On the Calculation of Process Performance Indicators ⋆

Antonio Manuel Gutiérrez–Fernández, Manuel Resinas, Adela del–Río–Ortega, Antonio Ruiz–Cortés

School of Computer Engineering
University of Seville
{amgutierrez,resinas,adeladelrio,aruiz}@us.es

Abstract. Performance calculation is a key factor to match corporate goals between different partners in process execution. However, although, a number of standards protocols and languages have recently emerged to support business process services in the industry, there is no standard related to monitoring of performance indicators over processes in these systems. As a consequence, BPMS use propietary languages to define measures and calculate them over process execution. In this paper, we describe two different approaches to compute performance measures on business process decoupled from specific Business Process Management System (BPMS) with an existing BPMS-independent language (PPINOT) to define indicators over business processes. Finally, some optimization techniques are described to increase calculation performance based on computing aggregated measures incrementally.

Keywords: Business Process Management, Key Performance Indicators, Complex Event Processing

1 Introduction

Nowadays, Business Process Management Systems (BPMS) massively support enterprise task flows and human interactions. Tools facilitate different stages in Business Process lifecycle: modelling, deployment, execution or monitoring. Furthermore, standards such as BPMN [6] or UML [9] provide visual languages to define tasks, workflows and decision making points so we can describe and analyze them independently of technology or runtime aspects.

However, certain kinds of problems (such as bottlenecks or resource underutilization) can only be detected by monitoring and measuring process executions and, so far, no standard provides business process monitoring or performance measuring. On the contrary, each BPMS uses proprietary languages to define

⋆ This work has been partially supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2012–32273 (TAPAS), TIC–5906 (THEOS) and COPAS (P12–TIC-1867))

measures and calculate them over business process execution[1,2] . Consequently, defining performance measures is locked to the BPMS.

In this paper, we focus on the definition and development of BPMS independent operations to compute measures over business process executions. To achieve this, we use a set of tools and techniques for the definition and automated analysis of Process Performance Indicators (PPIs) [8], which includes a metamodel and a visual notation. This language has been applied to processes described with BPMN in a number of previous works, although its semantics can be applied to generic workflow languages. Computing measures as non-intrusive methods depends on BPMS architecture and facilities to retrieve information from them. To avoid intrusiveness, this proposal provides two different approaches as contribution: consuming API to get relevant data once and capturing events along the process instances execution. In both perspectives, complex calculations can be computationally expensive, so we introduce incremental techniques to avoid unnecessary processing and performance deterioration. A number of Business Processes (BPs) execution tools, such as Camunda[3] or BIMP[4] are analyzed to apply these approaches and validate our proposal.

In the next section, languages to describe performance metrics over business process are analysed using an example scenario. In section 3, operations on business process and approaches to calculate measures are proposed. Results for the example scenario are presented in Section 4. In the section 5, a computing optimization is introduced. Finally, conclusions and possible work extensions are described in section 6.

2 Defining Performance Indicators

As automation of business processes has increased, a number of standards to define workflows, resources assignment or decision making points have emerged. These standards assist business process analysis independently of the specific system chosen to automate them (or even in the absence of such system). Similarly, defining performance indicators independently of a BPMS requires a language to do it at design time. There already exists a number of languages for describing process-related performance indicators defined in the literature with, in some cases, supporting tools: Pedrinaci et al. [7] present a metric ontology to allow the definition and computation of metrics integrated in SENTINEL, a Semantic Business Process Monitoring Tool; Castellanos et al.'s approach [2] proposes the use of templates provided by a graphical user interface (integrated in the iBOM platform) to define business measures related to process instances, processes, resources or of the overall business operations; Momm et al.'s approach to develop process monitoring systems [5] includes a metamodel for the specification of the performance indicators monitoring and an automated generation of the required

[1] http://wso2.com/products/business-activity-monitor
[2] http://www8.hp.com/us/en/software-solutions/business-process-monitoring
[3] http://camunda.org/
[4] http://qbp-simulator.cloudapp.net/

instrumentation and monitoring infrastructure; Wetzstein et al. introduce in [10] a framework for BAM as part of the semantic business process management, they describe a KPI ontology using WSML to specify KPIs over semantic business processes; Chau et al. [3] propose measuring process goals through capturing process events and defining an ad-hoc algorithm per measure to calculate the goal value.

Nevertheless, they present several issues: its expressiveness in the type of indicators that can be defined is limited [1]; the explicit connection or relationship between the performance indicators and the business process elements is partially or not addressed in them, hindering its computation on existing BPMSs; finally, all of them are coupled to specific platforms or systems.

An approach that overcomes the aforementioned issues is PPINOT[8], which proposes a language to express PPIs at design time, either graphically or by means of a template-based textual notation. However, although PPINOT has tooling support to define PPIs graphically, it lacks an implementation to compute them in process engines. This is why we will use PPINOT as starting point to define our approaches to compute PPIs in a way that is decoupled from specific BPMS. Metric definition in PPINOT is classified into three main categories depending on the number of process instances involved and the nature of the measure: base measures, aggregated measures, and derived measures.

Base measures: They are obtained directly from a single process instance and do not require any other measure to be computed. Aspects that can be measured include: 1) the duration between two time instants (time measures); 2) the number of times something happens (count measures); 3) the fulfillment of certain condition in both running or finished process instances (condition measures); and 4) the value of a certain part of a data object (data measures).

Aggregated measures: Sometimes, it is interesting not only knowing the value of a measure for a single process instance (base measures) but an aggregation of the values corresponding to the multiple instances of a process. For these cases, aggregated measures are used, together with an aggregation function such as average, maximum, etc.

Derived measures: They are defined as functions of other measures. Depending on whether the derivation function is defined over single or multi-instance measures, derived measures are classified accordingly as derived single-instance measures or derived multi-instance measures.

To illustrate the definition of a business process and its performance indicators, we consider a simple business process for registering appointments in public health system. The procedure to book an appointment with doctors starts when the user logs. Figure 1 depicts the described process in BPMN and the related PPIs using PPINOT. Once a user has been identified, the system displays all the available days for his family doctor in the next two weeks (all users in public health system have always a family doctor assigned). When the User **chooses a desirable day**, the system displays available time slots for that day (15 minutes slots). If the user **selects one of these slots** and confirms the decision, the system **registers the appointment and sends a notification to the user** personal inbox (mobile and/or email). If the user disagrees with the available slots, he can choose a different day to see its available slots. There are other activities related, such as choosing a new doctor or modify a previous appointment that are not considered in this example process.

According to Spanish official stats, there are an average of 6 visits/person-/year[5], so this process executes several hundreds of thousands times per day (240 million per year). This number can be even higher, considering the process instances that start but finish without a fixed appointment. Therefore, measuring performance indicators to optimize this process will have an impact on the computational resources.

In this process, the following performance indicators are of interest:

- Average time per appointment process instance (from the user logging to appointment registration)
- Average of day changes to review time slots per doctors

First metric gives basic performance information. Second metric gives hints about bad performance, as reviewing different days indicates the user finds it difficult to match available time slots to his preferences (useless schedule times or assignment between patient and doctors that could be improved).

These two metrics are easy to define over the business process. The average total time for an appointment is measured as the average of the time difference between an appointment registration (end of task "Register and Notify Appointment") and a user logs (end of task "Identify Patient"). Time slot reviews per doctor is measured as the average number of times a user reviews different days per doctor (i.e., task "Choose day" is executed). Both measures are Aggregated Measures of Time Measures in PPINOT.

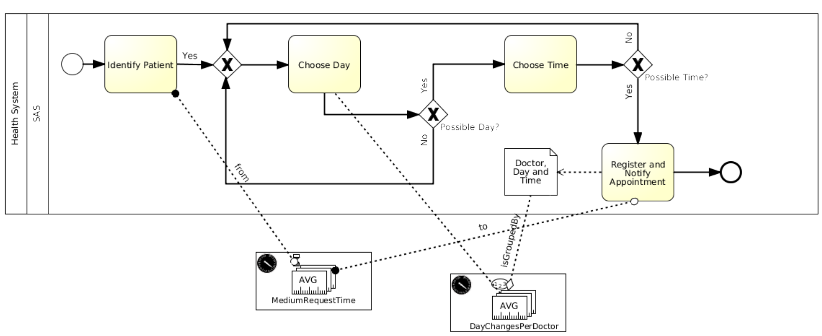

Fig. 1. Appointment Management BPMN + PPI

[5] http://goo.gl/QWtqTn

3 Methods to Retrieve Performance Information

Despite these indicators are simple, there is no standard in BPMSs to calculate them so they require to be interpreted and transformed in terms of the BAM facilities. So, even using a BPMS-independent definition language, such as PPINOT, for defining PPIs over BPMN business processes, indicator computation depends on the specific BPMS chosen for BP execution.

Regardless of the process engine, operation to compute PPIs in a generic interface is defined by:

- List <MeasureValue>PPICompute(PPI currentPPI)

That is, with the PPI properties, as the type of measure or the business process definition and task/s related to it, execution data are collected to get the appropriate values for the measure. For a PPI with base measure -as base measures correspond a single measure value por each process instance-, this operation will return as many values as process instances. For a PPI with aggregated measure, the scope of aggregation determines the aggregation of values from single process instances. In Table 1, we trace example execution data for this process and the expected results for the measures indicated in the previous section (final aggregated values). The result of this operation relies on having access to execution data from the process engine. We have identified two different approaches to get this data:

- Data capture
- Event capture

Once we have the required execution information, the indicator value is trivially processed with mathematical calculation.

Table 1. Execution trace

BPInstance ID	Process Execution Data	Measure Value	
1	Finished in 60 sec	MediumRequestTime	60 sec
	3 day changes, Doctor ”A”	DayChangesPerDoctor	”A”:3
2	Finished in 75 sec	MediumRequestTime	65 sec
	4 day changes, Doctor ”B”	DayChangesPerDoctor	”A”:3, ”B”: 4
3	Finished in 90 sec	MediumRequestTime	70 sec
	5 day changes, Doctor ”A”	DayChangesPerDoctor	”A”:4, ”B”: 4

3.1 Data capture

Process engines usually persist activities in an easy-to-understand database or even provide facilities to query them via an API. Both procedures, database query and API consumption, are depicted in Figure 2. PPICompute method

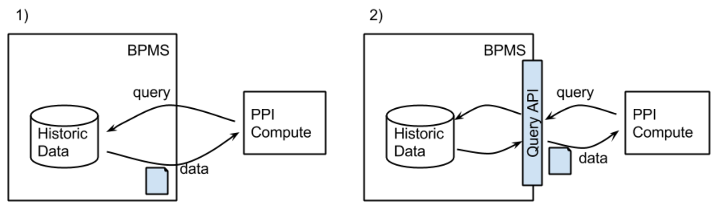

Fig. 2. Data capture direct from Database 1) or using a provided API 2)

directly queries required task information from persistance system (Diagram 1) or consumes API provided by BPMS (Diagram 2).

In this approach, the data capture provides at once all the required data for activities related to an indicator. Once the data are retrieved, some measures require post-processing these data. For example, in average time per process indicator, after retrieving timestamps from initial and final tasks, we have to get the difference between them for each process instance and then compute the average value for all the instances. Depending on the persistence or API components deployment, data querying may impact on process engine performance.

3.2 Event capture

Although data-capture based models are easy to design and implement, some process engines are not suitable for this approach. For instance, if we neither can access to persistence model nor to any data query API. In this case, we propose to capture relevant events[6] in process execution and use them to compute indicators. This approach requires extending BPMS listeners mechanisms (if they exist) or modifying the BPMN model (i.e.: observer pattern) to generate the required events so all events that occur during process execution are sent as a stream of events to PPICompute using a method such as (depicted in Figure 3):

```
void processEvent(RuntimeEvent event)
```

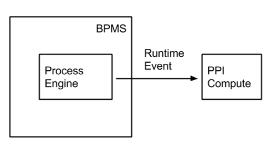

Fig. 3. Event capture

[6] when we talk about events here, we refer to any stage of interest in process flow, not BPMN events

RuntimeEvent object usually includes information about process instance identification, timestamp and event trigger (task starting, data change, ...), so we can check if it is relevant for the PPI and compute it (in positive case). Once we get the information, we always require event data post-processing. Although events are simple to process, a continuos capture has an impact on the communication channel, commonly internet and http.

4 Prototype Implementation

Following the proposed approaches, we have developed indicator calculation components in different business process tools. The method to choice depends on BPMS capabilities or performance criteria, as both approaches are similar in terms of expressiveness. Camunda is an open source BPMN system forked from Activiti which features a REST API to get history execution so we applied data capture approach. On the other hand, BIMP is a business process simulator with no data persistence so we use event capture approach. Both approaches are

4.1 Calculating indicators in Camunda

Camunda interface offers methods to retrieve any activity (task, gateway or event) information so we can easily get timestamps for time indicators or account tasks executions with the Camunda History Service. The implementation of timestamp capture for time measure in Camunda is illustrated in code 1.1, where camundaHistory is the Camunda History Service object.

Listing 1.1. Data processing

```
List<PPIValue> computeMeasure(PPIValue ppi){
  //startTimes is a Map<String,Long>
  String pName                  = ppi.getProcessName();
  TimeInstantCondition startTime = ppi.getFrom();
  String startTask  = startTime.getTask();
  State  startState = startTime.getChangesToState();
  HistoricProcessInstanceQuery processQuery       =
        camundaHistory.createHistoricProcessInstanceQuery();
  HistoricActivityInstanceQuery activityQuery     =
        camundaHistory.createHistoricActivityInstanceQuery();
  //Getting and Iterating Process Instances
  List<HistoricProcessInstance> processInstances  =
        processQuery.processDefinitionName(pName).list();
  for (HistoricProcessInstance pInstance:processInstances){
    String instanceId = pInstance.getId();
    activityQuery = activityQuery.processInstanceId(instanceId);
    //Retrieving Measure Start execution data
    HistoricActivityInstance startActivity =
        activityQuery.activityId(startTask).singleResult();
    if (startState.equals(GenericState.START)){
      startTimes.add(instanceId, startActivity.getStartTime());
    }else{
      startTimes.add(instanceId, startActivity.getEndTime());
    }
    //Similar for Time End Condition in Measure
    //After getting data, we calculate Average ...
  }
}
```

4.2 Calculating indicators in BIMP

As BIMP does not provide any facility to retrieve information, the event capture approach is used. So we capture and filter the BIMP stream of activities for measures of interest. The processing of an event for time capture follows the algorithm depicted in code 1.2.

Listing 1.2. Event processing

```
//startTimes, endTimes are Map<String,Long>
void process(RuntimeEvent entry) {
        if (entry.matches(Timer.START)) {
            startTimes.add(entry);
        } else {
            endTimes.add(entry);
        }
        if (startTimes.size() == endTimes().size()){
            //updateAverage
            average = getAverageTime();
        }
}
```

5 Optimizing measure processing

In both approaches, complex indicators in massive process instances scenarios can degrade performance. In the query method, data-capture measures, such as aggregations, require computationally expensive views (groups by), so real-time monitoring can compromise process engine enforcement if the database that stores history data is the same that stores runtime information. And, in both methods, processing a number of calculations have an impact on performance. Nowadays, improving big data processing is a research goal so a number of techniques and tools have been developed focusing on this topic. We take advantage of these techniques to extend our proposal[4].

On this regard, we use incremental calculation technique. This technique is implemented by a number of tools to provide scalable processing in big data scenarios (such as Apache DataFu over Apache Hadoop). For the introduced example, to calculate average time on 30 days window, first result requires processing full data for 30 days. However, if we store intermediate daily times, future 30 days windows processing can be made faster if we consider only the different day intervals. So, after computing the first 30 days window, calculating average for that window on the 31st day only requires adding data for the new day and removing the data from the 1st day (it is out of the new window). Incremental calculation is described below.

Let InstDay_i the number of process instances in a day i and TimeDay_i the accumulated process Time for all the process instance in a day i (from "Identify User" Task to "Register Appoinment" Task):

$$InstDay_i = N \tag{1}$$

Total Process Time per Day:

$$TimeDay_i = \sum_{j=0}^{N} ProcessInstanceTime_j \tag{2}$$

Accumulated Process Instances in the last 30 days:

$$AccumInstDay_i = \sum_{j=i-30}^{j} InstDay_i \tag{3}$$

Accumulated Process Time in the last 30 days:

$$AccumTimeDay_i = \sum_{j=i-30}^{i} TimeDay_j \tag{4}$$

Average Process Time in the last 30 days:

$$AverageTimeDay_i = \frac{AccumTimeDay_i}{AccumInstDay_i} \tag{5}$$

If we store intermediate accumulated and daily number of instances, and total process time, we incrementally calculate average for the following day:

$$AccumInstDay_{i+1} = AccumInstDay_i - InstDay_{i-30} + InstDay_{i+1} \tag{6}$$

$$AccumTimeDay_{i+1} = AccumTimeDay_i - TimeDay_{i-30} + TimeDay_{i+1} \tag{7}$$

$$AverageTimeDay_{i+1} = \frac{AccumTimeDay_{i+1}}{AccumInstDay_{i+1}} \tag{8}$$

This process is depicted in Figure 4, where a 3-day window time is computed for the first 3 days and intermediate results are stored (Intermediate Stage 1). In our example, these intermediate data are accumulated process time and accumulated number of instances. On the fourth day, instead of computing full measures for day 2, 3 and 4, we reuse previous results to extract data from day 1 and add day 4 execution data and get the new valid result. So, measure calculation leverages this technique to simplify queries, and avoid unnecessary data processing.

In the measure for average day changes per doctor, incremental calculation can also be applied in simpler form since there is no need to use sliding time window (and subtracting old data) but just adding new daily information to calculation.

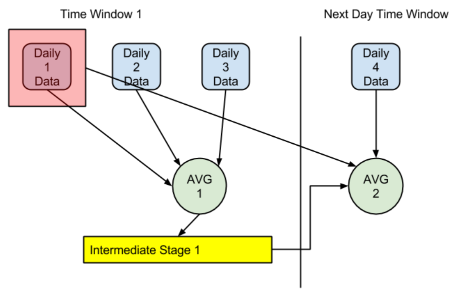

Fig. 4. Time Window calculation

6 Conclusions and Future Work

In this paper, we deal with BPMS-independent computation of process performance indicators. Specifically, we identify two mechanisms to implement this calculation over different BPMSs depending on the feasibility of data retrieving from them. Although Camunda and BIMP are analysed to succesfully implement these approaches, other BPMSs similar integration features so this proposal can easily be applied to them. As heavy processing of indicators can affect BPMS performance, we also provide mechanisms to increase efficiency. This proposal enables the run-time evaluation in different BPMSs of performance indicators defined in a BPMS-independent manner at design time. However, this work is applied to a synthetic scenario. A line of future work is to apply this proposal to real business process executions. Furthermore, it can also be extended to other BPMSs to be validated.

References

1. del-Río-Ortega et al., A.: On the Definition and Design-time Analysis of Process Performance Indicators. Inf. Syst. 38(4), 470–490 (2013)
2. Castellanos et al, M.: ibom: a platform for intelligent business operation management. In: Proc. of the 21st Int. Conf. on Data Engineering. pp. 1084– 1095. Hewlett-Packard Laboratories (2005)
3. Chau, T., Muthusamy, V., Jacobsen, H.a., Litani, E., Chan, A., Coulthard, P.: Automating SLA Modeling. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. pp. 10:126–10:143 (2008)
4. Hayes, M., Shah, S.: Hourglass: A library for incremental processing on hadoop. In: Big Data, 2013 IEEE International Conference on. pp. 742–752 (Oct 2013)
5. Momm et al., C.: Towards a model-driven development of monitored processes. In: Proc. Tagung Wirtschaftsinformatik 2007. pp. 319–336 (2007)
6. Object Management Group (OMG): Business process model and notation (BPMN) version 2.0 (Jan 2011), available from: http://www.omg.org/spec/BPMN/2.0/PDF

7. Pedrinaci et al., C.: Sentinel: a semantic business process monitoring tool. In: Int. Workshop on Ontology-Supported Business Intelligence. pp. 26–30 (2008)
8. del Río-Ortega, A.: On the Definition and Analysis of Process Performance Indicators. Ph.D. thesis, University of Seville (2012)
9. Sinogas, P., Vasconcelos, A., Caetano, A., Neves, J., Mendes, R., Tribolet, J.M.: Business processes extensions to UML profile for business modeling. In: ICEIS (2). pp. 673–678 (2001)
10. Wetzstein, B., Ma, Z., Leymann, F.: Towards measuring key performance indicators of semantic business processes. Bus. Inf. Syst. 7, 227–238 (2008)