APQL: A Process-model Query Language

Arthur H.M. ter Hofstede[1,2], Chun Ouyang[1], Marcello La Rosa[1,3], Liang Song[4], Jianmin Wang[4], and Artem Polyvyanyy[1]

[1] Queensland University of Technology, Brisbane, Australia
{a.terhofstede,c.ouyang,m.larosa,artem.polyvyanyy}@qut.edu.au
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
[3] NICTA Queensland Lab, Brisbane, Australia
[4] School of Software, Tsinghua University, Beijing, China
songliang08@mails.thu.edu.cn,jimwang@tsinghua.edu.cn

Abstract. As business process management technology matures, organisations acquire more and more business process models. The management of the resulting collections of process models poses real challenges. One of these challenges concerns model retrieval where support should be provided for the formulation and efficient execution of business process model queries. As queries based on only structural information cannot deal with all querying requirements in practice, there should be support for queries that require knowledge of process model semantics. In this paper we formally define a process model query language that is based on semantic relationships between tasks in process models and is independent of any particular process modelling notation.

Key words: business process model, process model collection, business process model query, query language

1 Introduction

With the increasing maturity of business process management, more and more organisations need to manage large numbers of business process models, and among these may be models of high complexity. Processes may be defined along the entire value chain and over time a business may gather hundreds and even thousands of business process models. As an example consider Suncorp, one of the largest Australian insurers. Over the years, Suncorp have gone through a number of organizational mergers and acquisitions, as a result of which the company has accumulated over 3,000 process models for the various lines of insurance. In this context, support for business process retrieval, e.g. for the purposes of process reuse or process standardization, is a challenging proposition.

Several query languages exist that can be used to retrieve process models from a repository, e.g. BPMN-Q [1] or BP-QL [2, 3]. These languages are based on syntactic relationships between tasks and not on their semantic relationships. However, to deal with all querying requirements in practice, it is not enough to rely on only structural information of the process models but often requires knowledge of process model semantics. Consider for example the two process

Postprint

models in Fig. 1. They describe two variants of a business process for opening bank accounts using the BPMN notation [4]. These two variants could capture the way an account is opened in two different states where a bank operates, and could be part of a collection of various process models in all states where the bank operates. Assume that a business analyst needs to find out which branches always require an assessment of the customer's credit history when opening an account. In this case, only using the structural relationships between tasks, we cannot discern between the two variants, i.e. we would retrieve them both, since in both models there is a path from task "Receive customer request" to task "Analyse customer credit history". However, based on semantic relationships between tasks, we can observe that task "Analyse customer credit history" follows task "Receive customer request" in all instances of the first process variant, but this is not the case for one instance of the second variant (the one with task "Open VIP account"). Thus we can correctly exclude the second process variant from the results of the query, and return the first variant only.

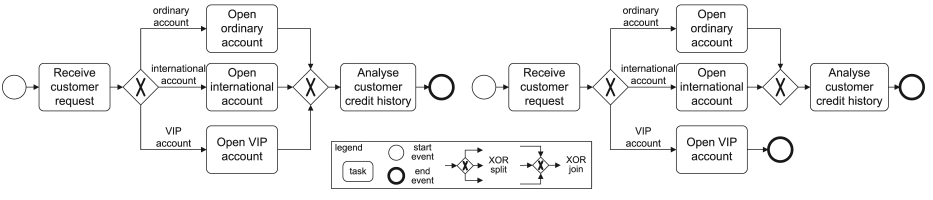


Fig. 1. Two variants of a business process for opening bank accounts.

In light of the above, we aim to address the development of a business process model query language based on semantic relationships between tasks in process models. We do so by proposing a new query language, namely *A Process-model Query Language (APQL)*, for retrieving required process models from model collections, e.g. process model repositories. This language relies on a number of basic temporal relationships between tasks which can be composed to obtain complex relationships between them. These predicates allow us to express queries that can discriminate over single process instances or task instances.

In this paper, we define both the syntax (Sect. 2) and semantics (Sect. 3) of APQL, provide examples to assist in understanding of the language definition (Sect. 4), discuss related work (Sect. 5), and finally conclude the paper (Sect. 6).

2 The Syntax of APQL

APQL is designed as a process model query language that is independent of the actual process modelling language used. This is important as in practice a variety of modelling languages are used (e.g. BPMN, EPCs) and the language should be generally applicable. Another important fact is that process models have a semantics and it should be possible to exploit this semantics when querying.

Based on the above design rationale, we define a set of 20 basic predicates to capture, in business process models, the occurrences of tasks as well as the semantic relationships between tasks. Below, the first two predicates capture the occurrence of a task t in *some* or *every* execution of a given process model r.

1. $posoccur(t, r)$: some execution of r exists where at least one instance of t occurs.
2. $alwoccur(t, r)$: in every execution of r, at least one instance of t occurs.

The next two predicates capture the *exclusive* and *concurrent* relationships between task occurrences. Note that these two predicates do not assume that an instance of t_1 or t_2 should eventually occur in a given process model r.

3. $exclusive(t_1, t_2, r)$: in every execution of r, it is never possible that an instance of t_1 and an instance of t_2 both occur.
4. $concur(t_1, t_2, r)$: t_1 and t_2 are not causally related, and in every execution of r, if an instance of t_1 occurs then an instance of t_2 occurs and vice versa.

Then we consider various forms of causal relationship between task occurrences. The relationship can be *precedence* (*pred*) or *succession* (*succ*), where one task may occur *immediately* or *eventually* preceding or succeeding another task. It may hold for *any* or *every* occurrence of the tasks in *some* or *every* process execution. Combining all these considerations results in 16 forms of causal relationships which are captured by the remaining 16 basic predicates as follows.

Let Φ be one of the following intermediate predicates,

1. $succ_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is eventually succeeded by an instance of t_2 (e.g. $...t_1...t_2...$).
2. $succ_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is eventually succeeded by an instance of t_2 (e.g. $t_1...t_1...t_2$).
3. $pred_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is eventually preceded by an instance of t_2 (e.g. $...t_2...t_1...$).
4. $pred_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is eventually preceded by an instance of t_2 (e.g. $t_2...t_1...t_1$).
5. $isucc_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is immediately succeeded by an instance of t_2 (e.g. $...t_1t_2...$).
6. $isucc_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is immediately succeeded by an instance of t_2 (e.g. $t_1t_2...t_1t_2$).
7. $ipred_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is immediately preceded by an instance of t_2 (e.g. $...t_2t_1...$).
8. $ipred_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is immediately preceded by an instance of t_2 (e.g. $t_2t_1...t_2t_1$).

Then

- $\Phi^{\forall}(t_1, t_2, r)$: $\Phi(t_1, t_2, i)$ holds for every process execution i of process model r, i.e. $\Phi(t_1, t_2, i)$ *always* holds in process r, and
- $\Phi^{\exists}(t_1, t_2, r)$: there *exists* some process execution i of process model r where $\Phi(t_1, t_2, i)$ holds, i.e. it is *possible* that $\Phi(t_1, t_2, i)$ holds in process r.

Next, the syntax of APQL is defined in the form of an abstract syntax, the advantages of which over a concrete syntax have been espoused by Meyer [5]. In

essence, in an abstract syntax we can avoid committing ourselves prematurely to specific choices for keywords or to the order of various statements.

In APQL a query is a sequence of *Assignments* combined with a *Predicate*.

$$\begin{aligned} \mathit{Query} &\triangleq s : \mathit{Assignments}; p : \mathit{Predicate} \\ \mathit{Assignments} &\triangleq \mathit{Assignment}^* \end{aligned}$$

The result is those process models that satisfy the *Predicate*. An *Assignment* assigns a *TaskSet* to a variable and when evaluating the *Predicate* every variable is replaced by the corresponding *TaskSet* (via the identifier of such task set).

$$\begin{aligned} \mathit{Assignment} &\triangleq v : \mathit{Varname}; ts : \mathit{TaskSet} \\ \mathit{Varname} &\triangleq \mathit{identifier} \end{aligned}$$

A *TaskSet* can be an enumeration of tasks or defined over other *TaskSet*s by *Construction* or *Application*. It can also be defined through a variable, a *TaskSetVar*.

$$\mathit{TaskSet} \triangleq \mathit{SetofTasks} \mid \mathit{Construction} \mid \mathit{Application} \mid \mathit{TaskSetVar}$$

A *Task* can be defined as a combination of a *TaskLabel* (a string) and a *SimDegree* (a real number). The idea is that one may be interested in *Tasks* of which the task label bears a certain degree of similarity to a given activity name. There are a number of definitions in the literature concerning label similarity and for a concrete implementation of the language one has to commit to one of these.

$$\begin{aligned} \mathit{SetofTasks} &\triangleq \mathit{Task}^* \\ \mathit{Task} &\triangleq l : \mathit{TaskLabel}; d : \mathit{SimDegree} \\ \mathit{TaskLabel} &\triangleq \mathit{string} \\ \mathit{SimDegree} &\triangleq \mathit{real}[0..1] \end{aligned}$$

A *TaskSetVar* is simply a variable that carries the identifier of the set of the tasks. Such a task set may be used in assignments.

$$\mathit{TaskSetVar} \triangleq \mathit{identifier}$$

A *TaskSet* can be composed from other *TaskSet*s through the application of the well-known set operators such as *union*, *difference*, and *intersection*. Another way to construct a *TaskSet* is by the application of a *TaskCompOp* (i.e. one of the basic predicates introduced earlier, but now interpreted as a function) on another *TaskSet*. In that case we have to specify whether we are interested in the tasks that have that particular relation with *all* or with *any* of the tasks in the *TaskSet*. For example, an application with *TaskSet* S, *TaskCompOp* *PosSuccAny* (i.e. $succ_{any}^{\exists}$) and *AnyAll* (indicator) *All*, is to yield those tasks that any instance of such a task succeeds an instance of each task in S in some process execution.

$$\begin{aligned} \mathit{Construction} &\triangleq ts_1, ts_2 : \mathit{TaskSet}; o : \mathit{Set_Op} \\ \mathit{Set_Op} &\triangleq \mathit{Union} \mid \mathit{Difference} \mid \mathit{Intersection} \\ \mathit{Application} &\triangleq ts : \mathit{TaskSet}; o : \mathit{TaskCompOp}; a : \mathit{AnyAll} \end{aligned}$$

$$
\begin{aligned}
\mathit{TaskCompOp} &\triangleq \mathit{Exclusive} \mid \mathit{Concur} \mid \\
&\quad \mathit{AlwSuccAny} \mid \mathit{AlwSuccEvery} \mid \mathit{AlwPredAny} \mid \mathit{AlwPredEvery} \mid \\
&\quad \mathit{PosSuccAny} \mid \mathit{PosSuccEvery} \mid \mathit{PosPredAny} \mid \mathit{PosPredEvery} \mid \\
&\quad \mathit{AlwISuccAny} \mid \mathit{AlwISuccEvery} \mid \mathit{PosISuccAny} \mid \mathit{PosISuccEvery} \mid \\
&\quad \mathit{AlwIPredAny} \mid \mathit{AlwIPredEvery} \mid \mathit{PosIPredAny} \mid \mathit{PosIPredEvery} \\
\mathit{AnyAll} &\triangleq \mathit{Any} \mid \mathit{All}
\end{aligned}
$$

A *Predicate* can consist of a simple *TaskPos*, with the intended semantics what the basic predicate *posoccur* specifies; a *TaskAlw*, with the intended semantics what the basic predicate *alwoccur* specifies; a *TaskRel*, with the intended semantics that all process models satisfying that particular relation should be retreived; or, it can be recursively defined as a binary or unary *Predicate* through the application of logical operators.

$$
\begin{aligned}
\mathit{Predicate} &\triangleq \mathit{TaskPos} \mid \mathit{TaskAlw} \mid \mathit{TaskRel} \mid \mathit{Bin_Predicate} \mid \mathit{Un_Predicate} \\
\mathit{Bin_Predicate} &\triangleq o : \mathit{BinLogOp}; p_1, p_2 : \mathit{Predicate} \\
\mathit{Un_Predicate} &\triangleq o : \mathit{UnLogOp}; p : \mathit{Predicate} \\
\mathit{BinLogOp} &\triangleq \mathit{And} \mid \mathit{Or} \\
\mathit{UnLogOp} &\triangleq \mathit{Not} \\
\mathit{TaskPos} &\triangleq l : \mathit{TaskLabel}; d : \mathit{SimDegree} \\
\mathit{TaskAlw} &\triangleq l : \mathit{TaskLabel}; d : \mathit{SimDegree}
\end{aligned}
$$

A *TaskRel* can be 1) a relationship between a *Task* and a *TaskSet* to check whether the *Task* occurs in the *TaskSet* (*TaskInTaskSet*), 2) a relationship between a *Task* and a *TaskSet* and involving a *TaskCompOp* and an *AnyAll* indicator to determine whether the *Task* has the *TaskCompOp* relationship with any/all *Tasks* in the *TaskSet* (*Task_TaskSet*), 3) a relationship between two *Tasks* involving a *TaskCompOp* predicate determining whether for the two *Tasks* that predicate holds (*Task_Task*), 4) a relationship between two *TaskSets* involving a *TaskCompOp* and an *AnyAll* indicator to determine whether the *Tasks* in those *TaskSets* all have the *TaskCompOp* relationship to each other or whether for each *Task* in the first *TaskSet* there is a corresponding *Task* in the second *TaskSet* for which the relationship holds (*Elt_TaskSet_TaskSet*), or 5) a relationship between two *TaskSets* determined by a set comparison operator (*Set_TaskSet_TaskSet*).

$$
\begin{aligned}
\mathit{TaskRel} &\triangleq \mathit{TaskInTaskSet} \mid \mathit{Task_TaskSet} \mid \\
&\quad \mathit{Task_Task} \mid \mathit{Elt_TaskSet_TaskSet} \mid \\
&\quad \mathit{Set_TaskSet_TaskSet} \\
\mathit{TaskInTaskSet} &\triangleq t : \mathit{Task}; ts : \mathit{TaskSet} \\
\mathit{Task_TaskSet} &\triangleq t : \mathit{Task}; ts : \mathit{TaskSet}; \\
&\quad o : \mathit{TaskCompOp}; a : \mathit{AnyAll} \\
\mathit{Task_Task} &\triangleq t_1, t_2 : \mathit{Task}; o : \mathit{TaskCompOp}
\end{aligned}
$$

$$
\begin{array}{rcl}
\mathit{Elt_TaskSet_TaskSet} & \triangleq & \mathit{ts}_1, \mathit{ts}_2 : \mathit{TaskSet}; o : \mathit{TaskCompOp}; \\
 & & a : \mathit{AnyAll} \\
\mathit{Set_TaskSet_TaskSet} & \triangleq & \mathit{ts}_1, \mathit{ts}_2 : \mathit{TaskSet}; o : \mathit{SetCompOp} \\
\mathit{SetCompOp} & \triangleq & \mathit{Identical} \mid \mathit{Subsetof} \mid \mathit{Overlap}
\end{array}
$$

3 The Semantics of APQL

We use denotational semantics to formally describe the semantics of APQL and adopt the notation in [5]. For each nonterminal T we introduce a semantic function M_T which defines the meaning of the nonterminal in terms of its parts.

First, we introduce some auxiliary notation in order to facilitate the subsequent definition of the semantics.

Definition 1 (overriding union). *The overriding union of $f : X \to Y$ by $g : X \to Y$, denoted as $f \oplus g$, is defined by $g \cup f \backslash \{(x, f(x)) \mid x \in dom(f) \cap dom(g)\}$.*

With the set of 20 basic predicates defined in the previous section, we use $\mathbb{BP}_u$ to denote the set of two *unary predicates* $\{posoccur, alwoccur\}$ which specify unary task relations, and similarly we use $\mathbb{BP}_b$ to denote the set of 18 *binary predicates* which specify binary task relations. The following two definitions introduce a higher order predicate that takes as input a unary or binary predicate, respectively. Note that the semantics of each predicate (ϕ/ψ) is language independent. For a task t in process model N, $L_N(t)$ specifies the label of t. A process model may have *silent tasks* which do not capture any task or activity in the process but are used for modelling purposes, e.g. a silent task used to capture an internal action that cannot observed by external uses. For a silent task t, we let $L(t) = \tau$.

Definition 2. *Let N be a process model and T the set of tasks in N, for $t_1, t_2 \in T$ and $\phi \in \mathbb{BP}_b$*

$$\mathsf{ref}^{\phi}(t_1, t_2, N) = \begin{cases} \phi(t_1, t_2, N) \textbf{ if } L_N(t_1) \neq \tau \wedge L_N(t_2) \neq \tau \\ \mathit{FALSE} \qquad \textbf{otherwise} \end{cases}$$

i.e. the relation ϕ should hold between t_1 and t_2 in N if both are non-silent tasks.

Definition 3. *Let N be a process model and T the set of tasks in N, for $t \in T$ and $\psi \in \mathbb{BP}_u$*

$$\mathsf{ref}^{\psi}(t, N) = \begin{cases} \psi(t, N) \textbf{ if } L_N(t) \neq \tau \\ \mathit{FALSE} \textbf{ otherwise} \end{cases}$$

i.e. the relation ψ should hold for t in N if t is a non-silent task.

As queries may use variables, we must know their values during query evaluation. A *Binding* is an assignment of task sets to variables:

$$\mathit{Binding} \triangleq \mathit{ProcessModel} \times \mathit{Varname} \rightarrowtail 2^{\mathit{Task}}$$

Queries are applied to a repository of process models, i.e.

$$Repository \triangleq 2^{ProcessModel}$$

A process model r consists of a collection of tasks T_r. For each task t in process model r we can retrieve its label as $L_r(t)$. Label similarity can be determined through the function Sim, where $Sim(l_1, l_2)$ determines the degree of similarity between labels l_1 and l_2 (which yields a value in the range [0,1]). Note that Sim is a parameter of the approach in which case one can choose his/her own similarity notion and the corresponding function Sim returns the similarity evaluation result to this parameter.

The query evaluation function M_{Query} takes a query and a repository as input and yields those process models in that repository that satisfy the query:

$$M_{Query} : Query \times Repository \rightarrow 2^{ProcessModel}$$

This function is defined as follows:

$$M_{Query}[q : Query, R : Repository] \triangleq M_{Predicate}(q.p, R, M_{Assignments}(q.s, R, \varnothing))$$

The evaluation of the query evaluation function depends on the evaluation of the predicate involved and the assignments involved. When evaluating a sequence of assignments we have to remember the values that have been assigned to the variables involved. Inititally this set of assignments is empty.

$$M_{Assignments} : Assignments \times Repository \times Binding \rightarrow Binding$$

The result of a sequence of assignments is a binding where the variables used in the assignments are bound to sets of tasks. If a variable was already assigned a set of tasks in an earlier assignment in the sequence the latest assignment takes precedence over the earlier assignment.

$M_{Assignments}[s : Assignments, R : Repository, B : Binding] \quad \triangleq$
  if $\neg(s.TAIL).EMPTY$ **then**
    $M_{Assignments}(s.TAIL, R, B \oplus M_{Assignment}(s.FIRST, R, B))$
  else B

The result of an individual assignment is also a binding where the variable is linked to the set of tasks involved.

$$M_{Assignment} : Assignment \times Repository \times Binding \rightarrow Binding$$

$M_{Assignment}[a : Assignment, R : Repository, B : Binding)] \quad \triangleq$
  $\{((r, a.v), M_{TaskSet}(a.ts, R, B)(r)) \mid r \in R\}$

A predicate can be evaluated in the context of a repository and a binding and the result is a set of process models from that repository.

$$M_{Predicate} : Predicate \times Repository \times Binding \rightarrow 2^{ProcessModel}$$

A predicate may yield all process models in the repository that contain a task sufficiently similar to that task (with respect to the task label and similarity

degree). A predicate may also specify relationship between tasks (i.e. a *TaskRel*) in which case it yields all the process models that satisfy this relationship. A conjunction yields the intersection of the process models of the predicates involved, while a disjunction yields the union. The negation of a predicate yields the process models in the repository that do not satisfy the predicate.

$$
\begin{array}{l}
M_{Predicate}(p : Predicate, R : Repository, B : Binding) \quad \triangleq \\
\quad \textbf{case } p \textbf{ of} \\
\quad\quad TaskPos \Rightarrow \{r \in R \mid \exists t \in T_r[Sim(p.l, L_r(t)) \geq p.d \wedge posoccur(t, r)]\} \\
\quad\quad TaskAlw \Rightarrow \{r \in R \mid \exists t \in T_r[Sim(p.l, L_r(t)) \geq p.d \wedge alwoccur(t, r)] \\
\quad\quad TaskRel \Rightarrow M_{TaskRel}(p, R, B) \\
\quad\quad Bin_Predicate \Rightarrow \\
\quad\quad\quad \textbf{case } p.o \textbf{ of} \\
\quad\quad\quad\quad And \Rightarrow M_{Predicate}(p.p_1, R, B) \cap M_{Predicate}(p.p_2, R, B) \\
\quad\quad\quad\quad Or \Rightarrow M_{Predicate}(p.p_1, R, B) \cup M_{Predicate}(p.p_2, R, B) \\
\quad\quad\quad \textbf{end} \\
\quad\quad Un_Predicate \Rightarrow R \backslash M_{Predicate}(p, R, B) \\
\quad \textbf{end}
\end{array}
$$

A *TaskRel* in the context of a repository and a binding yields a set of process models in that repository.

$$M_{TaskRel} : TaskRel \times Repository \times Binding \rightarrow 2^{ProcessModel}$$

A *TaskRel* can be used to determine whether a task in a process model occurs in a given task set, whether a given basic predicate holds between a task in a process model and one or all tasks in a given task set, whether a given basic predicate holds between tasks in a process model, whether a given basic predicate holds between two or between all tasks in two given task sets, or whether a given set comparison relation holds between two given task sets.

$$
\begin{array}{l}
M_{TaskRel}(tr : TaskRel, R : Repository, B : Binding) \quad \triangleq \\
\quad \textbf{case } tr \textbf{ of} \\
\quad\quad TaskInTaskSet \Rightarrow \\
\quad\quad\quad \{r \in R \mid \exists v \in M_{TaskSet}(tr.ts, R, B)(r)[Sim(tr.t.l, L_r(v)) \geq tr.t.d]\} \\
\quad\quad Task_TaskSet \Rightarrow \\
\quad\quad\quad \textbf{case } tr.a \textbf{ of} \\
\quad\quad\quad\quad Any \Rightarrow \{r \in R \mid \exists t_1 \in T_r \; \exists t_2 \in M_{TaskSet}(tr.ts, R, B)(r) \\
\quad\quad\quad\quad\quad [Sim(tr.t.l, L_r(t_1)) \geq tr.t.d \wedge rel^{tr.o}(t_1, t_2, r)]\} \\
\quad\quad\quad\quad All \Rightarrow \{r \in R \mid \exists t_1 \in T_r \; \forall t_2 \in M_{TaskSet}(tr.ts, R, B)(r) \\
\quad\quad\quad\quad\quad [Sim(tr.t.l, L_r(t_1)) \geq tr.t.d \wedge rel^{tr.o}(t_1, t_2, r)]\} \\
\quad\quad\quad \textbf{end} \\
\quad\quad Task_Task \Rightarrow \{r \in R \mid \exists v_1, v_2 \in T_r[Sim(tr.t_1.l, L_r(v_1)) \geq tr.t_1.d \wedge \\
\quad\quad\quad\quad\quad Sim(tr.t_2.l, L_r(v_2)) \geq tr.t_2.d \wedge rel^{tr.o}(v_1, v_2, r)]\} \\
\quad\quad Elt_TaskSet_TaskSet \Rightarrow \\
\quad\quad\quad \textbf{case } tr.a \textbf{ of} \\
\quad\quad\quad\quad Any \Rightarrow \{r \in R \mid \exists t_1 \in M_{TaskSet}(tr.ts_1, R, B)(r) \\
\quad\quad\quad\quad\quad\quad \exists t_2 \in M_{TaskSet}(tr.ts_2, R, B)(r)[rel^{tr.o}(t_1, t_2, r)]\} \\
\quad\quad\quad\quad All \Rightarrow \{r \in R \mid \forall t_1 \in M_{TaskSet}(tr.ts_1, R, B)(r) \\
\quad\quad\quad\quad\quad\quad \forall t_2 \in M_{TaskSet}(tr.ts_2, R, B)(r)[rel^{tr.o}(t_1, t_2, r)]\} \\
\quad\quad\quad \textbf{end}
\end{array}
$$

Set_TaskSet_TaskSet ⇒
case *tr.o* **of**
Identical ⇒
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) = M_{TaskSet}(tr.ts_2, R, B)(r)\}$
Subsetof ⇒
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) \subseteq M_{TaskSet}(tr.ts_2, R, B)(r)\}$
Overlap ⇒
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) \cap M_{TaskSet}(tr.ts_2, R, B)(r) \neq \varnothing\}$
end
end

A *TaskSet* within the context of a repository and a binding yields a mapping which assigns to each process model in the repository the collection of tasks within that model that satisfy the restriction imposed by the *TaskSet*.

$$M_{TaskSet} : TaskSet \times Repository \times Binding \rightarrow (ProcessModel \rightarrow 2^{Task})$$

When a *TaskSet* is a set of tasks, then for each process model the result is the set of tasks within that process model that are sufficiently similar to at least one of the tasks in that *TaskSet*. When the *TaskSet* is a variable, then the evaluation is similar except that the task set used is the task set currently bound to that variable. *TaskSets* can also be formed through *Construction* (where the set operators union, difference, and intersection are used) or *Application* (where task sets are formed through set comprehension, i.e. they are defined through properties that they have - these properties relate to the basic predicates).

$M_{TaskSet}(tks : TaskSet, R : Repository, B : Binding) \triangleq$
case *tks* **of**
SetofTasks ⇒
$\{(r, \{t \in T_r \mid \exists_{1 \leq i \leq tks.LENGTH}[Sim(tks(i).l, L_r(t)) \geq tks(i).d]\}) \mid r \in R\}$
TaskSetVar ⇒
$\{(r, X) \mid r \in R\}$ **where**
$X = \begin{cases} B(r, tks) & \textbf{if } (r, tks) \in dom(B) \\ \varnothing & \textbf{otherwise} \end{cases}$
Construction ⇒
case *tks.o* **of**
Union ⇒
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \cup M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
Difference ⇒
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \backslash M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
Intersection ⇒
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \cap M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
end
Application ⇒
case *tks.a* **of**
Any ⇒
$\{(r, \{t \in T_r \mid \exists v \in M_{TaskSet}(tks.ts, R, B)(r)[rel^{tks.o}(t, v, r)]\}) \mid r \in R\}$
All ⇒
$\{(r, \{t \in T_r \mid \forall v \in M_{TaskSet}(tks.ts, R, B)(r)[rel^{tks.o}(t, v, r)]\}) \mid r \in R\}$
end
end

4 Examples of APQL Queries

In this section we present some sample queries and show how they can be captured in APQL in order to further illustrate the language. The sample queries, specified in natural language, are listed below (which are numbered Q_1 to Q_{10}). In these queries, by default the value for the *AnyAll* identifier, when applicable, is *all*, and by default the value for the SimDegree is 1. According to the abstract syntax of APQL in Sect. 2, Fig. 2 shows the grammar trees for queries Q_1 to Q_6, and Fig. 3 shows the grammar trees for queries Q_7 to Q_{10}. Note that in the following A to L are task labels (i.e. activity names).

Q_1. Select all process models where task A occurs in some process execution and task B occurs in every process execution.
Q_2. Select all process models where in every process execution it is possible that task A occurs before task D.
Q_3. Select all process models where in every process execution task A always occurs before task D.
Q_4. Select all process models where in some process execution it is possible that task A occurs before task B and task B occurs before task K.
Q_5. Select all process models where in some process execution task A always occurs before task B.
Q_6. Select all process models where task B occurs in parallel with task C.
Q_7. Select all process models where task B occurs in parallel with task C and where task A occurs in parallel with task H.
Q_8. Select all process models where in every process execution task B and task C never occur together.
Q_9. Select all process models where in some process execution the immediate predecessors of task H are among the immediate successors of task B.
Q_{10}. Select all process models where in some process execution the immediate predecessors of task H may occur after the common immediate successors of task B and task C.

In order to illustrate the formal semantics of APQL, a number of process models, represented in BPMN, are presented in Fig. 4. For each sample query above and for each model it is indicated whether the model is part of the answer to the query (in that case the box corresponding to the query is ticked otherwise the box is not ticked). Note that in some models tasks with the same label occur (e.g. there are two tasks labeled A in model (5) in Fig. 4), in which case, APQL will treat these tasks as same tasks during query evaluation [1].

5 Related Work

Mindful of the importance of query languages for business process models, the Business Process Management Initiative (BPMI) proposed to define a standard

[1] Note that APQL is query language rather than a process modelling language.

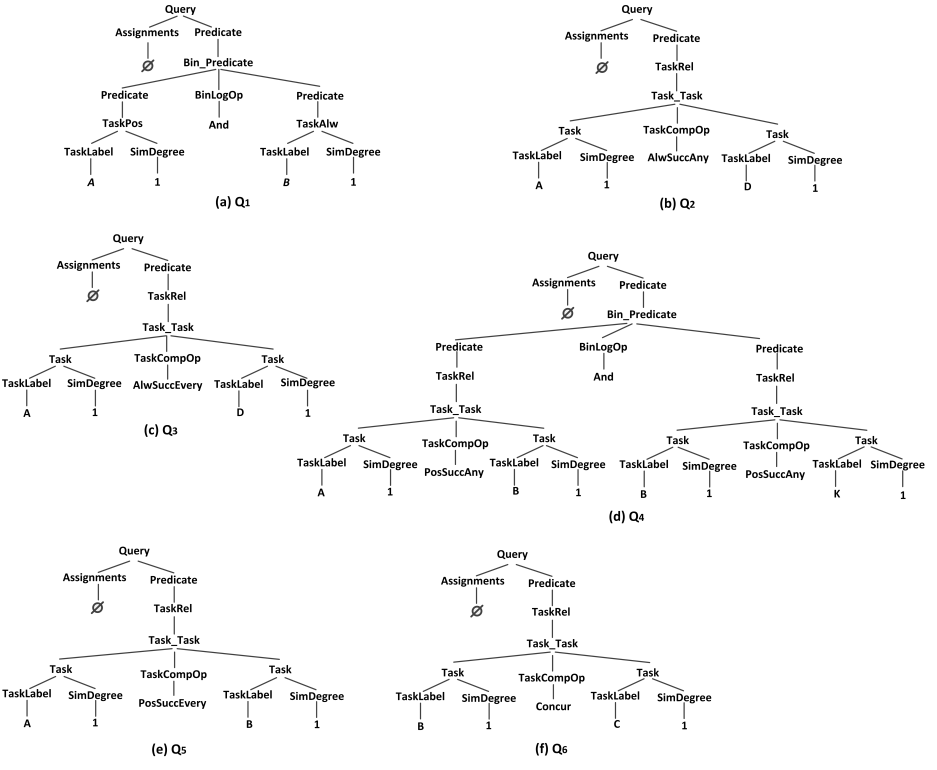
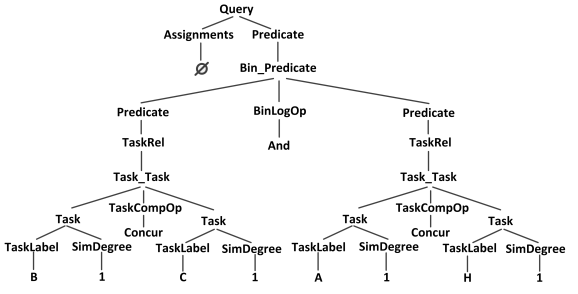


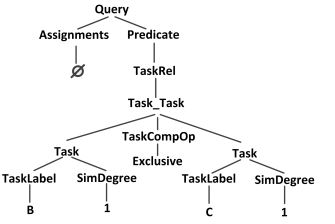
Fig. 2. The APQL grammar trees of sample queries $Q_1 - Q_6$

business process model query language in 2004[2]. While such a standard has never been published, two major research efforts have been dedicated to the development of query languages for process models. One is known as BP-QL [3], a visual query language based on an abstract representation of BPEL and supported by a formal model of graph grammars for processing of queries. BP-QL can be used to query process specifications written in BPEL rather than possible executions, and ignores the run-time semantics of certain BPEL constructs such as conditional execution and parallel execution. The other effort, namely BPMN-Q [1, 6], is also a visual query language which extends a subset of the BPMN modelling notation and supports graph-based query processing. Similarly to BP-QL, BPMN-Q only captures the structural (i.e., syntactical) relations between tasks, and not their behavioral relationships. In [7], the authors explore the use of an information retrieval technique to derive similarities of activity names, and develop an ontological expansion of BPMN-Q to tackle the problem of querying business processes that are developed with different terminologies. A framework of tool support for querying process model repositories using BPMN-Q and its extensions is presented in [8]. Recently, in [9], an approach that applies index-

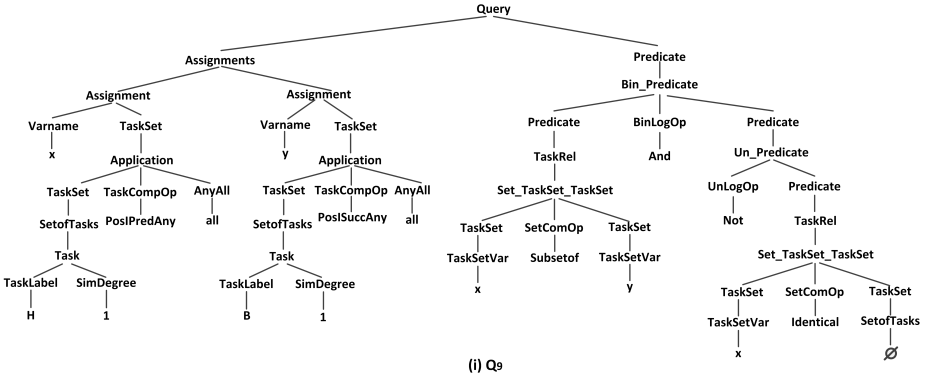
[2] http://www.bpmi.org/downloads/BPMI_Phase_2.pdf



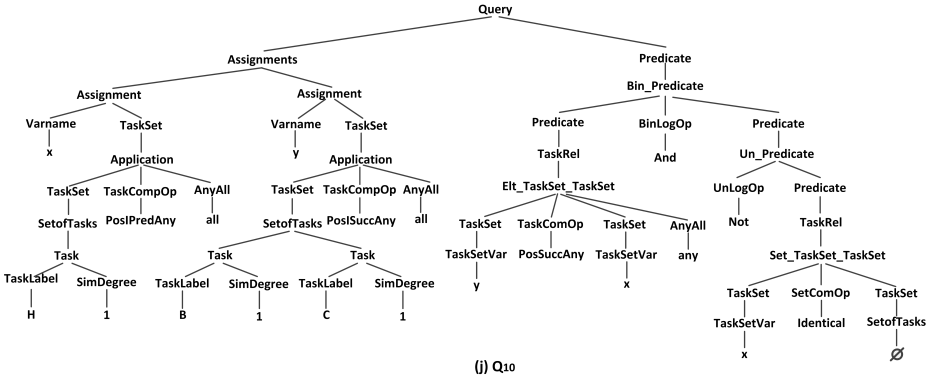
(g) Q_7



(h) Q_8



(i) Q_9



(j) Q_{10}

Fig. 3. The APQL grammar trees of sample queries $Q_7 - Q_{10}$

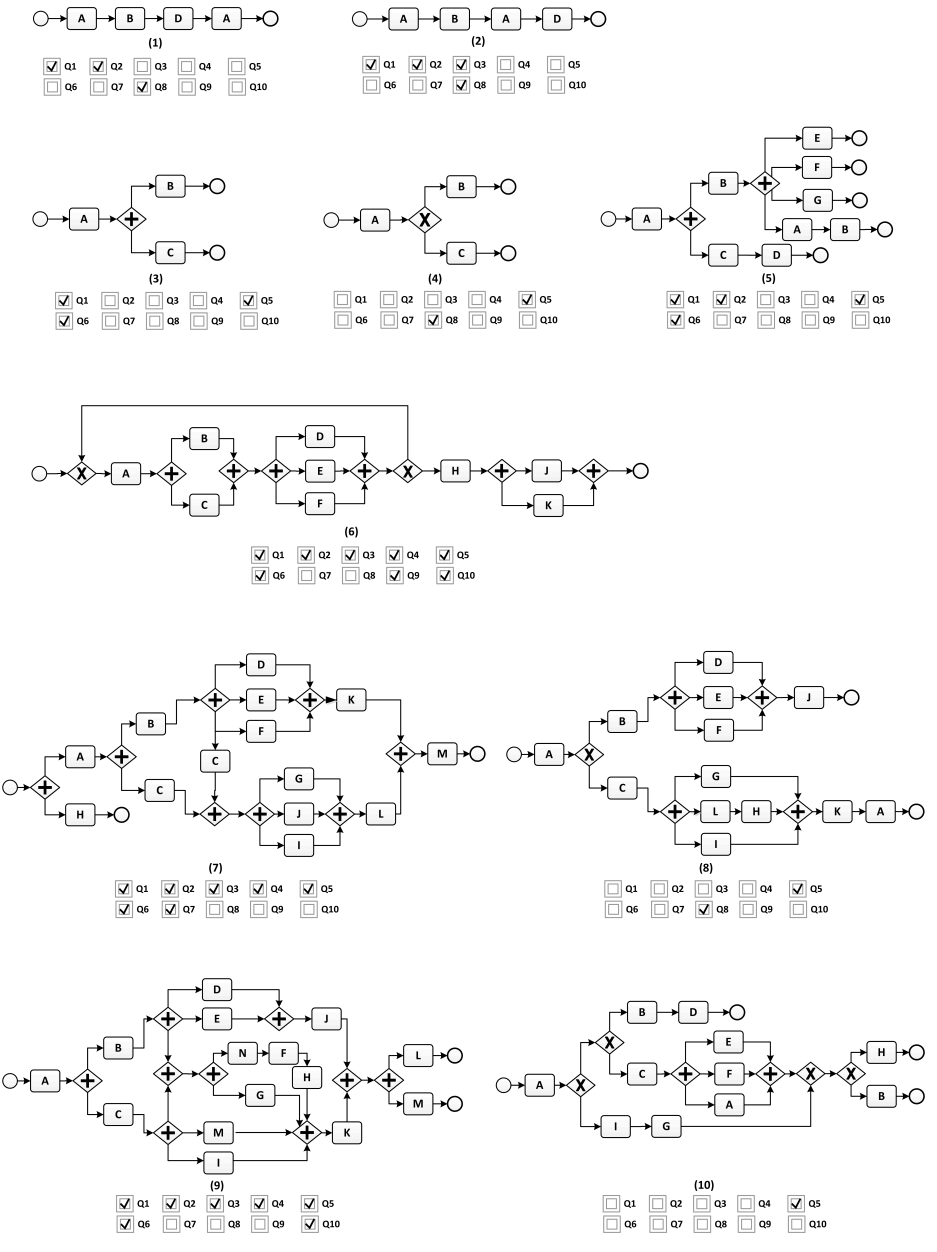


Fig. 4. A list of BPMN business process models and evaluation of sample queries $Q_1 - Q_{10}$ over these process models.

ing method based on the (graph-based) flow relation between tasks in BPMN process diagrams is proposed for efficient processing of BPMN-Q queries.

APQL presents three distinguishing features compared to the above languages. First, its abstract syntax and semantics have been purposefully defined to be independent of a specific process modelling language (such as BPEL or BPMN). This will allow APQL and its query evaluation technique to be implemented for a variety of process modelling languages. Second, APQL can express all possible temporal-ordering relations (precedence/succession, concurrence and exclusivity) between individual tasks, between an individual task and a set of tasks as well as between different sets of tasks. Third, APQL querying constructs need to be evaluated over the execution semantics of process models, rather than their structural relations. In fact, structural characteristics alone are not able to capture all possible order relations among tasks which can occur during execution, in particular with respect to cycles and task occurrences.

In addition to the development of process model query languages, other techniques are available in the literature which can be useful for querying process model repositories. In [10, 11], the authors focus on querying the content of business process models based on metadata search. VisTrails system [12] allows users to query scientific workflows by example and to refine workflows by analogies. WISE [13] is a workflow information search engine which supports keyword search on workflow hierarchies. In [14], the authors use graph reduction techniques to find a match to the query graph in the process graph for querying process variants, and the approach works on only acyclic graphs. In [15, 16, 17], a group of similarity-based techniques have been proposed which can be used to support process querying. In [18], a technique to query process model repositories is proposed based on an input Petri net. Finally, in [19], the notion of behavioural profile of a process model is defined, which captures dedicated behavioural relations like exclusiveness or potential occurrence of activities. However, these behavioural relations are derived from the structure of a process model. Thus, for the reasons mentioned above, behavioral profiles only provide an approximation of a process model's behavior, whereas APQL can precisely determine whether or not a process model satisfies a given query.

6 Conclusions

This paper contributes an innovative language, namely APQL, for querying process model repositories. APQL provides three main advantages over the state of the art. First, the language is expressive since it allows users to specify all possible order relationships among tasks or sets thereof. Second, the language is precise, since APQL queries are defined for evaluation over process model behavior, while existing query languages only support structural process characteristics. Third, the language's syntax and semantics are defined independently of any specific process modeling language.

The next stage is to operationalise APQL and the main task is to develop a technique for evaluation of APQL queries. This evaluation technique could

be designed on top of a well established mathematical technique for describing behavioural semantics, e.g. Petri nets. One challenge though, when it comes to determining semantic relationships between tasks, is how to determine these relationships in a feasible manner (i.e. without suffering from the well-known state space explosion problem).

Currently APQL only focuses on the control flow perspective of business process models. In the future, we will extend the language definition in order to include other process perspectives such as data and participating resources. Moreover, we plan to run structured interviews with domain experts to assess the overall ease of use and usefulness of APQL.

Acknowledgements

Song and Wang are supported by the National Basic Research Program of China (2009CB320700), the National High-Tech Development Program of China (2008AA042301), the Project of National Natural Science Foundation of China (90718010), and the Program for New Century Excellent Talents in University of China. ter Hofstede, La Rosa and Polyvyanyy are partly supported by the ARC Linkage Grant "Facilitating Business Process Standardization and Reuse" (LP110100252). In 2010 and 2011, ter Hofstede was a senior visiting scholar of Tsinghua University. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Awad, A.: BPMN-Q: A language to query business processes. In: Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07). Volume P-119 of LNI., GI (2007) 115–128
2. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: Proceedings of the 32nd International Conference on Very Large Data Bases. ACM (2006) 343–354
3. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes with BP-QL. Inf. Syst. **33**(6) (September 2008) 477–507
4. OMG: Business Process Model and Notation (BPMN) version 2.0. (January 2011) Available via `http://www.omg.org/spec/BPMN/2.0`.
5. Meyer, B.: Introduction to the Theory of Programming Languages. Prentice-Hall (1990)
6. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: Proceedings of the 6th International Conference of Business Process Management. Volume 5240 of Lecture Notes in Computer Science, Springer (2008) 326–341
7. Awad, A., Polyvyanyy, A., Weske, M.: Semantic querying of business process models. In: Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference. IEEE Computer Society (2008) 85–94

8. Sakr, S., Awad, A.: A framework for querying graph-based business process models. In: Proceedings of the 19th International Conference on World Wide Web. ACM (2010) 1297–1300
9. Awad, A., Sakrb, S.: On efficient processing of bpmn-q queries. Computers in Industry **63**(9) (2012) 867–881
10. Vanhatalo, J., Koehler, J., Leymann, F.: Repository for business processes and arbitrary associated metadata. In: Proceedings of the 4th International Conference of Business Process Management. Volume 4102 of Lecture Notes in Computer Science, Springer (2006) 426–431
11. Wasser, A., Lincoln, M., Karni, R.: ProcessGene Query - a tool for querying the content layer of business process models. In: Proceedings of the 4th International Conference of Business Process Management. Volume 4102 of Lecture Notes in Computer Science, Springer (2006) 1–8
12. Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.T.: Querying and reusing workflows with vistrails. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM (2008) 1251–1254
13. Shao, Q., Sun, P., Chen, Y.: Wise: A workflow information search engine. In: Proceedings of the 25th International Conference on Data Engineering. IEEE Computer Society (2009) 1491–1494
14. Lu, R., Sadiq, S.W.: Managing process variants as an information resource. In: Proceedings of the 4th International Conference of Business Process Management. Volume 4102 of Lecture Notes in Computer Science, Springer (2006) 426–431
15. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Process equivalence: Comparing two process models based on observed behavior. In: Proceedings of the 4th International Conference of Business Process Management. Volume 4102 of Lecture Notes in Computer Science, Springer (2006) 129–144
16. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling. Volume 67 of CRPIT, ACS (2007) 71–80
17. Dongen, B., Dijkman, R., Mendling, J.: Measuring similarity between business process models. In: Proceedings of the 20th International Conference on Advanced Information Systems Engineering. Volume 5074 of Lecture Notes in Computer Science, Springer (2008) 450–464
18. Jin, T., Wang, J., Wu, N., La Rosa, M., ter Hofstede, A.: Efficient and accurate retrieval of business process models through indexing - (short paper). In: Proceedings of the On The Move Confederated International Conferences. Volume 6426 of Lecture Notes in Computer Science, Springer (2010) 402–409
19. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. Software Engineering, IEEE Transactions on **37**(3) (2011) 410 –429