Process Instance Query Language and the Process Querying Framework

Jose Miguel Pérez Álvarez, Antonio Cancela Díaz, Luisa Parody, Antonia M. Reina Quintero, and María Teresa Gómez-López

Abstract The use of Business Process Management Systems (BPMSs) allows companies to manage the data that flows through process models (business instances) and to monitor all the information and actions concerning a process execution. In general, the retrieval of this information is used not only to measure whether the process works as expected but also to enable assistance in future process improvements by means of a postmortem analysis. This chapter shows how the measures extracted from the process instances can be employed to adapt business process executions according to other instances or other processes, thereby facilitating the adjustment of the process behavior at run-time to the organization needs. A language, named Process Instance Query Language (PIQL), is introduced. This language allows business users to query the process instance measures at run-time. These measures may be used both inside and outside the business processes. As a consequence, PIQL might be used in various scenarios, such as in the enrichment of the information used in Decision Model and Notation tables, in the determination of the most suitable business process to execute at run-time, and in the query of the instance measures from a dashboard. Finally, an example is introduced to demonstrate PIQL.

J. M. Pérez Álvarez (✉)
NAVER LABS Europe, Meylan, France
e-mail: jm.perez@naverlabs.com

A. C. Díaz · A. M. R. Quintero · M. T. Gómez-López
Dto. Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Sevilla, Spain
e-mail: acancela@us.es; reinaqu@us.es; maytegomez@us.es

L. Parody
Dto. Métodos Cuantitativos, Universidad Loyola Andalucía, Dos Hermanas, Spain
e-mail: mlparody@uloyola.es

1 Introduction

The management of the information regarding the current and past status of an organization is crucial. The analysis of this information can be essential for different purposes, such as to determine whether the objectives of the organization are achieved, to ascertain whether the company evolves as was planned, to verify whether decision-making processes must modify company's evolution [5], and to study how the management of the company can be improved. In this respect, it could be stated that the sooner the state of the company is ascertained, the sooner the decisions to "redirect" the company could be made. The status of an organization is reflected in the information systems used to support its operations and, frequently, by the BPMSs that are often used in business process-oriented organizations.

A BPMS enables the company's activities to be assisted by means of automating and monitoring the technical environment to achieve the defined objectives. In this context, the status of the organization can be extracted from the data generated during the execution of its activities, both stored in external repositories or by the BPMSs [4]. Although the extraction of knowledge from external data repositories is usually performed through SQL as the standard language that queries relational data, the extraction of the status of the organization from a BPMS according to the process executions has no standard form. However, BPMSs also help experts to extract information about processes: how many instances of a process have been executed, how many instances of a process are currently being executed, how the process was executed (when it was started, finished, and/or canceled), and how the activities of a process instance have been executed (who executed it, when it was executed, the specific values of the data consumed and generated, or whether the activity was canceled). In other words, BPMSs allow measuring how a specific process works.

The state of executions of the processes can be inferred from measures of the process instances, and consequently, these measures enable the improvement of processes themselves. Note, however, that these measurements enable not only the process execution improvement or a process model redesign but also the adaptation of the current process instances according to the rest of the instances and available resources at any given moment. These two different usages of measures are aligned with the two types of monitoring at run-time: active and passive monitoring [6]. On the one hand, active monitoring implies the automatic evaluation of measurements since the system includes the adequate tools to detect deviations and notify them at run-time. On the other hand, passive monitoring does not evaluate metrics automatically, but it is the user who proactively requests their evaluation. The monitoring of processes allows companies to ascertain whether certain performance indicators, such as Key Performance Indicators (KPIs) and Process Performance Indicators (PPIs), support the achievement of companies' goals. KPIs represent the companies' business goals and are described in their strategic plans [7], whereas PPIs are related to the processes instantiated in the companies to achieve their goals. The incorporation of measures regarding the process instances into the business

process execution can be crucial since it allows the adaptation of the process execution according to the performance of other business process instances at any given moment.

To measure the degree of efficiency of the process output and also the level of achievement of the company goals supported by KPIs and PPIs, the involved measures must be queried. This chapter introduces Process Instance Query Language (PIQL) as a mechanism to query these measures regarding the business process status, thereby making it possible to adapt the current execution of the instances based on the obtained measures. This language has been designed to be easily understandable for nontechnical people, since the managers may not be familiar with complex query languages. PIQL is close to the natural language and brings flexibility and agility to organizations because it empowers the managers to adapt many aspects of the organization by themselves.

The measures obtained by means of PIQL queries are useful in many scenarios, e.g., internally, within the process instance under execution to make decisions based on its status, or externally, in third-party applications that show the measured values in a dashboard. For example, PIQL allows business experts to extract measures to decide the assignment of a task to a person, depending on the number of activities executed by him/her in the past, decide when a process must be started or stopped, or determine whether a process evolves as expected. We have foreseen three scenarios where the use of PIQL can be useful: decision-making using Decision Model and Notation (DMN) [9], monitoring the evolution of the processes by means of external dashboards, and management of business processes to decide the most suitable process to execute and when it should be executed.

In order to illustrate a case where PIQL is used, an example is employed that consists of a set of business processes related to component assembly in industry. The example highlights the need to incorporate information about the instances of other processes to adapt the execution of the current process according to the needs and available resources at various moments.

The chapter is organized as follows: Firstly, Sect. 2 defines the main concepts needed to understand the rest of the chapter. Secondly, Sect. 3 explains a scenario to show the applicability of PIQL. Thirdly, Sect. 4 describes PIQL. Section 5 then details the implementation of PIQL. Section 6 applies PIQL to the motivating scenario, and Sect. 7 relates PIQL to the Process Querying Framework (PQF) introduced in [11]. Finally, conclusions are drawn and future work is proposed in Sect. 8.

2 Background

Processes of companies can be described using *business process models*. These models represent the activities of a company, and they can be automated by means of a Business Process Management System (BPMS). To lay the foundations of the

basic business process terminology used in the chapter, we adopt the following definitions provided by Weske in [12]:

Definition 2.1 A *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment.

Definition 2.2 A Business Process Management System (BPMS) is a generic software system that is driven by explicit process representations to coordinate the enactment of business processes.

Definition 2.3 A *business process model* is an abstract representation of a business process that consists of a set of activity models and execution constraints between them.

Definition 2.4 A *business process instance* represents a specific case of an operational business of a company and consists of a sequence of activity instances.

Activities in business processes can be manual activities (that is, not supported by information systems), user interaction activities (performed by workers using information systems), or system activities (which are executed by information systems without any human involvement). Activities can also be classified into atomic, those that cannot be decomposed, and non-atomic. An activity instance is a specific case of an activity model. In other words, business process models act as blueprints for business process instances, while activity models act as blueprints for activity instances.

Definition 2.5 A *task* is an activity that cannot be decomposed.

BPMSs help companies to monitor the execution of business processes by means of performance measures or indicators. These performance measures are usually graphically represented in dashboards, which are software tools that help experts to analyze data, detect business problems, and make decisions. KPIs and PPIs are two kinds of performance indicators. KPIs are employed to describe what the company wants to achieve (e.g., increase in the number of assembled components by 10%). When the indicators are related to the measurements of the processes of the organization, PPIs are used (e.g., reduce by 15% the time of the assembly process). Several of these by measures, above all those related to PPIs, are usually stored by BPMSs and can be obtained by observing processes and analyzing these observations [1, 2]. The aforementioned notions can be defined as follows:

Definition 2.6 A *dashboard* is a tool commonly used in business to visually represent the indicators that are related to business goal achievements.

Definition 2.7 A *Key Performance Indicator (KPI)* is an indicator that measures the performance of key activities and initiatives that lead to the success of business goals. A KPI often involves financial and customer metrics to describe what the company wants. Achievement of KPIs indicates whether the company is attaining its strategic goals or not.

Definition 2.8 A *Process Performance Indicator (PPI)* is an indicator that involves measures of the performance of the instances of the executed business processes. PPIs are also related to the goals of the company but involve the measurement of processes used to achieve them.

In order to support business process execution, every BPMS includes the following components, as shown in the classical architecture published in [3]: an execution engine, a process modeling tool, a worklist handler, and an administration and monitoring tools. The *external services* represent external application or services that are involved in the execution of a business process. The *execution engine* is responsible for enacting executable process instances, distributing work to process participants, and retrieving and storing data required for the execution of processes. The *process modeling tool* is the component that allows users to create and modify process models, annotate them with data elements, and store these models in or retrieve them from a *process model repository*. The *worklist handler* is in charge of offering work items to process participants and committing the participants to work lists. Finally, the *administration and monitoring tools* administer and monitor all the operational matters of a BPMS. These components retrieve and store data from two repositories: the process model repository and the execution log repository. The former stores process models, while the latter stores events related to process execution. Note that in the classical architecture, there is no connection between the data of the execution logs and the modeling module. In our approach, the *PIQL engine* is the component in charge of connecting the *administration and monitoring tools* and the *process modeling tools*. As a consequence, the modeler can execute queries to evaluate the status of business process executions. In addition, the *PIQL engine* is connected to *execution engine*, since the measures can be used internally in the execution of other processes, and it is also connected to *external services* since its measures can be used for different purposes, such as to monitor the status of the BPMS in an external dashboard. Figure 1 shows the classical architecture with the PIQL engine included.

Definition 2.9 *Process Instance Query Language (PIQL)* is a domain-specific language (DSL) to query process and task instances in order to obtain measures based on historical process executions.

Process modeling tools can use various notations to represent process models. The de facto standard notation used for modeling business processes is Business Process Model and Notation (BPMN), which supports business process modeling by providing a notation that, on the one hand, is comprehensible to business users, and, on the other hand, represents complex process semantics for technical users [8].

BPMN is not well-suited for modeling the decision logic since decisions are often intermingled with the control flow of process models. The Object Management Group (OMG) proposed DMN to decouple decision specifications from process models. The goal of DMN is to provide a common notation that is readily understandable by all business users in order to bridge the gap between the business decision design and its implementation [9]. In other words, DMN can be seen as a

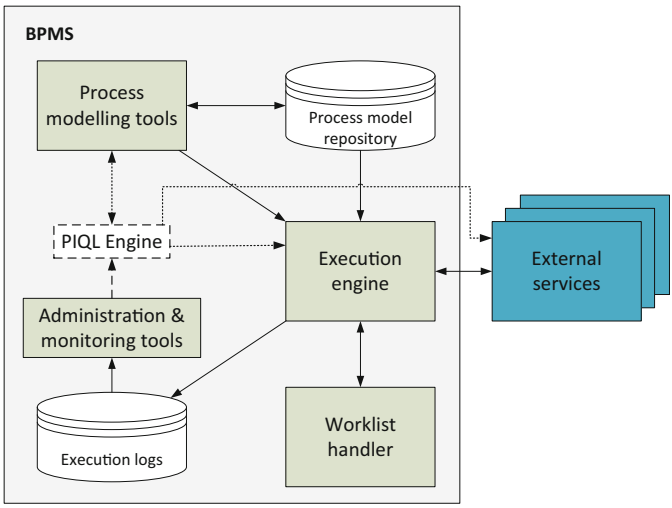

Fig. 1 Architecture of a BPMS that includes a PIQL engine

tool that allows business users to describe repeatable decisions within organizations. DMN provides two levels of modeling: the decision requirement level and the logic level. The former is modeled by means of a Decision Requirement Diagram (DRD), which shows how the decision is structured and what data is needed to make the decision. The latter is modeled using decision tables, which is the standard way of modeling complex business rules.

Definition 2.10 A *decision table* is a visual representation of a specification of actions to be performed depending on certain conditions. A decision table consists of a set of rules that specifies causes (business rule conditions) and effects (business rule actions and expected results); it specifies which inputs lead to which outputs.

A decision table is represented by means of a table, where one column represents one condition. For example, Table 1 shows a decision table that specifies which assembly station should be used depending on the kind of available pieces and the availability of the stations.

In Table 1, decision #1 states that if there are two or more #1657 pieces, five or more #6472 pieces, and at least one #2471 piece, and only Station 6 is available, then the component should be assembled in Station 6. Another example is Decision #3 that results in the assembly of the component in Station 15. The letter "F" marked with an "*" means "First" [9], and the decision engine will evaluate the rules in the proposed order and stop once it has found a rule that applies.

Table 1 Example DMN table for selecting a task based on the availability of resources

F*	Input						Output
#ID	Nº of pieces code #1657 (int)	Nº. of pieces code #6472 (int)	Nº of pieces code #2471 (int)	Is Station 6 available? (Boolean)	Is Station 12 available? (Boolean)	Is Station 15 available? (Boolean)	Station (string)
#1	>= 2	>= 5	>= 1	True	False	False	Station 6
#2	>= 1	>= 2	>= 6	False	True	False	Station 12
#3	>= 7	Any	>= 1	False	False	True	Station 15
…	…	…	…	…	…	…	…

3 Motivating Scenario

In order to show the applicability of PIQL in a real-world context, we introduce a motivating scenario related to the assembly process in a factory, similar to the ones used in the automobile and aerospace industries. A factory produces a set of *Components* in a production line. The production line is composed of a set of actions defined as the *Line of Pieces*. Each *Line of Piece* is executed in a factory *Station* and combines a set of *Pieces*. Figure 2 depicts a conceptual model of this scenario using the Unified Modeling Language (UML) [10]. The conceptual model includes the elements that are the most relevant elements to the assembly process.

The decision regarding which *Components* to assemble depends on the customers' requirements and the availability of the *Pieces* and *Stations* at the time. A specific *Piece* can be used in various *Components*. As a consequence, when a *Piece* is available, different assembly processes could be set up at that moment. Moreover, the same *Station* can be used in different assembly processes, and therefore the manager has to control the availability of the *Station* in each case. The management of *Stations* and *Pieces* involves a set of crucial and risky decisions for the company.

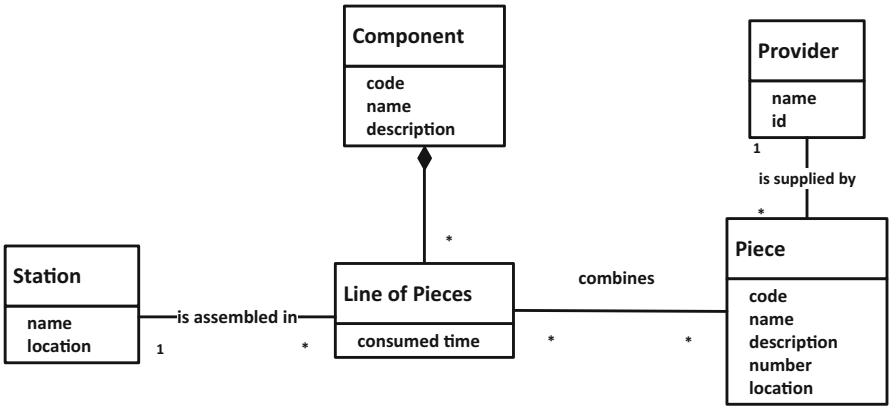

Fig. 2 Conceptual model of assembly process in UML notation [10]

Considering that there is a good amount of stock and that there are always *Pieces* available, then the main problem is in making decisions about the *Stations*. These decisions depend on all the running process instances, that is, *Components* that are being assembled in parallel, *Components* that are waiting for the assembly to start, or *Components* whose assembly is finishing.

Figure 3 shows two business processes related to the assembly of two different types of components. These processes model the sequence of stations visited by a component for its assembly. Although in both processes the components have to pass through a number of stations, the order and requirements vary. On the one hand, a *"Type A Component"* assembly must always pass through the three stations 6, 12, and 15, although the order does not matter (hence the process model does not require to visit all the three stations). On the other hand, a *"Type B Component"* assembly varies according to the requirements of the component at any given time; it may visit both stations, 12 and 15, or only visit one of the two. Finally, validation of *"Type A Component"* is automated by checking if the component has visited all stations, whereas validation of *"Type B Component"* requires expert supervision. The decisions that route the execution from one station to the other can be seen in Table 1.

Querying data related to the status of the company, such as the number of instances executed or availability of stations, is crucial in different contexts. The following subsections introduce some of the contexts in which these queries are essential.

Context 1: Dashboards

Business experts monitor and manage the evolution of the company's business processes, commonly by means of a dashboard. A dashboard visualizes several indicators to help experts carry out correct management. For instance, the "increase of the number of Type A Components by 24%" is an example of KPI defined for measuring the goal "increase market share." Meanwhile, "the number of instances successfully executed" and "the instantiation time of the 'Assembly of Type B Component' process" are examples of PPI. In order to obtain these indicators, various measures should be defined: the number of executions of the *"Assembly of Type A Component"* process indicates whether the KPI "increase of the number of Type A Components by 24%" is reached, whereas the number of executions of the *"Assembly of Type B Component"* process that have not been canceled solves the PPI detailed above. To obtain the execution time of an instance, its start and end times must be analyzed. In all these cases, the use of PIQL in obtaining these measures is essential.

Context 2: DMN Tables

Another context in which PIQL is crucial is at decision points. In Fig. 3, both diagrams contain decision tasks, such as the *"Decide the tasks according to the availability"* task in the *"Assembly of Type A Components"* process. This task takes into account the availability of Stations 6, 12, and 15 to decide which station should be used to continue with the assembly of the component. The decision logic related to this task can be modeled with a DMN table. In fact, Table 1 shows an example logic behind the decision of the *"Decide the tasks according to the availability"*

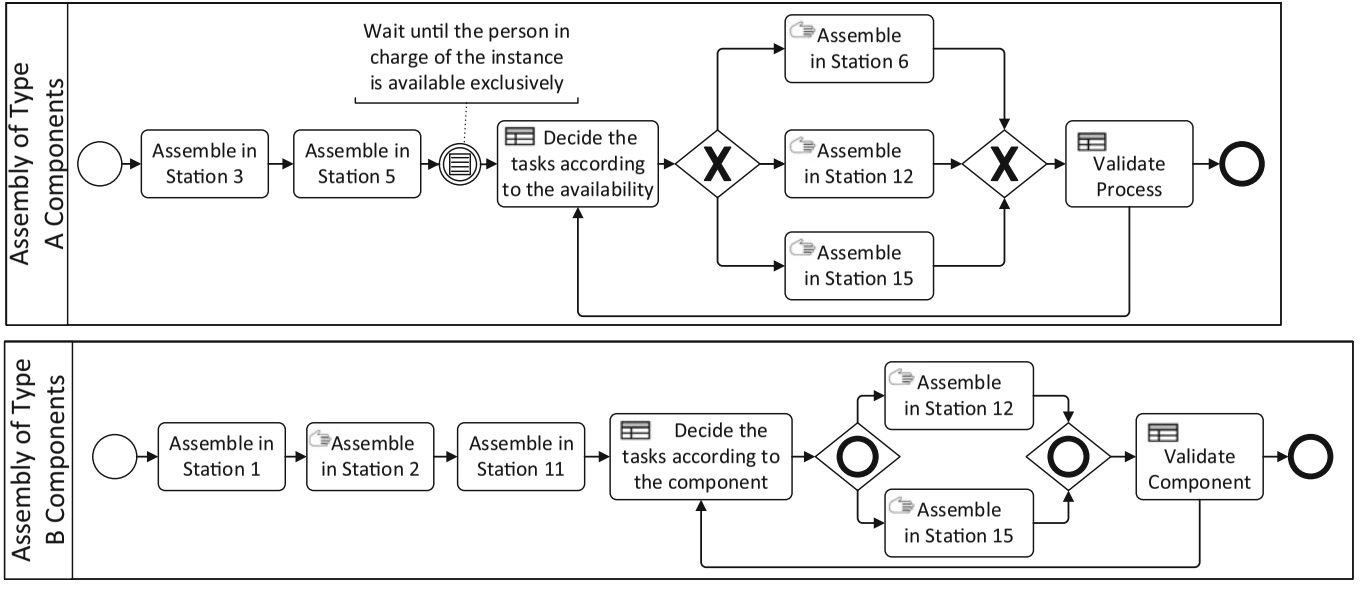

Fig. 3 Business process models for the assembly of two components captured in BPMN [8]

task. Depending on the number of pieces, whose codes are #1657, #6472, and #2471, and on the availability of the stations, the decision varies. The availability of Stations 6, 12, and 15 can be checked by executing a PIQL query since it is necessary to ascertain whether other process instances are using these stations.

Context 3: Dataflow in Business Process Management
Finally, another context where PIQL becomes decisive is in the kind of information that flows through the process, that is, the dataflow. At certain points, the information related to other instances or resources is vital and should be included as part of the dataflow. Following on with the previous example, after the execution of the *"Assemble in Station 5"* task in the *"Assembly of Type A Components"* process, there is a conditional event. The event needs to ascertain whether the person in charge of the process instance is executing some task. If the person is performing another task, then the process should wait until this person is released. This information can be obtained through the execution of a PIQL query.

4 Process Instance Query Language

This section explains the main components of PIQL: syntax, semantics, and notation. Furthermore, as the envisioned users of PIQL are nontechnical people, a set of *Patterns* and *Predicates* is defined in this section to help them write queries in a language that resembles the natural language.

A Process Instance Query Expression (PIQE) (an expression) is used to represent a PIQL query, which is evaluated within the context of processes (P) or tasks (T). The context specifies whether the query recovers information about process instances or about task instances. Note that the result of a PIQE execution is always a measure, that is, a numeric value. For example, Listing 1 shows a PIQE that retrieves information about the number of process instances (note that the expression starts with P to denote the process context). Furthermore, if we look at the PIQE shown in Listing 1, we can find some other features such as (i) every keyword or relation operator is written in uppercase letters and (ii) parentheses could be used to group expressions.

Listing 1 Process instance query expression

```
P (ProcessName IS-EQUAL-TO "Assembly of Type A Components"
  AND (Start IS-GREATER-THAN 2018-01-06)
  AND (End IS-LOWER-OR-EQUAL-TO 2018-03-16));
```

4.1 Syntax

The syntax of PIQL is defined using the Extended Backus-Naur Form (EBNF) [13]. Thus, if *x* and *y* are symbols, *x?* denotes that *x* can appear zero times or once, *x+* denotes that *x* can appear one or more times, *x** denotes that *x* can appear zero or more times, and finally, *x* | *y* represents that either *x* or *y* can appear. In addition, non-terminal symbols are enclosed in < and >, and symbols and keywords are enclosed in single quotes. For the sake of clarity, the definition of the non-terminal symbol <String> has not been included in the grammar shown below, but it should be taken into account that a String is a sequence of characters that starts and ends with double quotes. For instance, `"This is a String"` is a String.

⟨*PIQE*⟩ ::= ⟨*Context*⟩ ⟨*Disjunction*⟩ '`;`'

⟨*Context*⟩ ::= '`P`' | '`T`'

⟨*Disjunction*⟩ ::= ⟨*Conjunction*⟩ ('`OR`' ⟨*Conjunction*⟩)*

⟨*Conjunction*⟩ ::= ⟨*Negation*⟩ ('`AND`' ⟨*Negation*⟩)*

⟨*Negation*⟩ ::= '`NOT`'? ⟨*Comparison*⟩

⟨*Comparison*⟩ ::= ⟨*Addition*⟩ (⟨*ComparisonOperator*⟩ ⟨*Addition*⟩)*

⟨*ComparisonOperator*⟩ ::= '`IS-EQUAL-TO`' | '`IS-NOT-EQUAL-TO`'
| '`IS-LOWER-THAN`' | '`IS-GREATER-THAN`'
| '`IS-LOWER-OR-EQUAL-TO`'
| '`IS-GREATER-OR-EQUAL-TO`'

⟨*Addition*⟩ ::= ⟨*ArithmeticOperand*⟩ (('`PLUS`' | '`MINUS`') ⟨*ArithmeticOperand*⟩)*

⟨*ArithmeticOperand*⟩ ::= '`(`'? ⟨*Operand*⟩ '`)`'? (('`MULTIPLIED-BY`'
|'`DIVIDED-BY`') '`(`'? ⟨*Operand*⟩ '`)`'? '`)`'*

⟨*Operand*⟩ ::= '`(`' ⟨*Disjunction*⟩ '`)`' | ⟨*Property*⟩ | ⟨*Value*⟩ | ⟨*Variable*⟩

⟨*Property*⟩ ::= '`ProcessName`' | '`TaskName`' | '`Start`' | '`End`' | '`Canceled`'
| '`Who`'

⟨*Value*⟩ ::= ⟨*Number*⟩ | ⟨*String*⟩ | ⟨*Date*⟩ | ⟨*Boolean*⟩ | '`NULL`'

⟨*Date*⟩ ::= ⟨*Integer*⟩ '`/`' ⟨*Integer*⟩ '`/`' ⟨*Integer*⟩
| ⟨*Integer*⟩ '`-`' ⟨*Integer*⟩ '`-`' ⟨*Integer*⟩

⟨*Boolean*⟩ ::= '`true`' | '`false`'

⟨*Number*⟩ ::= ⟨*Integer*⟩ | ⟨*Float*⟩

⟨*Float*⟩ ::= ⟨*Integer*⟩? '`.`' ⟨*Digits*⟩

⟨*Integer*⟩ ::= '`-`'?⟨*Digits*⟩

⟨*Digits*⟩ ::= ‘(’‘0’..‘9’‘)’+

⟨*Variable*⟩ ::= ‘$’⟨*String*⟩

4.2 Semantics

The main concepts related to PIQL are detailed below.

Expression. An expression is a combination of one or more values, variables, and operators. Each expression can be seen as one single query, and it should be evaluated within a context: either the process or the task context.

Context. A context specifies whether one wants to retrieve information about *process instances* (P) or *task instances* (T). The context determines the kind of information and attributes that can be retrieved and/or used to define a query.

Process Instance Context. A process instance is described by a tuple

$$< \mathit{CaseId}, \mathit{ProcessName}, \mathit{Start}, \mathit{End}, \mathit{Canceled}, \mathit{Who}, \mathit{ListOfGlobalData} >,$$

where

- *CaseId* is an identifier that describes the process instance in an unequivocal way. It is assigned by the BPMS when the instance is created.
- *ProcessName* is the name of the instantiated process model.
- *Start* is the date and time when the instance started.
- *End* is the date and time when the instance finished. If the instance has not ended, this attribute is set to *null*.
- *Canceled* is the date and time when the instance was canceled. If the instance has not been canceled, this attribute is set to *null*.
- *Who* is the person who has started the execution of the instance. If the instance has been started by the system, this attribute is set to *system*.
- *ListOfGlobalData* is a collection of the process model global variables. The instantiation of these variables can be crucial at decision points.

Task Instance Context. A task instance represents the information related to the execution of a specific task within a process instance. It is described by the tuple:

$$< \mathit{CaseId}, \mathit{TaskId}, \mathit{TaskName}, \mathit{ProcessName}, \mathit{Start}, \mathit{End}, \mathit{Canceled}, \mathit{Who} >,$$

where the elements have the following meaning:

- *CaseId* is the identifier of the process instance of the activity being executed.
- *TaskId* is an identifier that describes the task instance in an unequivocal way. It is assigned by the BPMS when the instance is created.
- *TaskName* is the name of the task.

- *ProcessName* is the name of the process model that contains the task. Note that this property is derived from CaseId (by querying the process whose id is CaseId).
- *Start* is the date and time when the task started. If the task has not started, then this attribute is set to *null*.
- *End* is the date and time when the task finished. If the task has not finished, then this property is set to *null*.
- *Canceled* is the date and time when the execution of the task was canceled. If the task has not been canceled, then this property is set to *null*.
- *Who* is the person who has started the execution of the task instance. If the instance was started by the system, this attribute is set to *system*.

As already stated, the result of a PIQE is a measure, that is, a numeric value. PIQE supports three types of operators to filter instances: *Logical*, *Comparison*, and *Arithmetic*. Operators are modeled using uppercase letters and the hyphens symbol. The operators supported by PIQL, and grouped by types, are described below.

Logical Operators combine two Boolean values. The following logical operators are defined in PIQL:

- *NOT*: logical negation, it reverses the true/false outcome of the expression that immediately follows.
- *OR*: it performs the logic operation of disjunction.
- *AND*: it performs the logic operation of conjunction.

Comparison Operators define comparisons between two entities. These comparison operators are applied to the data types specified in the grammar: Date, Number, Float, Integer, and String. For numeric data types, natural sort order is applied; for Date type, chronological order is applied; and the comparison of String data is performed in alphabetical order. The following comparison operators are defined in PIQL:

- *IS-EQUAL-TO* evaluates whether two elements have the same value.
- *IS-NOT-EQUAL-TO* evaluates whether two elements have different values.
- *IS-GREATER-THAN* evaluates to *true* if the first element of the expression has a greater value than the second. For Dates, this means that the first date is later than the second one.
- *IS-GREATER-OR-EQUAL-TO* returns false when the second element of the expression has a higher value than the first one; otherwise, it returns true.
- *IS-LOWER-THAN* is the inverse of the previous operator. It returns true when the first element of an expression has a lower value than that of the second element; otherwise, it returns false. For Dates, this means that the first date is earlier than the second one.
- *IS-LOWER-OR-EQUAL-TO* is the inverse of *IS-GREATER-THAN* operator.

Table 2 Precedence of PIQL operators

Precedence	Operators	Associativity
5	()	Left to right
4	MULTIPLIED-BY	Left to right
	DIVIDED-BY	
3	PLUS	Left to right
	MINUS	
2	IS-EQUAL-TO	Left to right
	IS-NOT-EQUAL-TO	
	IS-LOWER-THAN	
	IS-GREATER-THAN	
	IS-LOWER-OR-EQUAL-TO	
	IS-GREATER-OR-EQUAL-TO	
1	NOT	Left to right
	OR	
	AND	

Arithmetic Operators are binary operators that define mathematical operations between two entities. These operators are applied to numerical data. The following arithmetic operators are defined in PIQL:

- *PLUS*: it performs the addition of the elements surrounding this operator.
- *MINUS*: in the A MINUS B expression, the MINUS operator performs the subtraction of B from A.
- *MULTIPLIED-BY*: it performs the multiplication of the first and second elements of the expression.
- *DIVIDED-BY*: it performs the division of the first element by the second element.

Table 2 shows the precedence and associativity of the different operators defined in PIQL. The column Precedence holds numbers that specify the precedence of the operator specified in the row. The greater the number is, the higher precedence the operators have.

Variable. A variable can be seen as a placeholder, as it is replaced with a specific value at run-time. A variable can be used to write PIQEs in a more flexible way. The value of a variable needs to be specified by the user at run-time. However, there is also a set of system variables that do not need to be specified, which are:

- $yesterday evaluates to the day before the current date.
- $today or $current_date evaluates to the current date.
- $this_instant evaluates to the current timestamp.
- $tomorrow evaluates to the day after the current date.

Two special symbols used in the PIQL grammar are composed of:

- "$": this operator is used to indicate a variable.
- ";": the semicolon operator marks the end of a PIQE.

4.3 Patterns and Predicates

Since we envision that the main users of PIQL will be nontechnical people, PIQL is enriched with a set of patterns and predicates that help users to write queries in a language that resembles the natural language. The patterns and predicates could easily be defined for several languages. Thus, for example, there could be a set of patterns for English speakers, another set for Spanish speakers, and so on. In this chapter, the set of patterns for English speakers is defined, as shown in Tables 3 and 4. These patterns and predicates can be automatically translated into PIQEs. Therefore, these patterns and predicates are not only the mechanisms that make PIQL more friendly to nontechnical people but also the mechanisms that make the language more flexible, because the patterns can be easily adapted to be closer to the modeler's mother tongue.

A pattern is a mapping between a sentence, or part of a sentence written in the expert's mother tongue, and an element of the PIQL grammar. For example, the pattern "The number of instances of processes" is mapped to the Process Instance Context element, in the case of the English language. This means that a nontechnical user can start a query by writing "The number of instances of processes" instead of just writing "P" to specify the context in which the query must be evaluated. Table 3 shows the PIQL patterns defined for English speakers and their relation to the grammar elements.

A predicate is a pattern that represents a Boolean-valued function written in a natural language. For example, instead of writing "The number of instances of processes whose end date is not equal to Null," the user can write "The number of instances of processes that are not finalized." Predicates are transformed into patterns, and then these patterns are transformed into PIQEs, written according to the grammar introduced in Sect. 4.1. A set of predefined predicates and their mappings to patterns is shown in Table 4.

Finally, note that the use of patterns and predicates is not mandatory. Thus, third-party applications or technical users can also use the *"raw"* language (without patterns and predicates).

Table 3 PIQL patterns for English language

Grammar component	Pattern	
Process instance context	The number of instances of processes	
Task instance context	The number of instances of tasks	
Properties	**Attributes**	**Pattern syntax**
	idCase	*With the case id*
	Process_Name	*With the name*
	Task_Name	*With the name*
	Start	*With the start date and time*
	End	*With the end date and time*
	Canceled	*Canceled*
	Who	*Executed by the user*
ARITHMETIC_OP	**Operator**	**Pattern syntax**
	PLUS	*Plus*
	MINUS	*Minus*
	MULTIPLIED-BY	*Multiplied by*
	DIVIDED-BY	*Divided by*
BOOLEAN_OP	**Operator**	**Pattern syntax**
	NOT	*Not*
	AND	*And*
	OR	*Or*
COMPARISON_OP	**Operator**	**Pattern syntax**
	IS-EQUAL-TO	*Is equal to*
	IS-NOT-EQUAL-TO	*Is not equal to*
	IS-LOWER-THAN	*Is less than*
	IS-GREATER-THAN	*Is greater than*
	IS-LOWER-OR-EQUAL-TO	*Is less than or equal to*
	IS-GREATER-OR-EQUAL-TO	*Is greater than or equal to*

Table 4 PIQL predicates for English language

Predicate	Transformed pattern
That are finalized	End date is not equal to Null
That are not finalized	End date is equal to Null
That are canceled	Canceled is not equal to Null
That are not canceled	Canceled is equal to Null
That are executed by {name}	The user is equal to {name}
With start before {date and time}	A start date is less than {date and time}
With end before {date and time}	An end date is less than {date and time}
With start after {date and time}	A start date is greater than {date and time}
With end after {date and time}	An end date is greater than {date and time}

5 Implementation

In order to validate the approach, an implementation of PIQL has been developed using a set of mature technologies. The core element of the implementation is the PIQL engine, which is in charge of executing queries and returning the results.[1] Note that the PIQL engine can be connected to any system in order to integrate the results of PIQEs in the various contexts in which PIQL may be used: dashboards, DMN tables, and dataflows. Before going into the details of the PIQL engine implementation, it should be mentioned that PIQL provides a dual format of a query: a user format and a machine format, which is less verbose and better processed by machines. The two formats and their relationships are depicted in Fig. 4. Note that the user format is related to the set of patterns and predicates introduced in the previous section, which helps users write queries in a language closer to the natural language.

Figure 4 shows how the PIQL engine works. Firstly, a user writes a query using a language close to English (user format). The PIQL engine then transforms the query into a PIQE by means of a "Grammar preprocessor" (machine format). Note that a PIQE does not contain any patterns or predicates and also that the "Grammar processor" is the component that allows the interaction with third-party applications that may use the machine format. The "PIQL grammar processor" evaluates the PIQE by extracting the information from the BPMS. Finally, the "Platform communication interface" is the component that deals with different BPMS technologies by means of different drivers. Figure 4 shows how the *Camunda*™ driver queries the BPMS using a REST API.

As mentioned previously, PIQL can be used in different contexts: dashboards, DMN tables, and dataflows. Hence, the PIQL engine should be integrated in each context. As an example, Fig. 5 shows the architecture defined to include context information in DMN tables. Since the integration of the PIQL engine in the other contexts is similar to this model, the rest of the section details this example.

The BPMS chosen is that of *Camunda*™ since it is an open-source platform that includes the workflow engine, the DMN evaluator, and the storage of logs for every process, which are required to implement the proposed architecture. *Camunda*™ also includes a set of APIs, which is the mechanism employed to extract the information regarding the processes and tasks needed for the evaluation of PIQEs. The main components depicted in Fig. 5 are:

- *REST Layer*: This component is in charge of managing the communication between *Camunda*™, the DMN evaluator, and the PIQL engine. This component is implemented using a model—view—controller framework and it feeds the DMN evaluator by means of a *REST API* that has been implemented using

[1] The readers may test the PIQL engine at http://estigia.lsi.us.es:8099/piql-tester.

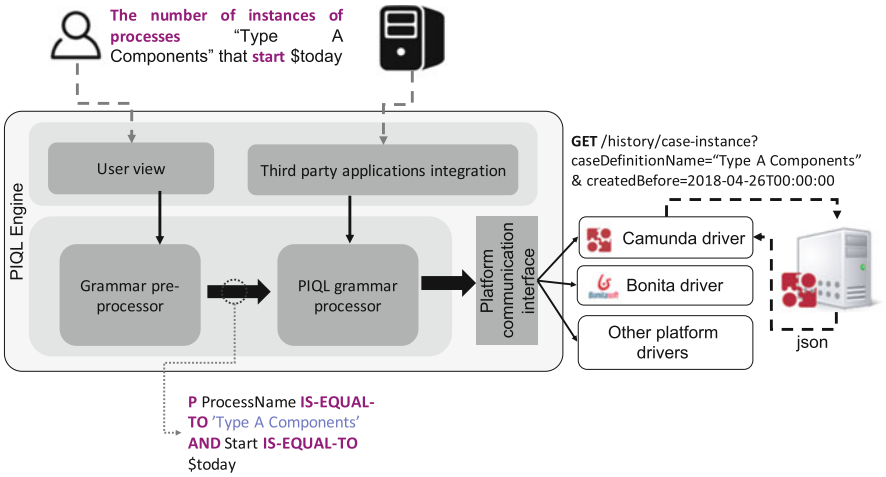

Fig. 4 Query transformation example

Jersey.[2] The data exchanged using this component is in *JSON* format.[3] When a DMN decision has to be evaluated, the BPMS requests the information using the *REST Layer* component. This request is then managed by the "Controller" layer.

- *Controller*: This component receives a request from the *REST Layer* and is in charge of using the *Grammar Preprocessor*, if needed, and later, the *PIQE Grammar Helpers*. Note that if the query is not written in the user-friendly notation, then the preprocessor is not needed.
- *Grammar Preprocessor*: This component handles the mapping of the user-friendly PIQL notation to PIQEs.
- *PIQE Grammar Helper*: This component, together with the *PIQL Engine*, is responsible for resolving the PIQEs. The technology employed to implement this component is *xText*, an open-source framework for the development of textual domain-specific languages.[4]
- *PIQL Engine*: This component analyzes the query by means of the *PIQE Grammar Helpers*. It then calculates the information needed to solve the query and extracts that information from the BPMS. Finally, it returns the value of the PIQE to the controller in order to finalize the request. Note that this component does not have direct communication with the platform (*Camunda*™ in this case) since, to decouple the engine from the platform, a *driver* is introduced. This driver *acts* as an abstract interface to access the real platforms.

[2] https://jersey.github.io/.

[3] https://www.json.org/.

[4] https://www.eclipse.org/Xtext/.

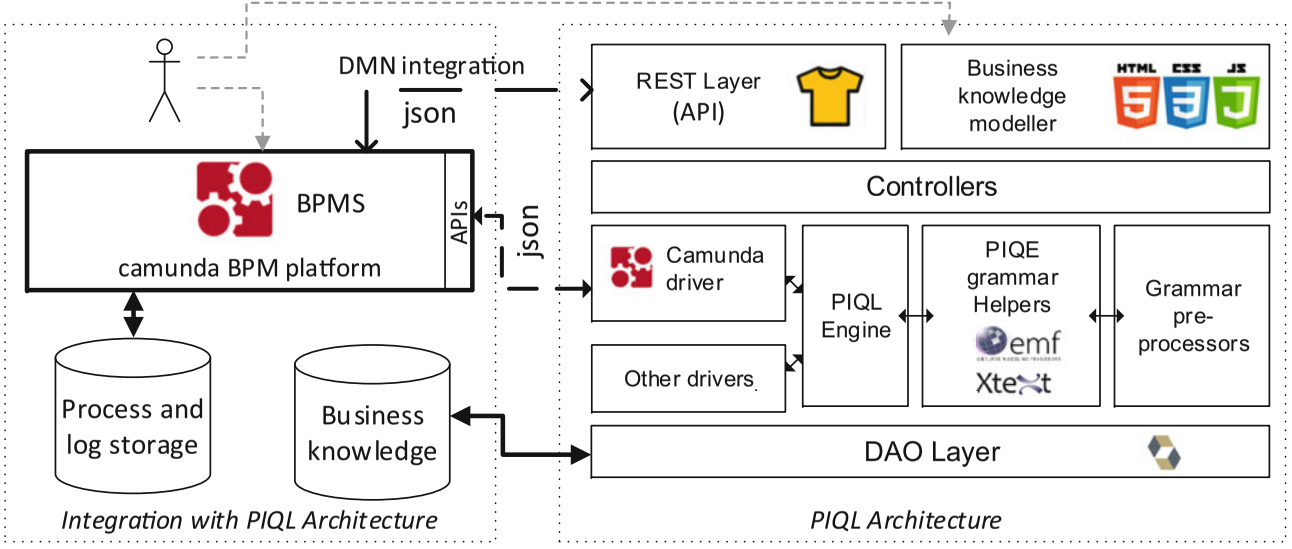

Fig. 5 PIQL architecture

- *Driver*: A driver is responsible for communicating with a real platform and this is the component that knows the specific details of that platform. Note that every system can accept different requests and return different responses. Even worse, there are ad hoc systems which do not provide any API but do provide other ways of retrieving information. This means that one PIQE has to be reformulated, and the reformulation depends completely on the system being used. For example, in some cases, the driver should carry out data processing before returning PIQL data, while in other cases a PIQE needs to be translated into several API requests.
- *Camunda Driver*: In the case of *Camunda*™ the driver uses the *Camunda history service*, which, in turn, uses *Camunda*™ *REST APIs*.
- *DAO Layer*: This component is responsible for storing the business knowledge. *Hibernate* is the technology employed to implement and manage the object-relational mapping.[5]
- *Business Knowledge Modeler*: This component allows users to handle PIQEs and to manage the DMN tables. The implementation of this component has taken advantage of the architecture revealed herein and constitutes a Web application implemented using *HTML, CSS*, and *AngularJS*.

6 Application

The following subsections show examples of queries that are used in the three contexts explained during the chapter. Note that each query is written in the two PIQL notations: the user-friendly format and the machine format.

6.1 Dashboard Enriched with PIQL

A dashboard is composed of a set of measures that enables business experts to easily visualize the state of the company by means of different KPIs and PPIs. An example of KPI in the context of the motivating scenario introduced in Sect. 3 could be "increase in the number of Type A Components by 24%." The following PIQL queries should be executed to obtain the measures that allow the user to verify whether the KPI is reached:

- **Query 1:** The number of process instances with the name *"Assembly of Type A Components"* that start after *2017-12-31* and end before *2019-01-01*.

```
/* PIQE using user-friendly notation */
The number of instances of processes with the name
          'Assembly of Type A Components' that start
    after  2017-12-31 and end before 2019-01-01
```

[5] http://hibernate.org.

```
/* PIQE */
P ProcessName IS-EQUAL-TO 'Assembly of Type A Components
    ' AND Start IS-GREATER-THAN 2017-12-31 AND End
    IS-LOWER-THAN 2019-01-01;
```

- **Query 2:** The number of process instances with the name *"Assembly of Type A Components"* that start after *2018-12-31* and end before today.

```
/* PIQE using user-friendly notation */
The number of instances of processes with the name '
    Assembly of Type A Components' that start after 2018
    -12-31 and end before $today
```

```
/* PIQE */
P ProcessName IS-EQUAL-TO 'Assembly of Type A Components
    ' AND Start IS-GREATER-THAN 2018-12-31 AND End
    IS-LOWER-THAN $today;
```

Note that the comparison of the results obtained from **Query 1** and **Query 2** determines whether the increase of 24% has been reached in 2019; if today is in 2019.

Additionally, an example of a PPI that could be shown in the dashboard is "the number of successfully executed instances of the Assembly of Type B Component process." The corresponding PIQE calculates the number of process instances with the name *"Assembly of Type B Components"* that are not canceled and that ended before today.

```
/* PIQE using user-friendly notation */
The number of instances of processes with the name '
    Assembly of Type B Components' that end before $today
    and are not canceled
```

```
/* PIQE */
P ProcessName IS-EQUAL-TO 'Assembly of Type B Components'
    AND End IS-LOWER-THAN $today AND Canceled IS-EQUAL-TO
    null;
```

6.2 DMN Enriched with PIQL

In the DMN context, PIQL can be applied as an extension of the standard by means of using expressions written in PIQL to define variables. These variables can be included in decision tables. A decision table defines a set of input variables whose values should be taken into account to make the decisions. In our approach, a PIQE

Table 5 Adaptation of DMN Table 1 with PIQL

F*	Input						Output
#ID	Nº of pieces cod. #1657 (integer)	Nº of pieces cod. #6472 (integer)	Nº of pieces cod. #2471 (integer)	*$avSt6* (*integer*)	*$avSt12* (*integer*)	*$avSt15* (*integer*)	Station (string)
#1	>= 2	>= 5	>= 1	0	–	–	Station 6
#2	>= 1	>= 2	>= 6	–	0	–	Station 12
#3	>= 7	Any	>= 1	–	–	0	Station 15
...	...	...	...	...	...	...	...

can be used to calculate the value of the input variable. For example, the DMN table in Sect. 3 (see Table 1) models the requirements that decide which task must be executed in accordance with the availability of pieces and stations in the context of the motivating scenario. Note that to calculate the availability of the different stations, we need to query not only the running instances of the *"Assembly of Type A Component Process"* but also the running instances of all the other processes that use Station 6, 12, or 15. These queries can be executed using the PIQL engine. Table 5 is an adaptation of the DMN, Table 1, that takes the advantages of using PIQL.

The main difference between Tables 1 and 5 is related to the use of PIQL to answer the questions, *"Is Station 6 available?"*, *"Is Station 12 available?"*, and *"Is Station 15 available?"*. In Table 1, the cells in the *"Is Station 6 available?"* column hold Boolean values, while the same cells in Table 5 hold integer values. These integer values are the results of the evaluation of PIQEs whose values are stored in the *"$avSt6"* variable. Equally, the *"Is Station 12 available?"* and *"Is Station 15 available?"* columns are replaced with the values that hold the *"$avSt12"* and *"$avSt15"* variables. Note that the change of the data type (from Boolean to integer) has been carried out because PIQEs always return numeric values. This requirement is not a problem, because the availability of *"Station 6"* can be obtained by counting the number of task instances with the name *"Assemble in Station 6"* and with a null end date. If the result of this PIQE is 0, then the station 6 is available, that is, nobody is using this station. In contrast, if the result of this PIQE is greater than or equal to 1, then the station is not available. The PIQE that enables us to ascertain whether station 6 is available, and whose value is stored in the *"$avSt6"* variable, is formulated as follows:

```
/* PIQE using user-friendly notation */
The number of instances of tasks with the name 'Assemble in
     Station 6' that are not finalized
```

```
/* PIQE */
T TaskName IS-EQUAL-TO 'Assemble in Station 6' AND End
    IS-EQUAL-TO null;
```

Another place in which PIQL can be used is in the context of the *"Validate Process"* task (see the *"Assembly of Type A Components"* process in Fig. 3). This

task checks whether the component satisfies all the requirements to be assembled. For example, one of the main requirements may be that all Type A Components have to pass through the three stations (6, 12, and 15), without a predefined order, to finish the assembly process. Thus, to check this requirement, the corresponding PIQE has to answer the following question: Has a specific process instance already executed the *"Assemble in Station 6", "Assemble in Station 12",* and *"Assemble in Station 15"* tasks? Note that this question makes sense in a DMN scenario in which a decision has to be made. In order to answer this question, three different queries should be evaluated:

1. The number of instances of tasks with a name that is equal to *Assemble in Station 6* and with a case id that is equal to $id
2. The number of instances of tasks with a name that is equal to *Assemble in Station 12* and with a case id that is equal to $id
3. The number of instances of tasks with a name that is equal to *Assemble in Station 15* and with a case id that is equal to $id

After evaluating the queries, the decision task should check that the three results are greater than zero.

```
/* PIQE using user-friendly notation */
/* $Q_St6 */
The number of instances of tasks with the name 'Assemble in
     Station 6' with CaseId is equal to $id
/* $Q_St12 */
The number of instances of tasks with the name 'Assemble in
     Station 12' with CaseId is equal to $id
/* $Q_St15 */
The number of instances of tasks with the name 'Assemble in
     Station 15' with CaseId is equal to $id
```

```
/* PIQE */
/* $Q_St6 */
T TaskName IS-EQUAL-TO 'Assemble in Station 6' AND CaseId
    IS-EQUAL-TO $id;
/* $Q_St12 */
T TaskName IS-EQUAL-TO 'Assemble in Station 12' AND CaseId
    IS-EQUAL-TO $id;
/* $Q_St15 */
T TaskName IS-EQUAL-TO 'Assemble in Station 15' AND CaseId
    IS-EQUAL-TO $id;
```

6.3 Dataflow Enriched with PIQL

Following with the motivating scenario introduced in Sect. 3, the conditional event of the "Assembly of Type A Components" process (see Fig. 3) needs to *determine*

whether the person in charge of the process instance is executing any task. The availability of the person is stored in a variable of the process (whether local or global), and its value can be obtained as a result of evaluating a PIQL expression. Thus, the process uses the value of this variable to verify whether the event should be thrown. The PIQE that is evaluated to calculate the value of the variable mentioned previously should count the number of task instances executed by the user in charge of the process instance that remain unfinished. Thus, if the user in charge of the process is *Lydia Friend*, then the PIQE should be formulated as follows:

```
/* PIQE using user-friendly notation */
The number of instances of tasks executed by 'Lydia Friend'
     that are not finalized
```

```
/* PIQE */
T Who IS-EQUAL-TO 'Lydia Friend' AND End IS-EQUAL-TO null;
```

Remember that PIQEs return numeric values, and as a consequence, verification of whether *Lydia Friend* is performing another task implies determining whether the number of tasks that this person is executing is greater than zero or, in other words, if the PIQE returns a number greater than zero.

7 Framework

Process Querying Framework (PQF) [11] establishes a set of components to be configured to create a process querying method. As a query language, PIQL implements some of these components. This section details these components and how they relate to the framework. In addition, the answers to the decision questions regarding the design that are implemented in PIQL, as suggested by the PQF, are also included.

PIQL enables the extraction of information from process instances and the tasks executed in these instances. During the execution of a specific process, not only the information that flows through this process is crucial for making decisions about the future evolution of the process, but also the information regarding the execution of other processes. This dependency between process executions is due to information and resources shared among them.

Therefore, since PIQL establishes a set of queries on top of the event log data repository, the functional requirements become a set of Create, Read, Update, and Delete (CRUD) operations over this repository. PIQL can read the system logs at run-time to extract the necessary knowledge about past and current process instances. Thus, the BPMS has to record the process and task executions and provide a mechanism to query these executions, while PIQL establishes a set of "read process" queries that isolates the user from technical details.

Figure 6 shows the main components of the PQF that PIQL supports. The *Model and Record* active components denoted by rectangles are essential for

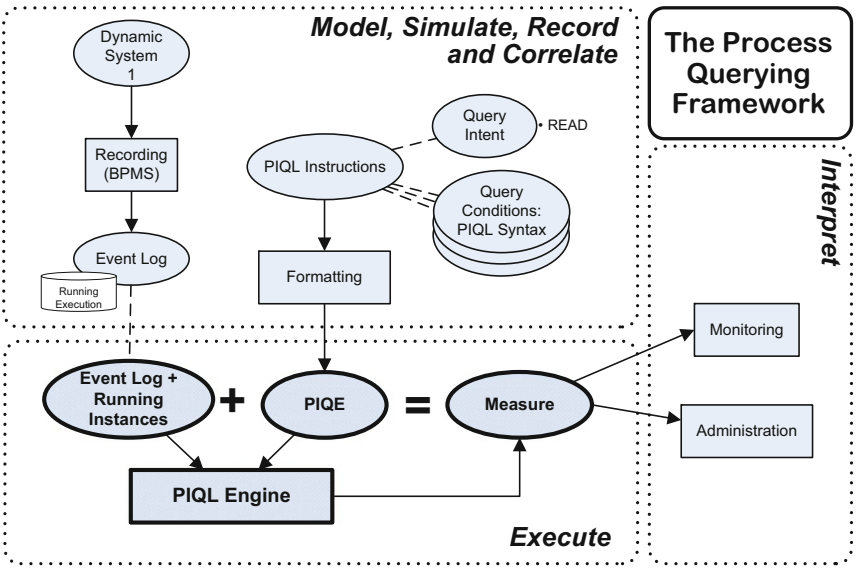

Fig. 6 Instantiation of the process querying framework for PIQL

PIQL. Firstly, a process model is defined in a BPMS, and the event-log repository is acquired automatically through the storage of the information generated by process executions. This storage is carried out by the BPMS itself (see Fig. 1). The majority of BPMSs enable the extraction of information about instances both from past executions and from current executions (although the processes have not finished). Therefore, this event-log repository includes information about finished and unfinished processes. Section 4 defines the set of process querying instructions supported by PIQL. PQF establishes that a Process Query Instruction is composed of a *process query intent* and a set of *process query conditions*. The *process query intent* of PIQL is to read in order to obtain a measurement (a specific quantity value), and the *process query conditions* are its parameters, i.e., the inputs required to execute a specific type of queries.

PIQL delegates the functionality of the "Prepare" component of the framework to BPMSs, which internally have the necessary mechanisms for efficiently querying the stored information and for providing the tools to take advantage of these querying mechanisms. Once both the information stored and the queries are established, the next step is the execution. As explained in Sect. 4, we define a PIQL engine in order to execute the queries. At run-time, the PIQL engine links the queries with the repositories and instantiates the specific values according to the query under execution. The results of the PIQL queries are measurements that are interpretable by users, which means the results may be integrated in a DMN table, visualized in a dashboard, or used to enrich the dataflow, as seen in this Chapter.

Finally, PIQL answers the decision questions regarding the design proposed by the PQF [11] in the following way:

- **DD1. Which behavior models to support?**
  PIQL defines queries over process instances and tasks executed in these instances. Not only is the information generated by completed processes, but also the data generated by running processes is considered to be stored. When certain processes share the same resources and execute the same activities, the information related to the running instances becomes crucial. Therefore, this information is stored and queried using PIQL.
- **DD2. Which processes to support?**
  PIQL establishes queries on top of finite process semantics, where the collection of processes lead to a terminate state. However, since during the execution of a process the information included in the instance is recorded, PIQL can extract the instance information without the need of finishing the process.
- **DD3. Which process queries to support?**
  The *intent* of PIQL is the reading in order to obtain a measurement, that is, a numerical value. The operations in PIQL enable a combination of logic, comparative, and arithmetic operations. PIQL is capable of selecting specific behavior, i.e., process instance from a process repository, and of establishing a measurement.

8 Conclusions and Future Work

This chapter introduces a query language called Process Instance Query Language (PIQL) and the corresponding execution engine. In combination, these enable business experts to extract information from BPMSs. The syntax of the language has been specified using the Extended Backus-Naur Form (EBNF) grammar and its semantics has been specified. In order to validate the approach, various artifacts have been developed using a set of mature technologies: an implementation of the grammar, an engine that can be used to extract information from different platforms, and a driver for the integration of the engine with the *Camunda*™ BPMS platform. The artifacts are part of a modular and extensible platform ready to be integrated with other BPMS platforms.

In order to illustrate the potential of PIQL, a set of PIQL queries has been presented. Furthermore, a real-world example of an assembly business process in a factory has been introduced to demonstrate how PIQL may help nontechnical people extract information about the status of the factory and, as a consequence, improve the decision-making. The examples illustrate the flexibility of PIQL, and how it can contribute to the organizations. Finally, we show how PIQL implements the components of the PQF introduced in [11].

To conclude, future work will deal with (i) the extension of PIQL to enrich the type of filters that can be included in queries with elements such as the use of

resources, execution times, business load, and security aspects, (ii) the inclusion of other data models to be queried, such as business models or business data. Note that currently only business instances can be queried by means of PIQL, and (iii) the improvement of the PIQL engine performance using data caches, indexing, and other similar mechanisms.

Acknowledgments This work was funded by Junta de Andalucia (European Regional Development Fund ERD/FEDER) with the projects COPERNICA (P20_01224) and METAMORFOSIS (FEDER_US-1381375).

References

1. del Río-Ortega, A., Resinas, M., Cabanillas, C., Cortés, A.R.: On the definition and design-time analysis of process performance indicators. Inf. Syst. **38**(4), 470–490 (2013). http://doi.org/10.1016/j.is.2012.11.004
2. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate quality of service computation for composite services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) Service-Oriented Computing, pp. 213–227. Springer, Berlin, Heidelberg (2010)
3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Publishing Company (2013)
4. Gómez-López, M.T., Borrego, D., Gasca, R.M.: Data state description for the migration to activity-centric business process model maintaining legacy databases. In: BIS, pp. 86–97 (2014). http://doi.org/10.1007/978-3-319-06695-0_8
5. Gómez-López, M.T., Gasca, R.M., Pérez-Álvarez, J.M.: Decision-making support for the correctness of input data at runtime in business processes. Int. J. Cooperative Inf. Syst. **23**(4) (2014)
6. González, O., Casallas, R., Deridder, D.: Monitoring and analysis concerns in workflow applications: from conceptual specifications to concrete implementations. Int. J. Cooperative Inf. Syst. **20**(4), 371–404 (2011)
7. Maté, A., Trujillo, J., Mylopoulos, J.: Conceptualizing and specifying key performance indicators in business strategy models. In: Conceptual Modeling - 31st International Conference ER 2012, Florence, Italy, October 15–18, 2012. Proceedings, pp. 282–291 (2012)
8. Object Management Group: Business Process Model and Notation (BPMN) Version 2.0. OMG Standard (2011)
9. Object Management Group: Decision Model and Notation. Reference Manual. OMG Standard (2014)
10. Object Management Group: Unified Modeling Language Reference Manual, Version 2.5. OMG Standard (2015)
11. Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decis. Support Syst. **100**, 41–56 (2017). https://doi.org/10.1016/j.dss.2017.04.011
12. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2012). http://doi.org/10.1007/978-3-642-28616-2
13. Wirth, N.: What can we do about the unnecessary diversity of notation for syntactic definitions? Commun. ACM **20**(11), 822–823 (1977). http://doi.acm.org/10.1145/359863.359883