completed business processes, which is also known as business process mining [vdAvDH+03] for a sample. While the purpose of business process mining is to reverse engineer the process definition from logs, purpose of querying here is wider. Depending on how much details the log gives, querying can provide us with statistics about duration of processes, bottlenecks, and the like.

In this paper we introduce a new query language that addresses the first category shown above. The rest of the paper is organized as follows, Section 2 discusses usages scenarios that are behind the design of the language in addition to requirements we extracted from these scenarios. Section 3 introduces the language informally discussing the abstract and concrete syntax of the language with an example. In Section 4 we formally describe the operation of the language. Section 5 discusses the different components of the language and how it was implemented. Details of how queries are processed are discussed in Section 6. Related work is discussed in Section 7. Section 8 concludes the paper with a view on how the work can be extended.

2 Usage Scenarios and Requirements

Our work on the development of this query language was motivated with business scenarios that were extracted from a thorough scanning of literature as shown in the following subsections.

2.1 Change Impact Analysis

Change in the design of process models is constant. This change could be due to new rules, obligations, or as a react to competitors improvement of service. Uncontrolled application of change would lead to degraded service or product delivered to customers rather than enhanced. This is because the change of some business activities or group of activities will lead to unforeseen affects on other business activities. Before applying change a business analyst or higher level manager should have a view on the related activities to the area of change. Impact or dependency (control flow, data flow) between activities are the major elements to be queried in this situation. Queries could take the form (for instance) like in [DC05].

2.2 Discovery of Frequent Process Patterns

Modularization has been always a principle of good software design. Modularity helps localize the effect of software updates and control redundancy. The same motivation can be applied to process models as an input to a transformation that delivers an executable busi-

ness process. If certain patterns of a group of business activities appear in the same way in several models, it seems feasible to move those patterns to sub processes and replace their occurrences by calls to these sub processes. Frequent patterns have been proven to exist in real business [TLR07], one of the difficulties that faced the authors were the lack of tools to query the definition of process models to extract those frequent patterns.

2.3 Checking Fulfillment of Quality Constraints

Enforcement of some quality checking operations in the business process might be necessary to fulfill requirements of international quality standards like TQM, or ISO[FES05] . For instance in manufacturing processes it is necessary that the product passes through a quality check process after manufacturing an before moving it to warehouse or making it available to customer. Checking conformance of process models to this constraint can be done by querying the structure of process models.

We believe the above scenarios motivate the need of a unified access to a shared repository of business processes where structure of processes is the target of queries. The queries in such cases also share the following properties:

- Ad-hoc: usually queries are started by a claim or a doubt by a business analyst who tries to prove his claim from the underlying pool of processes.
- Iterative: usually it is not just a query that captures a snapshot and it is over, rather it is a progressive process with the nature of discovery. It begins with a simple query then the results are modified to be input for another phase of querying. The cycle stop is dependent on what the user is looking for and how far she is satisfied with the result.

2.4 Requirements

From the above usage scenarios we can see that the query language should target both business users 2.1,2.3 and technical users 2.2. We summarize the requirements for the query language in the following points:

1. The language should be of visual interface. The visual interface increases the usability chance for the language specially by non-technical users.
2. The language must support the navigation of process structures to answer queries.
3. Query definition goes in the same way a process definition goes. (Try to introduce the least number of new notations). This makes the learning curve for the language lower.
4. The language should support the notion of paths between nodes in the process graph.

5. The result of the query can be either the whole process model containing a match to the query or only the matching part. This should be based on a user predefined preference.

6. The ability to modify the results of queries to create new queries to support the iterative nature for the querying scenarios.

3 BPMN-Q

According to the requirements mentioned in section 2.4, the language we introduce is a visual language that is based on notations from BPMN [bpm06] (Requirements 1,2). The language currently addresses a subset of modeling notations available in BPMN (Activities, Events, Simple gateways). Figure 1 shows a metamodel (abstract syntax) for the language which is an extension for the BPMN Metamodel,as an extension the *Activity* metaclass describes both concrete and variable activities (will be further explained in Section 3.1), also *GateWay* metaclass is further distinguished as either a split or join. The *Connectivity* metaclass was extended with new *Path* metaclass (details in Section 4.1) to satisfy requirement 4. while Figure 2(a) shows the graphical notation for elements already in BPMN, also the notation to support the new concepts are shown in 2(b) (Requirement 3). The symbol with nested square, diamond and a star we call it a generic shape (visualization of *FlowObject* metaclass), we introduce this shape for the sake of increasing the expressive power of the language; whenever the user is not sure about the type of thing he needs in a query, the generic shape is placed. At runtime the query processor will generate and test all possibilities as will be shown in Section 6. The same idea for the diamond with S (visualization of *Split* metaclass), and that with J (visualization of *Join* metaclass) inside.

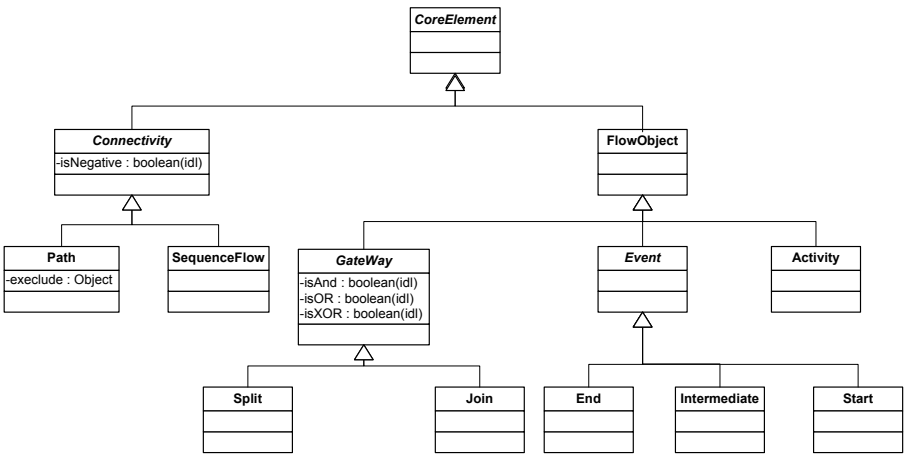

Figure 1: Language Metamodel

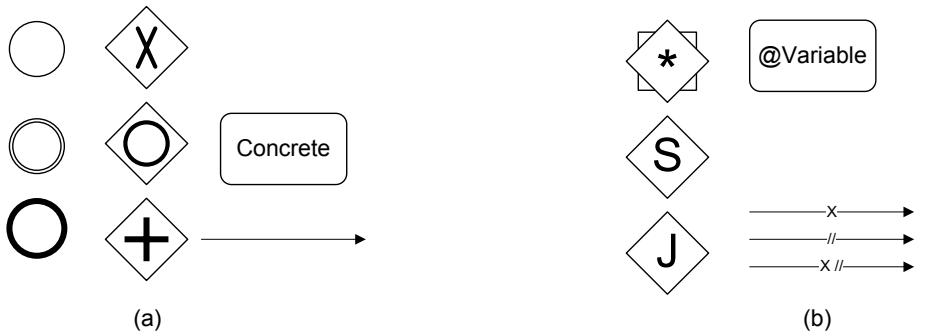

Figure 2: Language Elements

3.1 Example

We start with an informal introduction via an example to show how this is intended to work. Figure 3 shows a simple process model against which we will apply example queries.

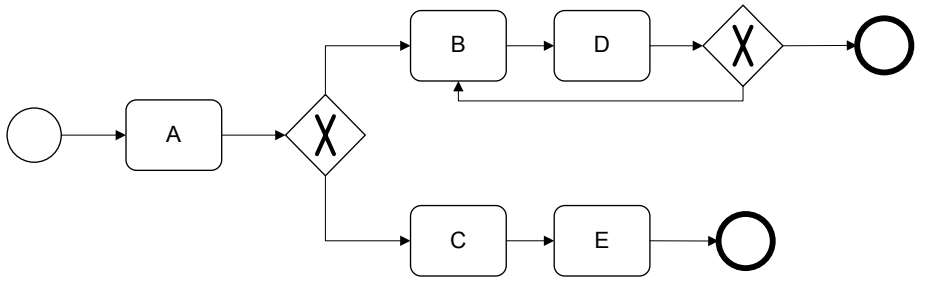

Figure 3: Sample business process model

Figure 4 contains five queries that represent simple edge expression queries (a),(b) and path expression queries (c),(d) and complex expression queries (e).

It is clear that the query representation is much like the way process models are defined (Requirement 3). There are two extensions shown in these queries:

1. Queries (a),(b) have an activity whose name is @X. This notation we call the variable activity i.e. an activity that might be bound to one or more activities in the queried process model. We prefix the activity name with the symbol "@" to inform the query processor that this is a variable activity node. In a single query expression no two variable nodes can have the same name.

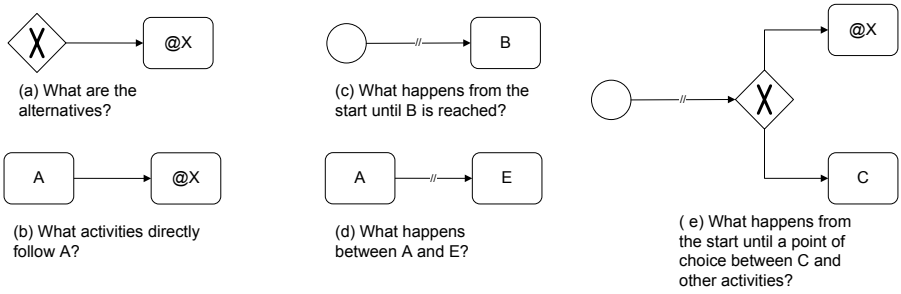

Figure 4: Sample queries

2. Queries (c), (d) the sequence flow arrow is labeled with the Symbol //. This is the symbol to represent path expressions between nodes.

To answer any query the query expression is matched to the process model. The query is answered via a set of resolution phases. If any of these phases fail, the query processing is terminated with no match. Details of processing queries are in Section 6. A single query graph maybe matched with more than one sub graph from the same process graph.

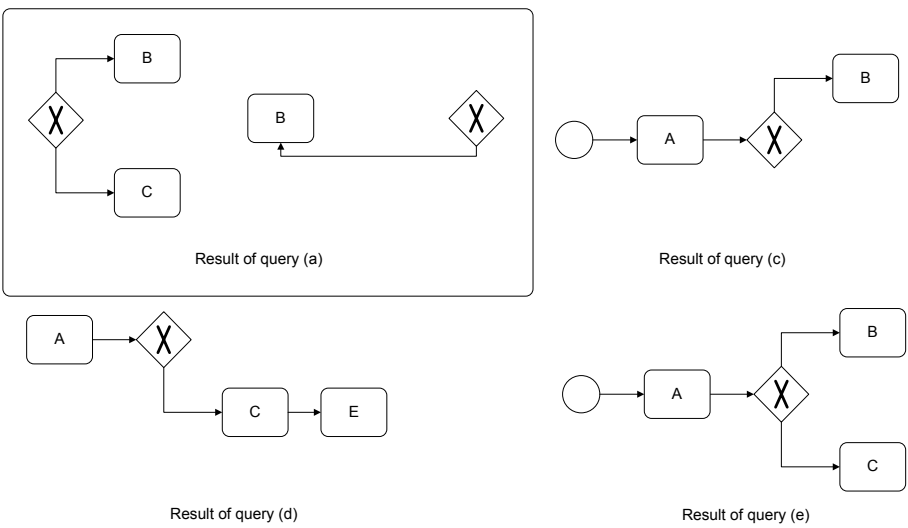

Figure 5: Queries results

Figure 5 shows the answer for the queries from Figure 4 against the process model in Figure 3. We notice that query (b) had no result because the variable node could not be bound with a concrete activity node that is a direct successor to activity A, also query (a) had two different results due to the fact that the process model had two different XOR

splits that have successor activities.

4 Formalization

In this section we give the formal background of both process graphs and query graphs. A process model based on BPMN can be viewed as a directed typed attributed labeled graph. Here each node in the graph has a type (task, gateway, event etc), each of them might be associated with other attributes like (name, further type details, ID) more details are in Section 5.

Definition 4.1 *A process graph is a tuple PG= (N, E, T, L) where*

- *N = finite set of nodes.*
- $E \subseteq N \times N$.
- *T:* $N \rightarrow \{$*ACTIVITY, EVENT, GATEWAY*$\}$
- *L:* $N \rightarrow l$ *is a labeling partial function .*

As we have seen in Section 3 a query is also a graph. It is a directed typed attributed labeled graph, where type ACTIVITY is further described as either a concrete activity or a variable activity. Edges in query graph are also typed as either sequence flow or paths that connect two nodes.

Definition 4.2 *A query graph is a tuple G= (N, S,NS, P, NP, T,L) where*

- *N = finite set of nodes.* $N = CA \cup VA \cup EV \cup GW$ *where*
  - *CA = set of concrete activities.*
  - *VA = set of variable activities.*
  - *EV = set of events.*
  - *GW = set of gateways.*
  - $CA \cap VA \cap EV \cap GW = \emptyset$
- $S \subseteq N \times N$ *= sequence flow edges between nodes.*
- $NS \subseteq N \times N$ *= negative sequence flow edges between nodes.*
- $P \subseteq N \times N$ *= path edges between nodes.*
- $NP \subseteq N \times N$ *= negative path edges between nodes.*
- *T:* $N \rightarrow \{$*CONCRETE ACTIVITY,VARIABLE ACTIVITY, EVENT, GATEWAY*$\}$
- *L:* $N \rightarrow l$ *is a labeling partial function .*

4.1 Paths

In Definition 4.2, path expression means actually all possible paths that can be found between the source and target nodes even if these paths contain cycles (due to the graph oriented nature of BPMN). Path expression is the most expensive in its computation, so when we describe how query processing takes place in Section 6, we will see how it is postponed by the query processor. The evaluation of a path expression contributes to the answer process graph according to Definition 4.3.

Definition 4.3 *A function allpaths:* $N \times N \times PG \rightarrow \{PG \cup \emptyset\}$ *= PG'(N',E',T,L) where:*

- *PG'* $\subseteq$ *PG.*
- $x \in$ *N' iff:*
  - *x = source.*
  - *x =target.*
  - *x lies on a path from source to target in pg* $\in$ *PG.*
- $\forall$ *x,y* $\in$ *N' e(x,y)* $\in$ *E* $\rightarrow$ *e(x,y)* $\in$ *E'*

It is possible also according to the path metaclass in Figure 1 to restrict the path expression to exclude certain activity node from the path.

4.2 Negative Edges and Negative Paths

According to Definition 4.2, it is possible to express in the query what we call negative edges NS, and negative paths NP. These represent further Boolean conditions that can be enforced on the result of the query. For instances if two nodes A and B are connected with a negative edge in the query graph this means that in any match to the query the nodes bound to A and B must not have a sequence flow edge between them. The same applies to negative paths. Negative edges and negative paths evaluation works according to Definitions 4.4,4.5

Definition 4.4 *A function checkNegativeEdge:*$N \times N \times PG \rightarrow \{true,false\}$ *= true iff e(n,m)* $\notin$ *E.*

Definition 4.5 *A function checkNegativePath:* $N \times N \times PG \rightarrow \{true,false\}$ *= true iff allpaths(n,m,p) =* $\emptyset$.

5 Architecture and Implementation

Figure 6 shows architecture for the query language. We briefly describe each component.

Query Editor: is a visual editor where the user can compose query in a way that conforms to Definition 4.2 and the queries look like those in fig. 4. The task of the query editor ends with passing the composed query graph to the processor component. The query editor is implemented by an extension to Microsoft Visio (see Figure 7 for a snapshot).

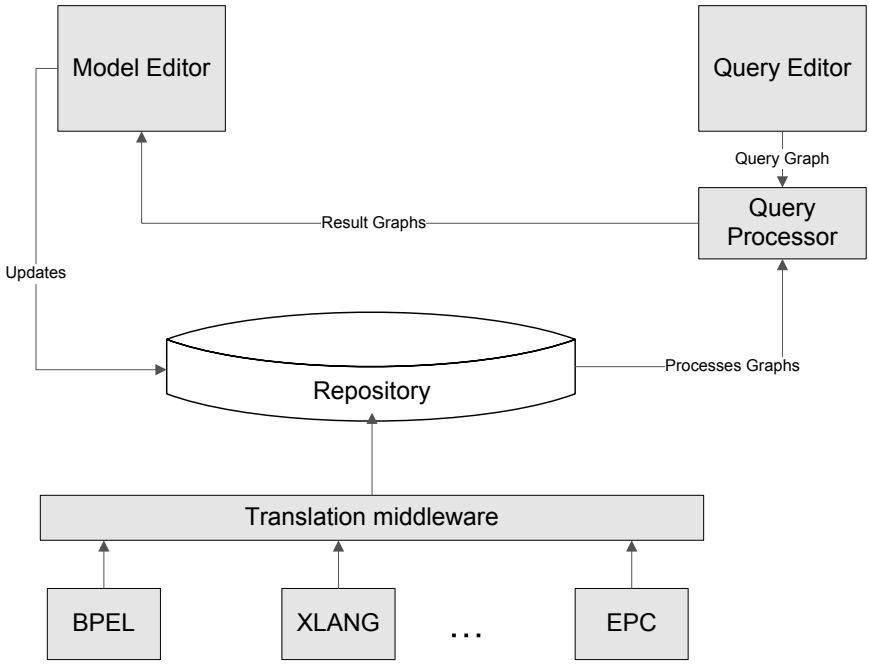

Figure 6: Suggested Architecture

Query Processor: receives the query graphs and works on answering it according to steps detailed in Section 6.
Repository: is a central database that stores an abstract uniform representation of the enterprise process models. This abstraction conforms to Definition 4.1. The database schema is simple with the following tables:

- Model(ID,Name,Description).
- Activity(ID,Name,Model).
- Event(ID,Name,Type,Model).
- GateWay(ID,Name,Type,Model).
- SequenceFlow(ID,Source,Destination,Model).
- Paths(Source,Target,Path,Model): This table is used to encode paths with all lengths that are extracted from process models. Extraction of paths comes in a post step to the storage of process models in other tables.

Model Editor: displays the results or messages 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 (Requirement 6). This is also implemented by extending Microsoft Visio.
Translation Middleware: translates business process definitions from specific languages syntax to the repository internal representation with metadata associated with the process graph relating it back to its source. This Middleware makes it possible to unify the query interface against different process definition languages(not currently implemented in the prototype).

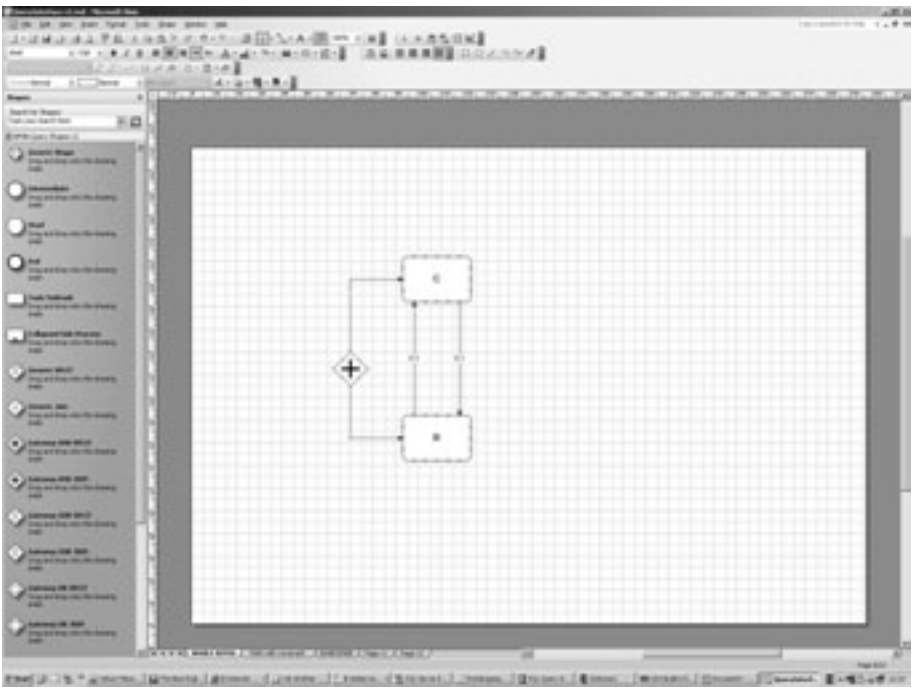

Figure 7: Query Editor

6 Query Processing

With the start of execution the query processor receives the corresponding query graph. To answer the query we need to search process definitions stored in the repository for those that satisfy the query graph. As we can see the repository as a graph database, one of the best ways to reduce the search space is to filter the database for only graphs which seem promising to satisfy the query [SWG02]. Within the set of graphs resulting from the filtering, we try to find an answer for the query. We have mentioned in Section 4 that all nodes are attributed by IDs. Actually we can distinguish two nodes of the same type by

ID. The process of assigning an ID to a node in the query graph with respect to a process graph is called *binding*. The query processor has to examine all possible bindings for each node. The query processor follows an approach in finding bindings that as much as possible binds a node in a way that will reduce the possible bindings for other nodes that are connected with it via *sequence flow* edges. We call this the *informed binding*. With following a set of binding steps query graph is transformed into a process graph that is the answer to the query.The query processor does the following steps for bindings 1) **Replace generic shapes**. This is a pre processing step whenever the query graph contains any of the generic shapes in Figure 2, the processor generates new versions of the original query graph in each the generic shape is replaced by shapes that can appear in process graphs. This operation simulates the principle of late binding in Object Oriented Programming OOP. 2) **Bind concrete activity nodes**. 3) **Bind event nodes**. 4) **Bind gateway nodes**. 5) **Bind variable activity nodes**. 6) **Check negative edges**. 7) **Check negative paths**. 8) **Substitute paths**.
After each binding step a new version of the query graph is created where the node is replaced by its bound node from the process graph, all edges and paths are also updated to refer to the bound node. Once a query graph version passes the first five steps, all its nodes are concrete i.e. are assigned IDs from the checked process graph. The query processor then goes to check negative edges and negative paths, steps 6, 7 according to Definitions 4.4,4.5 respectively. The last step the query processor does is to resolve path edges between nodes according to Definition 4.3 which means the maximal sub graph of the process graph in which nodes are either the two end nodes stated in the path , or a node that lies on a path from the start node to the end node. This step was postponed because (a) It is the most time consuming step, (b) We have to know exactly which nodes we look for paths between. If the query processor does not find an answer to any of the steps mentioned above, it terminates the evaluation of the query graph version. On the other hand, when a query graph version passes all the steps successfully it is now considered as an answer to the initial query and is forwarded to the Model Editor component to be displayed.

6.1 Performance

Table 1 shows the running time in milliseconds of queries varying in complexity, these queries are shown in Figure 8 . The queries were run against a repository containing 143 nodes, and 170 edges distributed among 11 process models. The machine used to run queries is a PC running Windows XP with 1 GB of RAM.

Query	Runtime	Query	Runtime
(a)	109	(e)	1172
(b)	219	(f)	5663
(c)	1313	(g)	3828
(d)	1141		

Table 1: Queries running time in milliseconds

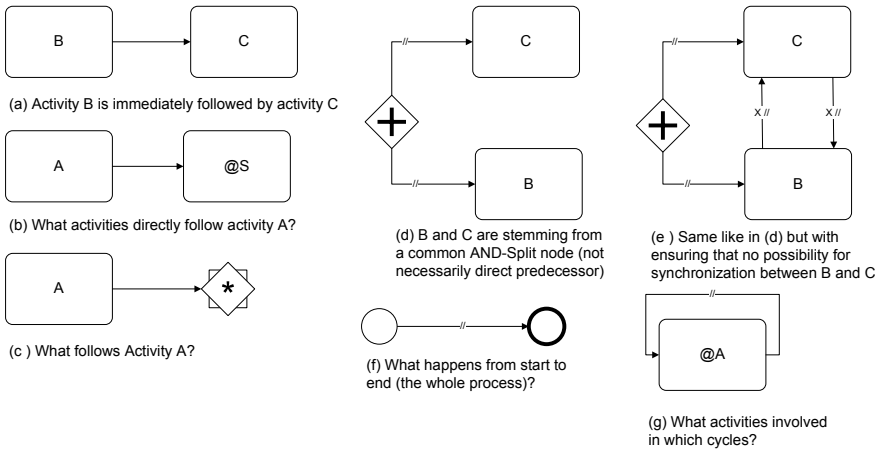

Figure 8: Queries of varying complexity

It can be simply deduced that the number of generic symbols (generic shapes, splits, and/or join), paths, variable nodes are of direct effect on the running time of a query, and this is due to that each type of them increases the number of possibilities to find a binding. Each of the alternative query graphs has to be tested for a match. In order to control the growth of generated query graphs we followed the concept of informed binding we talked about in this section. Another major effect on performance is the substitution of paths. The process of finding all possible paths between two nodes in a process graph at runtime goes in O(N^3), of course if more than one path expression exists in a query graph the running time of the query is unacceptable. It is even worse with the iterative nature of queries, with each run all paths have to be computed again. To solve this problem we added a computation step that is executed at loading time of a new process model to repository, and each time a process model is modified. This step is to compute paths with all lengths between nodes (if there is a path). So the cost of computing a path is now constant at query execution time rather than cubic.

7 Related Work

In [VKL06] a repository for process definitions based on BPEL [ACD$^+$03] was built. BPEL files are annotated with organization specific metadata, the repository was queried through a set of APIs that address at first place queries issued by programmers to select a specific BPEL file for execution based on metadata search criteria. Our work is distinguished in that we address humans at analyst level, query is based on the structural properties of process models. Similar work in [WLK06] queries the content of business process based on a framework for describing this content. The framework consists of four concentric levels: high level flow, business flow, activity, and task. Each level is associated

with metadata that are used for querying. The query goes from the broader scope narrowing the result set with each step from a level to the next level based on values specified for associated metadata at each level. Work in [BEKM06], [LS06] are considered close to our work from the point that they query process definition from structural point of view. The Business Process Query Language BPQL in [BEKM06] works on an abstract representation of BPEL files. We can distinguish our work with the handling of cycles in the process graph, a point that is not available in BPEL. Another point is the ability to query with generic expressions like generic shapes, splits, and/or joins. Querying process variants in [LS06] utilizes graph reduction techniques to find a match to the query graph in the process graph, comparing it to our work we have more concepts like the variable activities, path finding, and generic shapes which increase the expressive power of the language in addition to handling process graphs with cycles while [LS06] works on acyclic graphs only.

8 Conclusion and FutureWork

In this paper we introduce a new visual query language for searching repositories of business processes. The language was an extension of BPMN in its abstract syntax. Also we discussed an architecture where the language fits, the details of query processing were discussed. We have shown throughout Section 3 and Section 5 how requirements 1,2,3,4, and 6 were satisfied. Regarding requirement 5 we chose in our prototype implementation to show the matching model with highlighting the answer within it. As Future work, covering data flow, resources, and messaging between interacting processes are open areas to be included in the query language.

References

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services version 1.1. Technical report, OASIS, 2003.

[BEKM06] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying business processes. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 343–354. VLDB Endowment, 2006.

[BEMP07] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Query-based monitoring of BPEL business processes. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1122–1124, New York, NY, USA, 2007. ACM Press.

[bpm06] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, OMG, 2006.

[DC05] Weizhen Dai and H. Dominic Covvey. Query-Based Approach to Workflow Process Dependency Analysis. Technical Report 01, School of Computer Science and the Waterloo Institute for Health Informatics Research, Waterloo,Ontario,Canada, 2005.

[FES05] Alexander Förster, Gregor Engels, and Tim Schattkowsky. Activity Diagram Patterns for Modeling Quality Constraints in Business Processes. In Lionel C. Briand and Clay Williams, editors, *MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2005.

[LS06] Ruopeng Lu and Shazia Wasim Sadiq. Managing Process Variants as an Information Resource. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 426–431. Springer, 2006.

[MS04] Mariusz Momotko and Kazimierz Subieta. Business Process Query Language a Way to Make Workflow Processes More Flexible. In *ADBIS'2004: Proceedings of the 8th East-European Conference on Advances in Databases and Information Systems*, pages 306 –321. Springer Berlin / Heidelberg, 2004.

[SWG02] Dennis Shasha, Jason Tsong-Li Wang, and Rosalba Giugno. Algorithmics and Applications of Tree and Graph Searching. In *Symposium on Principles of Database Systems*, pages 39–52, 2002.

[TLR07] Lucineia Heloisa Thom, Cirano Lochpe, and Manfred Reichert. Workflow Patterns for Buisiness Process Modeling. In Barbara Pernici and John Atle Gulla, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 349–357. Springer, 2007.

[vdAvDH+03] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.

[VKL06] Jussi Vanhatalo, Jana Koehler, and Frank Leymann. Repository for Business Processes and Arbitrary Associated Metadata. In *Demo Session of the 4th International Conference on Business Process Management*, pages 25–31, Vienna, Austria, 2006.

[WLK06] Avi Wasser, Maya Lincoln, and Reuven Karni. ProcessGene Query- a Tool for Querying the Content Layer of Business Process Models. In *Demo Session of the 4th International Conference on Business Process Management*, pages 1–8, Vienna, Austria, 2006.