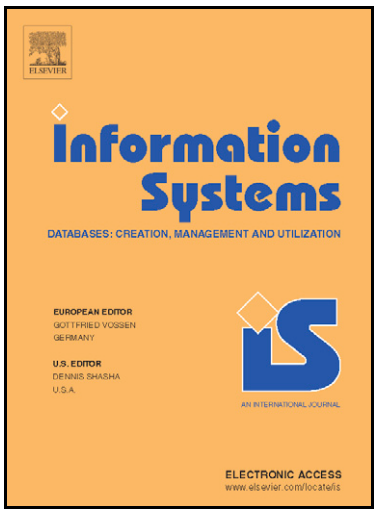

www.elsevier.com/locate/infosys

On the definition and design-time analysis of process performance indicators

Adela del-Río-Ortega, Manuel Resinas, Cristina Cabanillas, Antonio Ruiz-Cortés

PII: S0306-4379(12)00146-9
DOI: http://dx.doi.org/10.1016/j.is.2012.11.004
Reference: IS832

To appear in: *Information Systems*

Received date: 31 July 2012
Revised date: 6 November 2012
Accepted date: 8 November 2012

Cite this article as: Adela del-Río-Ortega, Manuel Resinas, Cristina Cabanillas and Antonio Ruiz-Cortés, On the definition and design-time analysis of process performance indicators, *Information Systems,* http://dx.doi.org/10.1016/j.is.2012.11.004

On the Definition and Design-Time Analysis of Process Performance Indicators☆

Adela del-Río-Ortega[a,*], Manuel Resinas[a], Cristina Cabanillas[a], Antonio Ruiz-Cortés[a]

[a]*Dpto. de Lenguajes y Sistemas Informáticos, University of Seville, Av. Reina Mercedes s/n, 41012 Seville, Spain*

Abstract

A key aspect in any process-oriented organisation is the evaluation of process performance for the achievement of its strategic and operational goals. Process Performance Indicators (PPIs) are a key asset to carry out this evaluation, and, therefore, having an appropriate definition of these PPIs is crucial. After a careful review of the literature related and a study of the current picture in different real organisations, we conclude that there not exists any proposal that allows to define PPIs in a way that is unambiguous and highly expressive, understandable by technical and non-technical users and traceable with the business process (BP). In addition, like other activities carried out during the BP lifecycle, the management of PPIs is considered time-consuming and error-prone. Therefore, providing an automated support for them is very appealing from a practical point of view.

In this paper, we propose the PPINOT metamodel, which allows such an advanced definition of PPIs and is independent of the language used to model the business process. Furthermore, we provide an automatic semantic mapping from the metamodel to Description Logics (DLs) that allows the implementation of design-time analysis operations in such a way that DL

☆This work has been partially supported by the European Commission (FEDER), Spanish Government under the CICYT projects SETI (TIN2009-07366) and TAPAS (TIN2012-32273); and projects THEOS (TIC-5906) and ISABEL (P07-TIC-2533) funded by the Andalusian local Government.

*Corresponding author. Tel +34954557001

Email addresses: adeladelrio@us.es (Adela del-Río-Ortega), resinas@us.es (Manuel Resinas), cristinacabanillas@us.es (Cristina Cabanillas), aruiz@us.es (Antonio Ruiz-Cortés)

reasoners' facilities can be leveraged. These operations provide information that can assist process analysts in the definition and instrumentation of PPIs. Finally, to validate the usefulness of our proposal, we have used the PPINOT metamodel at the core of a software tool called the PPINOT tool suite and we have applied it in several real scenarios.

Keywords: Business Process Management, Process Performance Measurement, Process Performance Indicators, Automated Analysis

1. Introduction

Business Process Management (BPM) intends to support business processes using methods, techniques, and software to design, enact, control and analyse operational processes involving humans, organisations, applications, documents and other sources of information [1]. There exists a growing interest in business processes (BPs) for both, academia and business. Many companies are taking this process-oriented perspective in their business, as a way of identifying which steps really create value, who is involved in the process and which is the exchanged information; ultimately, finding out how to improve, where to increase quality, reduce waste or save time [2].

To achieve this improvement of processes, it is important to evaluate their performance, since it helps organisation to define and measure progress towards their goals. Performance requirements on business processes can be specified by means of Process Performance Indicators (PPIs), a particular case of KPIs. PPIs can be defined as quantifiable metrics that allow to evaluate the efficiency and effectiveness of business processes. They can be measured directly by data that is generated within the process flow and are aimed at the process controlling and continuous optimization [3].

According to Franceschini et al. [4] and based on the conclusions drawn in our previous works [5, 6], four requirements for the definition of PPIs can be established: (1) expressiveness, the definition should be unambiguous and complete; (2) understandability, PPIs should be understood and accepted by process managers and employees; (3) traceability with the business process, enabling to maintain coherence between both assets, BP models and PPIs; and (4) possibility to be automatically analysed, allowing thus not only to gather the information required to calculate PPI values, but also to infer knowledge to answer questions like *what are the business process elements related to PPI P?*.

Unfortunately, in practice, PPIs are usually defined either in an informal and ad-hoc way, usually in natural language, with its well-known problems of ambiguity, lack of coherence/traceability with the process, incompleteness (missing information), and not amenable to automated analysis; or they are defined from an implementation perspective, at a very low level, and too close to the technical view, becoming thus hardly understandable to managers and users and losing the perspective of the higher-level process. Furthermore, there is a significative lack of support of standard business process notations such as BPMN [7] or BPEL [8] to define such PPIs.

From an academic point of view, There already exists a number of research proposals to define PPIs, but we argue that none of them cover all of the four critical elements for indicators because of the following reasons. First, they do not allow a complete definition of PPIs due to their limited expressiveness. Most proposals allow the definition of PPIs related to control flow and time. However, only Wetzstein et al. [9] allow the definition of PPIs related to the state of the process and none of them can define PPIs related to data. They also have limited expressiveness regarding the definition of PPIs based on aggregated or derived measures and in the definition of the period of analysis of the PPIs. A second issue of current proposals is that in several of these proposals there is just a partial traceability between PPIs and BP models. This may cause maintenance problems to keep the coherence between both assets and it is also an important limitation to instrument the information systems of the organisation to take the measures, specially when these information systems are BPMS (Business Process Management Systems). Finally, the automated analysis of the PPIs that these proposals allow is almost inexistent. Only Popova et al. [10] provide mechanisms to analyse the relationships between PPIs at design-time.

From this discussion we conclude that a definition of PPIs that fulfills the aforementioned requirements is still an unresolved challenge. In this paper we address this challenge by presenting the PPINOT metamodel of which a preliminary version was introduced at [5]. Furthermore, we also introduce a design-time analysis of the relationships between PPIs and BP elements to automatically find out the business process elements that may have an influence on a PPI and the BP elements that must be measured to calculate the PPIs. The main benefits of our contribution can be summarised as follows:

1. The PPINOT metamodel allows an unambiguous and complete defini-

tion of PPIs. It allows the definition of all of the PPIs found in the literature review and addresses all of the aforementioned limitations. Furthermore, its expressiveness has been tested in several real scenarios so that it can provide a solid basis for defining PPIs in any organisation.

2. Since the PPINOT metamodel is at the same level of abstraction than business processes rather than at a more technical level such as information systems logs, it is easier to understand by process managers and employees. In fact, the PPINOT metamodel is the foundation of a graphical notation that extends BPMN to define PPIs (`http://www.isa.us.es/ppinot/`) and a set of templates and linguistic patterns [11] designed to make PPIs easier to understand by non-technical people.
3. A number of design-time analysis capabilities have been defined for the PPINOT metamodel in terms of a set of analysis operations that extract information from the relationships between PPIs and BP elements. Furthermore, these analysis operations are automated by mapping the PPINOT metamodel into a DL knowledge base and then using standard DL reasoning operations to analyse it instead of implementing ad-hoc analysis algorithms. This mapping can also be used to implement other new analysis operations such as the dependencies between PPIs as outlined in [5].
4. A PPI defined using the PPINOT metamodel can always be traced back to the BP elements used in its definition. This traceability provides several benefits such as the possibility to extract information from the relationships between PPIs and BP elements by means of the aforementioned analysis operations, or the implementation of a tool that automatically instruments an open source BPMS to calculate the values of PPIs.
5. The PPINOT metamodel is independent of the language used to model the business process. In fact, a binding to use it with BPMN is described in this paper.

To validate the usefulness of the PPINOT metamodel, we have developed a software tool called the PPINOT tool suite that uses the PPINOT metamodel at its core to offer: (1) a graphical editor, which allows to model PPIs over Business Process Diagrams (BPDs) using PPINOT graphical notation; (2) a templates editor, which allows a textual definition of PPIs using templates and linguistic patterns; (3) an analyser, which implements the

aforementioned design-time analysis operations, and (4) an instrumenter and reporter, which allows to extract the information required to calculate the values of the defined PPIs during business process execution in the BPMS Activiti, and present a report with such values. Furthermore, we have applied our proposal in several real scenarios, namely: the IT Department of the Andalusian Health Service, the Information and Communication Service of the University of Seville, and the Consejería de Justicia y Administración Pública of the Andalusian Local Government.

The remainder of this paper is organised as follows. In Section 2 we introduce the main concepts related to the definition and management of PPIs. In Section 3 we propose PPINOT metamodel for the definition of PPIs, that serves as a basis for the automated analysis presented in Section 4. Section 5 describes the formalisation of our proposal using Description Logic and the implementation of the aforementioned automated analysis. Some implementation details are given in Section 6. Section 7 presents the application of our approach to a real scenario. In Section 8, we describe the actions conducted to validate our proposal. The related work is presented in Section 9. Finally, Section 10 draws the conclusions from our work, summarizes the paper and outlines our future work.

2. Definition and Management of PPIs

The management of PPIs can be divided into several steps.

To illustrate them, we present an excerpt of a real scenario that takes place in the context of the IT Department of the Andalusian Health Service. We focus on the business process of managing Request for Changes in the existing Information Systems. This process was modelled by the quality office of this department using BPMN, but for the sake of understandability, we have simplified the real process obtaining the diagram depicted in Figure 1.

The process starts when the requester submits a Request For Change (RFC). Then, the planning and quality manager must identify the priority and analyse the request in order to make a decision. According to several factors like the availability of resources, the requirements requested, and others, the RFC will be either approved, cancelled, or raised to a committee for them to make the decision.

The RFC is a data object with its states (cf. Figure 1) and some properties defined, from which we highlight the following ones, that will be referred to throughout the paper: *Project*, which refers to the project to which such

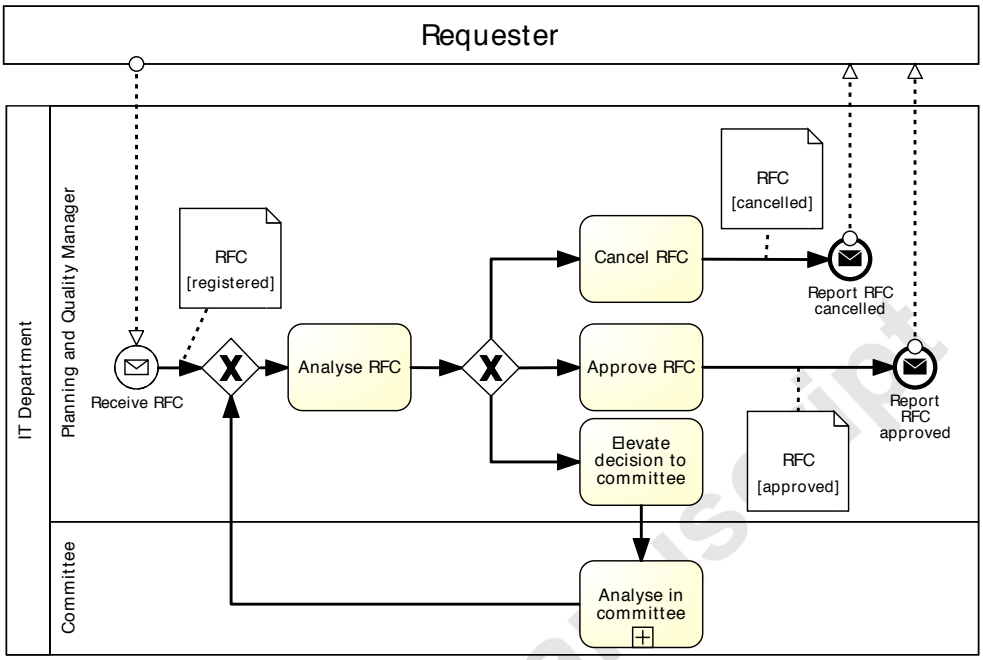

Figure 1: Process of the request for change management (simplified)

RFC is associated, *InformationSystem*, which defines the information system for which a change is required or to which that change affects, *type of change*, which classifies the change required in *adaptive*, *corrective* or *perfective*, and *priority*, which refers to the importance of resolving that RFC.

After modelling the process, the first step is to define the PPIs that are necessary to evaluate it. In our example, the IT department defined the indicators using natural language and collected them in tables. Again, for the sake of simplicity, we only show an excerpt of this table (Table 1). Target values reflected in this table have been changed due to privacy reasons. The responsible for all these PPIs defined in Table 1 is the "planning and quality manager."

Once PPIs are defined, the next step is to decide how to calculate them from the data that is generated during the execution of the process. This involves two actions: first, it is necessary to decide which measures must be taken during the execution of the process to calculate the PPIs. For instance, measures such as the moment when a RFC is received, the moment when the committee starts and ends its analysis, or the number of RFCs received

Name	Description	Target value	Scope
PPI1	RFCs cancelled from RFCs registered	4%	weekly
PPI2	Average time of committee decision	1 working day	weekly
PPI3	Corrective RFCs from approved RFCs	2%	weekly
PPI4	Perfective and adaptive RFCs from approved RFCs	4%	weekly
PPI5	Average time of RFC analysis	2 working days	weekly
PPI6	Number of RFCs under analysis	2 RFCs	weekly
PPI7	Number of RFCs per type of change	20 corrective RFCs 30 adaptive RFCs 20 perfective RFCs	monthly
PPI8	Number of RFCs per project	50 for project rr.hh 60 for project diraya 1 for project pharma	monthly
PPI9	Average lifetime of a RFC	3 working days	monthly

Table 1: PPIs defined for the RFC management process

in a period of time should be taken in the Request for Change Management to calculate PPI5, PPI2 and PPI7, respectively. The second action is to decide how to obtain the measures that are necessary to calculate the PPIs. This usually requires the instrumentation of the information systems that support the business process so that it provides such measures. For instance, configuring the event log of the information system appropriately or using an API of the information system to query its history.

After deciding how measures are obtained, they must be actually gathered during the execution of the process so that the values of the PPIs can be calculated based on them. If the gathering of the measures is carried out automatically, BAM techniques [12] can be used to provide near real time monitoring of business activities, measurement of PPIs, their representation in dashboards, and automatic and proactive notification in case of deviations [13]. Furthermore, this information, either obtained automatically or not, can be analysed using techniques from the fields of process mining [14, 15], business process intelligence [16], data warehousing and classical data mining

with the goal of evaluating and improving business process models and their implementation.

Finally, PPIs can change because of an evolution of the business process and, hence, they should be updated accordingly, or they can also change independently of the business process. For instance, because there has been a change in the business strategy that involve a change in the way a process is evaluated, or because the evaluation of the PPIs has revealed they are not measuring all important parts of the process. In any case, the change involves a new definition of PPIs, either creating new ones or changing old ones.

The management of PPIs that we have described can be improved by the contributions presented in this paper as follows. On the one hand, the PPINOT metamodel provides all of the steps that compose the PPI management with a common understanding of the PPIs defined for a business process. This is a requirement to enable an automated management of PPIs from their definition to their instrumentation, calculation or evaluation like having a well-defined business process model is a requirement to enable an automated business process management. On the other hand, the design-time analysis of the relationships between PPIs and BP elements can be used to automatically find out the business process elements that may have an influence on a PPI at design-time, which helps during the definition of PPIs to make sure that the PPIs measures all important parts of the process, and the BP elements that must be measured to calculate the PPIs, which helps in the step of deciding which measures must be taken during the execution of the process to calculate the PPIs.

3. PPINOT: A Metamodel for Defining PPIs

When managing PPIs, the first obstacle is to delimit a PPI conceptually since there is no consensus about the key elements and their relationships that need to be taken into account when defining PPIs. Consequently, it is necessary to design a representation simple and easy to understand, but also expressive enough to accommodate the different domains and situations where PPIs can be defined. In this paper, we tackle this problem by introducing the PPINOT metamodel. This metamodel is the result of an extensive analysis of a variety of PPIs defined by different organisations, a careful study of the related literature and a process of successive refinements of the metamodel after applying it to different scenarios.

The following sections detail the main features of the metamodel, which has been driven by these requirements:

Must have a high expressiveness. The metamodel must have a high expressiveness with respect to three different dimensions:

- PPIs properties: As with other indicators, it is recommended that PPIs satisfy the SMART criteria [17]. SMART is an abbreviation for five characteristics of good indicators, namely: *Specific* (it has to be clear what the indicator exactly describes), *Measurable* (it has to be possible to measure a current value and to compare it to the target one), *Achievable* (it makes no sense to pursue a goal that will never be met), *Relevant* (it must be aligned with a part of the organisation's strategy, something that really affects its performance) and *Time-bounded* (a PPI only has a meaning if it is known the time period in which it is measured). Therefore, the metamodel must include all the information that is necessary to define PPIs according to the SMART criteria.

- Type of measure: The metamodel must be able to express the measures used in all of the PPIs found in both the literature review and in the organisations whose PPIs have been analysed so that it can provide a solid basis for defining PPIs in any organisation.

- Type of scope: The metamodel must be able to express a variety of filters that selects the process instances that are considered for the computation of the PPI.

Must keep the traceability with a business process model. The definition of a PPI in the metamodel must be done so that it can always be traced back to the BP elements used in its definition. As stated in Section 1, this is a desirable property since it provides several benefits such as the implementation of a tool that automatically instruments an open source BPMS to calculate the values of PPIs.

Must be possible to use it with different business process modelling languages. The metamodel must be independent of the language used to model the business process, but it must be able to use it with standard business process models such as BPMN.

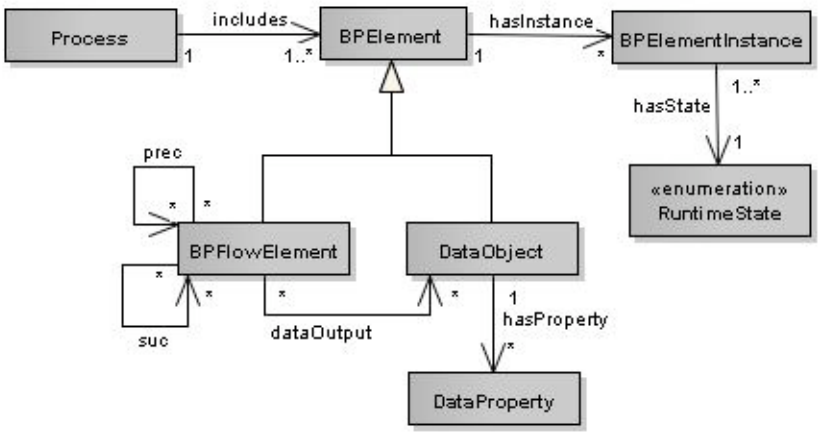

Figure 2: Abstract Business Process Modelling Language

3.1. An Abstract Business Process Modelling Language for Defining PPIs

The PPINOT metamodel must keep the traceability with a BP model, which means it must include elements that relate PPIs with business process elements. However, at the same time, it must be independent of the language used to model the business process. To address both requirements at the same time, the solution used in the PPINOT metamodel is to define an abstract business process modelling language so that any business process modelling language whose elements can be mapped to it can be used together with the PPINOT metamodel. Note that the abstract business process modelling language is not a complete BP modelling language, but only includes the minimum set of elements that is necessary to define PPIs.

The Abstract Business Process Modelling Language is composed of the elements depicted in Figure 2. Every business process includes BP elements that can be flow elements or DataObjects. BP flow elements have two relationships: *suc* and *prec* that references to the BP flow elements that succeeds or precedes the BP flow element in the control flow, respectively. They also have a *dataOutput* relationship to relate BP flow elements with the data object they modify. Examples of flow elements in typical BP modelling languages are activities or events. Regarding data objects, they have a set of data properties defined.

Any BP element when instantiated has a state associated that changes as the process instance is executed. The set of state values for every BP element changes depending on the language or notation. Except for DataObjects,

whose state values are usually defined by the user, we consider that there exist one or a set of values corresponding to the start or activation of the BP element, and one or a set of values for its end. For instance, in BPMN 2.0 activities can have the following states: ready, active, withdrawn, completing, completed, failing, failed, terminating, terminated, compensating and compensated. In this case, the start corresponds to the change to state active, and the end can correspond to withdrawn, completed, failed, terminated or compensated.

Any business process modelling language that we want to use together with the PPINOT metamodel must be bound to this abstract business process modelling language. This binding involve two actions: mapping the elements of the BP modelling language to the ones used in the abstract BP modelling language and defining which elements can be used together with the modelling concepts defined by the PPINOT metamodel. Appendix C provide details about how this binding can be done with BPMN 2.0.

3.2. Conceptual Modelling of PPIs

Process Performance Indicators (PPIs) can be defined as quantifiable metrics that allow to evaluate the efficiency and effectiveness of business processes. They can be measured directly by data that is generated within the process flow and are aimed at the process controlling and continuous optimization [3]. The PPINOT metamodel models a PPI (c.f. Figure 3) according to this definition and taking into account the requirement that its properties must include all the information that is necessary to define PPIs according to the SMART criteria. In particular, its attributes are defined as follows:

- `identifier: string`. Every PPI must be uniquely identified by an identifier.
- `name: string`. It provides a descriptive name for every PPI.
- `relatedTo: Process`. It references to the process for which the PPI is defined.
- `goals: string [0..*]`. It allows the user to establish the strategic or operational goal/s that the PPI is related to. It highlights the relevance of the PPI (connecting to the *Relevant* characteristic of the

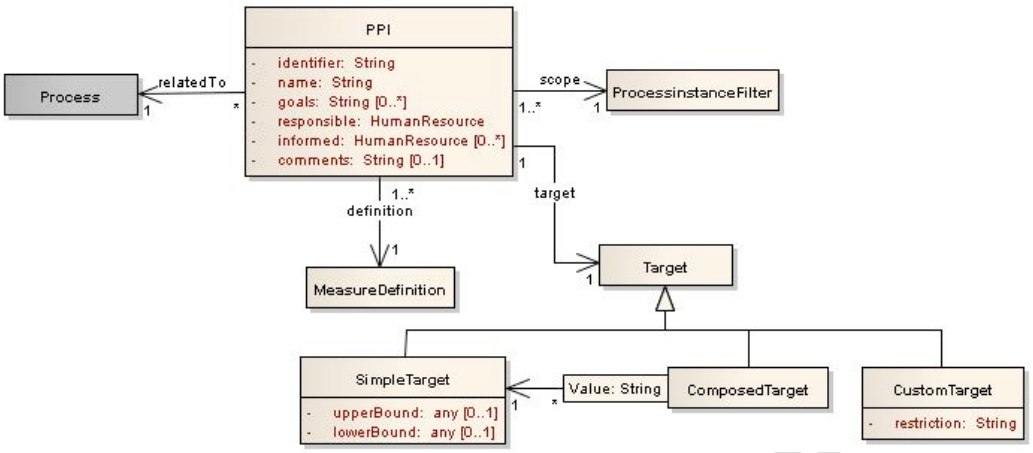

Figure 3: PPIs in PPINOT Metamodel

SMART criteria). It can be fulfilled with an expression in natural language. A more formal definition of the relationship between the PPI and the organisational goals is out of the scope of this paper. Some approaches regarding this issue can be found in [10, 18]

- `definition: MeasureDefinition`. It provides a definition about how the indicator is measured. This field is related to the two first characteristics of the SMART criteria (*Specific* and *Measurable*).

- `target: Target`. Every PPI has an associated target value to be reached indicating the consecution of the previously defined goals. In order to fulfill the *Achievable* characteristic of the SMART criteria, this target value must be reasonable, based on previous experiences and predictions based on simulations. PPINOT allows the definition of three kinds of target values: a simple target, a composed target and a custom target. A simple target is used to specify the lower bound and/or upper bound that make up the range within which the PPI value should be (If only an upper bound is defined, it acts as a maximum; if only a lower bound is defined, it acts as a minimum; finally, if both bounds are set, they define a range within which the PPI value must be). The composed target allows to define several target values or ranges, for those cases where the value of the PPI is a map (*e.g.* PPI7 and PPI8 from our case study). Finally the custom target offers the possibility to define a restriction that the PPI value

must fulfill (allowing a higher case mix for the target value, e.g. utility functions can be defined, or a metamodel of preferences like the one presented in [19] can be used).

- `scope: Filter`. This attribute indicates the subset of instances of the perviously specified process that must be considered to compute the PPI value. This field is related to the *Time-Bounded* characteristic of the SMART criteria.

- `responsible: HumanResource`. It refers to the human resource in charge of the PPI. This human resource can be a person, a role, a department or an organisation. A more detailed definition of the types of human resources are out of the scope of this metamodel. However, some approaches regarding this issue can be found in [20, 21].

- `informed: HumanResource [0..*]`. It represents the human resources that are interested in the PPI, i.e., who must be informed. This human resource can be also a person, a role, a department or an organisation. Unlike the responsible, which must be only one resource, there may be many resources informed about the state of the PPI.

- `comments: String`. Other information about the PPI that cannot be fitted in previous fields can be recorded here.

In the following sections, we detail how `MeasureDefinition` and `Filter` are defined in the PPINOT metamodel. Furthermore, we also introduce the concept of `Condition`, which is necessary to express the relationship between `MeasureDefinitions` and the business process.

3.3. Measure definitions

Figure 4 depicts the two dimensions into which the definition of measures for PPIs can be classified. The first dimension (Y axis) is the number of process instances that are necessary to calculate the PPI value. There are two possible values in this dimension, namely: *single-instance measures* if a single process instance is used to calculate the measure, and *multi-instance measures* if the PPI value is calculated using a set of process instances. Usually, most PPIs are defined using multi-instance measures. The second dimension (X axis) represent the types of measure that can be used to calculate the value of the PPI, namely: time, count, condition, data or derived.

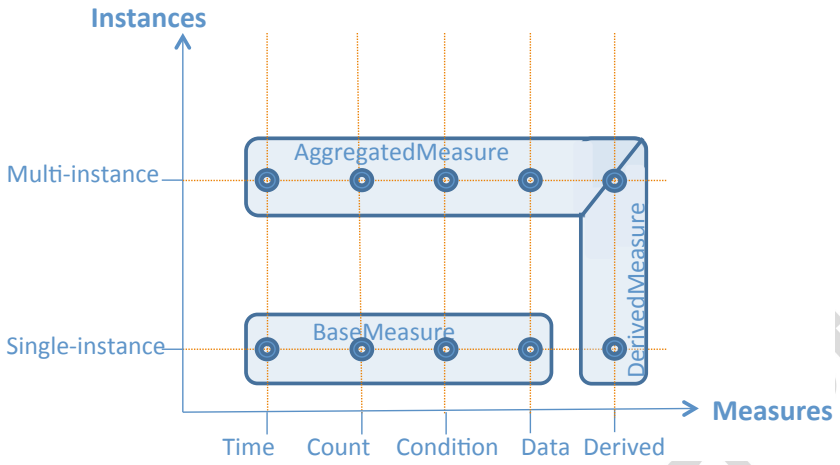

Figure 4: PPI dimensions

These dimensions are captured in the PPINOT metamodel by means of three classes: `BaseMeasure`, `AggregatedMeasure` and `DerivedMeasure` (cf. Figure 5). The relationship of these classes with the dimensions are depicted in Figure 4. A `BaseMeasure` represents a single-instance measure that measures time, count, conditions or data. An `AggregatedMeasure` represents an aggregation of single-instance measures, i.e., a multi-instance measure that measures time, count, conditions or data. Finally, a `DerivedMeasure` represents either a single-instance or a multi-instance measure that calculates the value of the PPI by performing a mathematical function over other measures.

Note that derived multi-instance measure can be defined either as a `DerivedMeasure`, i.e., performing a mathematical function over other multi-instance measures, or as an `AggregatedMeasure`, i.e., aggregating several derived single-instance measures. Both ways of defining them are necessary because a derived multi-instance measure cannot always be defined as an aggregation of derived single-instance measures. For instance, a derived multi-instance measure such as $\frac{max(timeinanalyseincommittee)}{max(timeinprocess)}$ cannot be defined as $max(\frac{timeinanalyseincommittee}{timeinprocess})$.

Next we detail how each type of measure definition can be specified in the PPINOT metamodel. In addition, Appendix B provide a complete set of invariants for the PPINOT metamodel defined in OCL.

Time Measure. It measures the duration of time between two time instants. For instance:

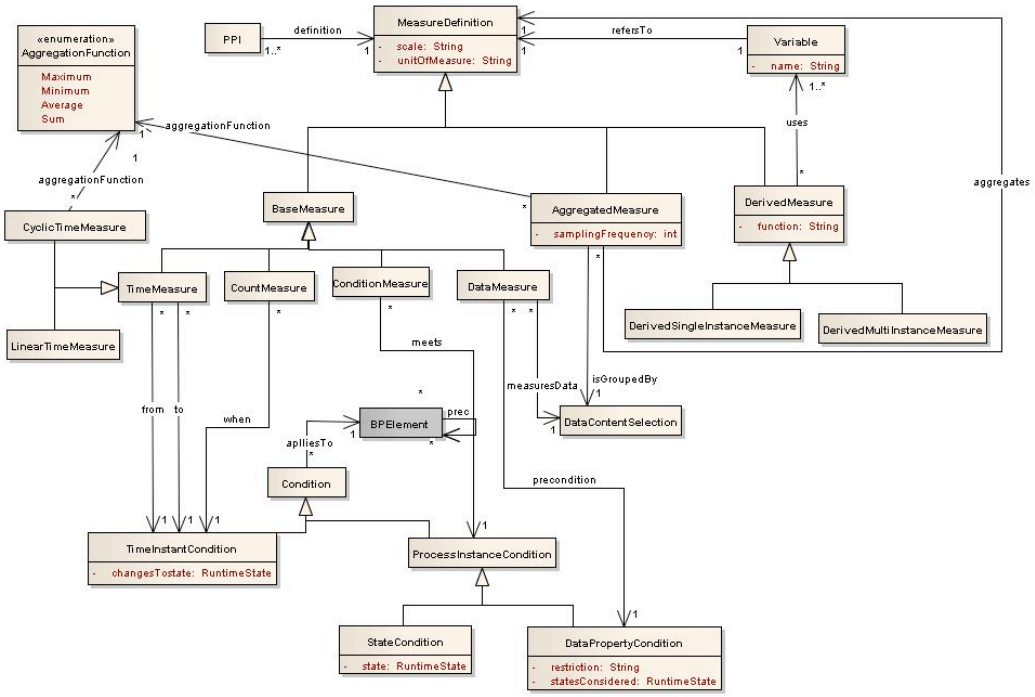

Figure 5: Measure definition in PPINOT Metamodel

The duration between the time instant when activity analyse RFC changes to state active and the time instant when activity analyse RFC changes to state completed.

In the PPINOT metamodel, the two time instants are represented by means of associations `from` and `to` between class `TimeMeasure` and class `TimeInstantCondition`. This latter class is used to model time instants by defining the BP element to which the condition applies (association `appliesTo`) and its change of state (attribute `changesToState`) or trigger in case of an event. Although this definition is enough for modelling usual time measures, there is one consideration that needs to be done if the time measure is taken between elements located within a loop. In this case, two ways of measuring time can be considered, namely:

- Linear (class `LinearTimeMeasures`), in which the measure is defined taking into account the first occurrence of the time instant condition `from` and the last occurrence of the time instant condition `to`.

- Cyclic (class `CyclicTimeMeasures`), in which the measure is defined by aggregating the values for the time between the pairs of the time instant conditions of each iteration. The kind of aggregation is defined by means of attribute `aggregationFunction`

Count Measure. It measures the number of times something happens. For instance:

The number of times activity analyse RFC changes to state completed.

In the PPINOT metamodel, the aforementioned "something happens" is modelled by means of association `when` between class `CountMeasure` and class `TimeInstantCondition`.

Condition Measure. It is a boolean value that measures the fulfillment of certain condition in both running or finished process instances. There are two types of conditions depending on whether it refers to the state of a BP element such as:

The fulfillment of the condition activity analyse in committee is currently in state active.

or to a restriction of a DataObject, such as:

The fulfillment of the condition priority=high over the dataObject RFC.

In the PPINOT metamodel, the condition whose fulfillment is being measured is modelled by means of association `meets` between class `ConditionMeasure` and class `ProcessInstanceCondition`. In addition, `ProcessInstanceCondition` is refined into the two types of conditions, namely `StateCondition` and `DataPropertyCondition`. In the first one the condition must include the state (attribute `state`). In the second one, the condition can include restrictions on both the content of the dataObject (attribute `restriction`) and its state (attribute `statesConsidered`). Note that the PPINOT metamodel does not prescribe any specific language for defining the restrictions on the content of the dataObject, but it is something specific of the BP modelling language used.

Data Measure. It measures the value of a certain part of a dataObject. For instance:

The PPI is defined as the value of information systems of RFC.

In the PPINOT metamodel, the data measure is modelled by means of attribute `measuresData` that selects the part of a dataObject that is being measured (*e.g.* information systems in the previous example). Furthermore, attribute `precondition` allows one to specify a condition that the dataObject must fulfill when the measure is performed (for instance to be in certain state). Finally, note that the PPINOT metamodel does not prescribe any specific language for defining attribute `measuresData` since it depends on the way the dataObject is modelled. For instance, if the dataObject is an XML document, `measuresData` could be an XPath expression pointing to a specific part of the XML.

Derived Measure. It is defined as a mathematical function over one or more measure definitions. There are two types of derived measures depending on whether the measure definitions are single-instance or multi-instance measures. An example of this measure is:

*The PPI is defined as the mathematical function (a/b)*100 where a is the number of times dataObject RFC is in state approved and b is the number of times dataObject RFC is in state registered.*

In the PPINOT metamodel, derived measures are modelled by means of attribute `function`, which defines the mathematical function and association `uses`, which is used to relate the variables used in `function` with their corresponding measure definitions.

Aggregated Measure. It is defined by aggregating one of the previous measures in several process instances using an aggregation function such as sum or average. Furthermore, when aggregating measures it is possible to group them by the content of a certain dataObject. An example of this measure is:

The PPI is defined as the sum of the number of times event Receive RFC is triggered and is grouped by project of RFC.

In the PPINOT metamodel, aggregated measures are modelled using attribute `aggregationFunction`, which defines the kind of aggregation is being applied, association `aggregates`, which specifies the single-instance measure that is being aggregated, and attribute `isGroupedBy`, which may define the grouping of the measure. Finally, a sampling frequency can be defined, so that we do not need to measure every instance, but one out of X, being X the sampling frequency. This makes sense in environments where taking a measure is hard or costly (*e.g.* when the measure can not be obtained automatically).

3.4. Scope

The scope of a PPI can be seen as a filter that selects the process instances that are considered for the computation of the PPI. In particular, if the PPI is defined as a single-instance measure, the filter selects the set of instances whose value must be compared to the target value. If the PPI is defined as a multi-instance measure, the filter determines the set of instances that have to be taken into account when computing the value of the PPI.

Since there are different types of filters that can be applied to the process instances, the PPINOT metamodel (cf. Figure 6) allows the definition of the scope based on:

- The last instances that have been executed (`LastInstancesFilter`)
- A temporal condition over the process instances (`TimeFilter`)
- The state of the process instances (`ProcessStateFilter`)
- Any combination of them using *and*, *or* and *not* (`ComposedFilter`).

In addition, for the particular case of the `TimeFilter`, two associations are defined. First, the temporal condition, which allows the selection of instances that started or finished "before", "before or at", "after" or "after or at" a certain point in time. This point in time can be a concrete date or a time window defined by the time from now and the unit. And second, the periodicity, which indicates the frequency with which the PPI is calculated. As usual, there are four types of periodicity: daily, weekly, monthly and yearly. If a weekly periodicity is selected, the day of the week must be completed, and for the case of monthly periodicity, whether to take into account the day of the month (e.g. 3rd January) or the day of the week

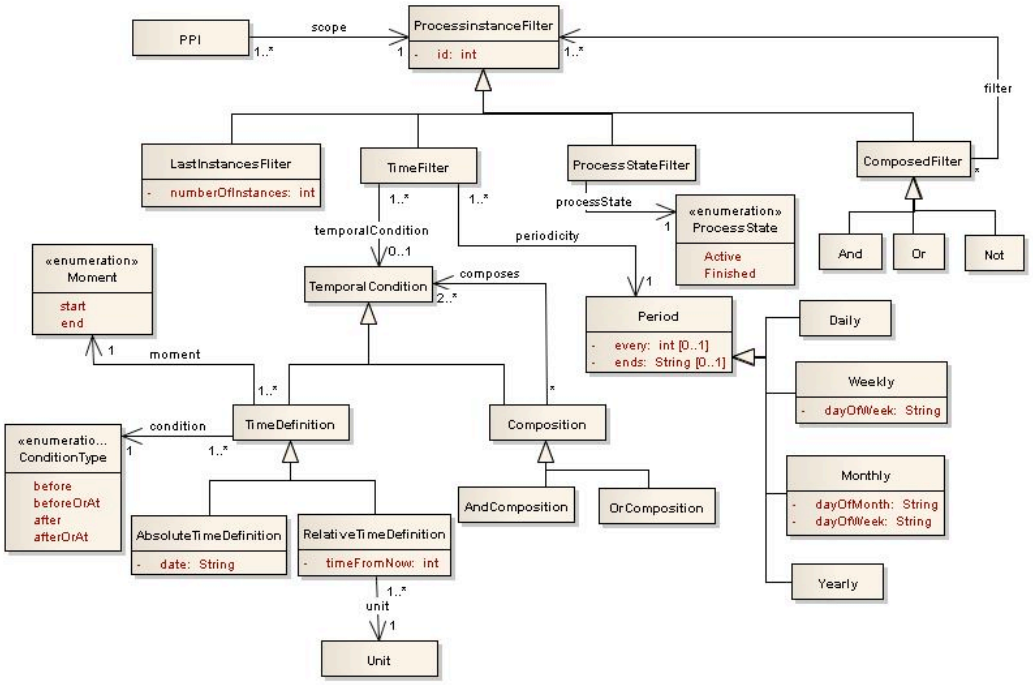

Figure 6: Filter definition

(e.g. third tuesday of the month) must be selected. The frequency of such periodicity (e.g. every 2 months, for a monthly periodicity) and when to finish taking such measure (ends 31-12-2014) can also be specified.

4. Automated Analysis of Relationships between PPIs and BP Elements

The automated analysis of PPIs can be defined as the computer-aided extraction of information from PPI models and instances and other models related to them. This analysis allows to investigate properties of PPI specifications and their relationships with other models. It is often useful to define the analysis of models in terms of analysis operations, which take a set of parameters as input and returns a result as output, as it has been successfully done in other fields such as feature models [22].

In this paper we focus on the automated analysis of the design-time relationships between PPIs and BP elements, which is a subset of the automated analysis of PPIs. In the remainder of this section we detail two kind of rela-

tionships that can be defined between PPIs and BP elements and a number of analysis operations that can be defined for each of them.

4.1. Operations for the relationship measured by

A BP element is *measured by* a PPI when the value of the PPI is calculated by measuring some aspect of the execution of the BP element. For instance, let us say we have a PPI that measures the *average lifetime of an approved RFC* and, hence, it can be defined as the average of the duration between the time instant when the event *Receive RFC* is triggered and the time instant when the end event *Report RFC Approved* is triggered. In this case, the elements measured by the PPI are the event *Receive RFC* and the event *Report RFC Approved* since the value of the PPI is calculated by measuring the moment in which both events are triggered.

This relationship can be straightforwardly inferred from the measure definition of a PPI in the PPINOT metamodel. If the measure definition is a base measure, then the elements measured by the PPI are those to which the condition used by the measure definition applies. If the measure definition is an aggregated measure or a derived measure, then the elements measured are those measured by the base measure that the aggregated or derived measure aggregates or uses, respectively. Two analysis operations can be defined based on this relationship.

4.1.1. BP elements measured by a set of PPIs

This operation takes a set of PPIs and their corresponding BP model as input and returns the set of business process elements that are measured by those PPIs.

$$MeasuredBPElement(PPI[1..*], BP) : BPElement[0..*]$$

This operation is useful when processes are instrumented to take the measures that are necessary to calculate the PPIs. It provides information that helps process analysers and developers to abstract away from other aspects of the PPIs and focus on the BP elements whose execution must be measured. Furthermore, if PPIs change, it can provide useful information about whether new BP elements should be instrumented.

4.1.2. PPIs that measure a given BPElement

This operation takes a set of PPIs, their corresponding BP model and a BP element as input and returns the PPIs that are calculated by measuring the given BP element.

$$MeasuredPPIs(PPI[1..*], BP, BPElement) : PPI[0..*]$$

The information provided by this operation is useful to find out about which are the PPIs that are affected if there is a BP element whose execution cannot be measured.

4.2. Operations for the relationship involved in

A BP element is *involved in* a PPI when it has an influence in the value of the PPI. For instance, if we take the same example as before (the *average lifetime of an approved RFC*), the elements involved in the PPI are all the elements that by taking more or less time to execute make the average lifetime longer or shorter. In this case these elements are the element in which the time measure ends (event *Report RFC Approved*) and all the elements between that element and the element in which time starts to be counted (event *Receive RFC*), i.e. activity *Analyse in committee*, activity *Analyse RFC*, activity *Elevate decision to committee*, activity *Analyse in committee* and activity *Approve RFC*. Note that event *Receive RFC* is not included since time starts counting when *Receive RFC* is triggered, which means it does not have any influence on the duration of the RFC lifetime.

Unlike relationship *measured by*, the set of elements involved in a PPI cannot always be directly inferred from the PPINOT metamodel since there are many factors that can make a BP element to have an influence in the value of a PPI. For example, the type of an RFC may have an influence on the lifetime of an approved RFC since some activities may take more time if the RFC involves an adaptive change than if it involves a corrective change.

Nevertheless, it is possible to leverage the definition of the PPI and the control flow of the business process to make a design-time estimation of the BP elements that may have an influence on the PPI as summarised in Table 2. This estimation can be later refined by means of techniques similar to those described in [10] such as:

- Knowledge of domain experts, which can be used to determine possible domain-specific influences of a BP element in a PPI.

Measure type	Elements involved
Time	(1) The elements that are executed between the start and the end of the time measure and (2) the elements at the start or at the end if some of the time of their execution is included in the time measure.
Count	(1) The element that is being counted and (2) the XOR gateways that have taken the execution path to that element.
Condition	(1) The element used in the condition and (2) if the condition involves a data object, the activities that can write in it.
Data	(1) The data object whose value is measured and (2) the activities that can modify the data object.
Aggregated	(1) The elements involved in the measure that it aggregates and (2) if it groups results by some value, the data object that provides it.
Derived	The elements involved in the measures used in the mathematical function applied to calculate the value.

Table 2: BP elements involved in a PPI

- Data mining techniques, which can be applied to the data collected during the execution of the process. For instance, an analysis of the executions of a process may conclude that the content of the RFC have an influence on the previous PPI if the likelihood of having a longer average lifetime changes depending on the type of RFC.

Obtaining automatically a design-time estimation of the BP elements that have an influence on a PPI is useful because of the following reasons. First, it is not necessary to have execution data available, which is an advantage when execution data is very costly to obtain or when it refers to business process or PPIs that are still being designed and, hence, no execution data is available. Second, although domain experts can determine possible domain-specific influences of a BP element with a PPI without execution data, this task is error prone, specially when the number of PPIs or elements are high. Therefore, this automated estimation can be used as an input or a complement for domain experts to determine domain-specific influences. Finally, if the design-time estimation is obtained automatically, it helps the modeller

of PPIs to quickly analyse different configurations of PPIs in order to find one that covers the most relevant BP elements.

Regardless of whether the relationship is inferred at design-time or refined using one of the aforementioned techniques the following analysis operations can be defined.

4.2.1. BP elements involved in a PPI

This operation takes a set of PPIs and the corresponding BP model as input and returns the set of business process elements involved in those PPIs.

$$InvolvedBPElement(PPI[1..*], BP) : BPElement[0..*]$$

Related to this operation other similar operations can be defined that returns:

- The BP elements that are not involved in any PPI:

$$NotInvolvedBPElement(PPI[1..*], BP) : BPElement[0..*]$$

- The BP elements that are involved in all of the PPIs:

$$InvolvedInAllBPElement(PPI[1..*], BP) : BPElement[0..*]$$

The information provided by these operations is useful to find out which are the elements in a business process that are not involved with any PPI. This would mean that those elements are not being taken into consideration by the current set of PPIs and, hence, that it may be convenient to introduce or modify a PPI to cover them. It is also useful when a PPI must be replaced with others (maybe because it is very costly to obtain its value) in order to assure that every element of the business process that was taken into consideration before is taken into consideration in the new case.

4.2.2. PPIs associated to a BP element

This operation is the inverse of the previous one. It takes a set of BP elements, the BP model that contains those BP elements and the PPIs model defined for that BP as input and returns the set of PPIs that are associated to those BP elements.

$$AssociatedPPI(BPElement, BP, PPI[1..*]) : PPI[0..*]$$

For instance, the following PPIs are associated to the activity *Analyse RFC*:

- PPI5 because it directly measures the average duration of that activity.
- PPI6 because it measures the number of RFCs with state *in analysis*, which is the state in which RFCs are while the activity *Analyse RFC* is being executed. Therefore, this number may be influenced by the throughput of this activity amongst other reasons.
- PPI9 because it measures the average lifetime of a RFC and, hence, the duration of activity *Analyse RFC* has an influence on it.

The information obtained in this operation can assist during the evolution of business processes. For instance, if part of the business process evolves and is modified (e.g., an activity is deleted), this operation allows to identify which PPIs will be affected and may be updated.

4.2.3. PPIs associated to the same, a subset or a superset of the elements of other PPI

These operations take a PPI, the set of PPIs defined by a business process and the corresponding BP model as inputs and returns the set of PPIs that are associated to either:

- the same elements as the given PPI:

$$PPISameElements(PPI, PPI[1..*], BP) : PPI[0..*]$$

- a subset of the elements of the given PPI:

$$PPISubsetElements(PPI, PPI[1..*], BP) : PPI[0..*]$$

- a superset of the elements of the given PPI:

$$PPISupersetElements(PPI, PPI[1..*], BP) : PPI[0..*]$$

The information provided by these operations can be useful to find out about possible redundancies amongst the PPIs defined for a business process. For instance, many PPIs associated to a superset of the elements of a given PPI could be a sign that the PPI is providing redundant information. Nevertheless, these operations just provides hints and it is always the task of the PPI modeller to decide whether an indicator provides redundant information or not.

5. Automating the Analysis of the PPINOT Metamodel Using DLs

In the previous section, we have provided a description of operations for design-time analysis of relationships between PPIs and BP elements. However, their semantics has been defined in an intuitive way. In this section, we provide a precise definition of them by means of a semantic mapping. A semantic mapping is a way to provide semantics to a model by mapping the concepts into a semantic domain, i.e., a target domain whose semantics has been formally defined [23]. The advantage of defining such semantic mapping is that it allows one to use the techniques specific to the target semantic domain for analysing the source models [24].

In this paper, we have chosen description logics (DLs) [25] as the semantic domain for our mapping because of two reasons. On the one hand, the definition of a set of PPIs for a business process fits nicely into the way DLs express their concepts and, hence, it provides a very natural way to describe the problem. On the other hand, powerful reasoning systems for DL like Racer [26], Hermit [27] and Pellet [28] have been developed, allowing thus to build upon such standard DL reasoning services to automatically analyse PPIs instead of implementing ad-hoc analysis algorithms.

Note that other semantic mappings to formalisms different than DLs can be defined for the PPINOT metamodel. However, a discussion of the appropriateness of other formalisms, although a relevant research topic, is outside the scope of this paper.

5.1. Foundations of Description Logics

A DL-based knowledge base (KB) is a finite set of terminological and assertional sentences. It thus has two components: the TBox and the ABox. The TBox describes *terminology*, i.e., the ontology in the form of *concepts*, which denote sets of individuals, *role* definitions, and their relations; the ABox contains *assertions* about *individuals* using the terms from the ontology.

Semantically, DL are found on predicate logic, but their language is formed so that it would be enough for practical modeling purposes and also so that the logic would have good computational properties such as decidability [29]. As exemplified in Table 3 and Table 4 [1], DL have a rich set

[1] In this paper a syntax commonly used for DLs [25] is used.

Axiom	DL Syntax	Example
Sub concept	$C_1 \sqsubseteq C_2$	$TimeMeasure \sqsubseteq BaseMeasure$
Equivalent con.	$C_1 \equiv C_2$	$ComposedFilter \equiv And \sqcup Or \sqcup Not$
Disjoint with	$C_1 \sqsubseteq \neg C_2$	$TimeMeasure \sqsubseteq \neg CountMeasure$
Sub role	$P_1 \sqsubseteq P_2$	$when \sqsubseteq relatesTo$
Inverse role	P^-	$isUsedBy^-$
Transitive role	P^+	suc^+
Equivalent role	$P_1 \equiv P_2$	$definition \equiv isUsedBy^-$
Role composit.	$P_1 \circ P_2$	$refersTo^- \circ uses^- \sqsubseteq isUsedToCalculate$
Functional role	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1definition$
Inv. func. role	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1isUsedBy^-$

Table 3: Some DL axioms

Constructor	DL Syntax	Example
Intersection	$C_1 \sqcap \cdots \sqcap C_n$	$MeasureDefinition \sqcap DataMeasure$
Union	$C_1 \sqcup \cdots \sqcup C_n$	$TimeMeasure \sqcup DataMeasure$
Complement	$\neg C$	$\neg DerivedMeasure$
One of	$x_1 \sqcup \cdots \sqcup x_n$	$PPI1 \sqcup PPI3 \sqcup PPI8$
All values from	$\forall P.C$	$\forall definition.MeasureDefinition$
Some values	$\exists P.C$	$\exists refersTo.TimeMeasure$
Has value	$P.x$	$isUsedBy.PPI1$
Max cardinality	$\leq nP$	$\leq 1appliesTo$
Min cardinality	$\geq nP$	$\geq 1uses$

Table 4: DL concept constructors

of knowledge representation constructs that can be used to formally specify PPI-domain knowledge, which in turn can be exploited by description logic reasoners for purposes of inferencing, i.e., deductively inferring new facts from knowledge that is explicitly available [30]. Amongst others, DL reasoners usually implement the following reasoning operations:

- $satisfiability(C)$: It determines whether a description of the concept C is not contradictory with the KB.
- $subsumes(A, B)$: It determines whether concept A subsumes concept B, i.e., whether description of A is more general than description of B.
- $individuals(C)$: It finds all individuals that are instances of concept

C.

- $realization(i)$: It finds all concepts to which the individual i belongs.

5.2. Mapping the PPINOT Metamodel into DL

The mapping of the PPINOT metamodel into DL involves defining a DL knowledge base that includes the classes and relations of the metamodel as axioms of its TBox. In particular, we must create one DL concept for each class of the metamodel (keeping the same names) and set in the knowledge base the hierarchies that appear in the metamodel using the subclass axiom (e.g. $BaseMeasure \sqsubseteq MeasureDefinition$).

The directed associations in the metamodel are mapped into roles of the TBox whose domain is the DL concept corresponding to the class of the metamodel that acts as source, and whose range is the DL concept corresponding to class that acts as target. The cardinality restrictions of the associations are mapped as concept inclusion axioms to the source class. For instance, the cardinality restriction: "*each Condition appliesTo at most one BPElement*" is mapped into the following axiom: $Condition \sqsubseteq \leq 1 appliesTo.BPElement$.

Finally, we introduce two new DL roles. Role $relatesTo$ is defined as a super-role of the roles that relate measures with conditions (i.e. $from \sqsubseteq relatesTo$, $to \sqsubseteq relatesTo$, $when \sqsubseteq relatesTo$ and so on) and role $isUsedToCalculate$ is defined as the composition of the inverse of roles $refersTo$ and $uses$ ($refersTo^{-} \circ uses^{-} \sqsubseteq isUsedToCalculate$) to relate the derived measure to the measures used to calculate it.

A complete mapping of the PPINOT metamodel and the abstract BP modelling language into a DL knowledge base can be found at `http://www.isa.us.es/ppinot`

5.3. Automated Analysis of PPIs using DL

On the basis of the mapping of the metamodel to DL, the analysis operations detailed in Section 4 can be formulated so that DL reasoners can be used to implement them. The general approach we follow to formalise these operations in DL is first to define the relationship as a new role in the knowledge base and, then to formulate the operations in terms of DL reasoning operations.

5.3.1. Operations for the relationship measured by

This relationship is formalised in DL by defining a new role in the KB ($measuredBy$), whose domain is $BPElement$ and whose range is PPI so that $measuredBy(e, p)$ means that a BP element e is *measured by* PPI p.

To give semantics to the relationship in terms of the elements defined in the KB we use the fact that the relationship *measured by* can be inferred from the measure definition of a PPI. The idea is to introduce a new role $measures$ with domain $MeasureDefinition$ and range $BPElement$ that relates measure definitions with BP elements and, then, use this new role in the definition of $measuredBy$ as follows:

$$measures^{-} \circ definition^{-} \sqsubseteq measuredBy$$

Finally, role $measures$ is used to relate each measure definition with the BP elements used in the definition. In the KB, this can be formalised using the composition of roles as follows:

$$relatesTo \circ appliesTo \sqsubseteq measures$$

$$measuresData \circ data \sqsubseteq measures$$

$$aggregates \circ measures \sqsubseteq measures$$

$$isGroupedBy \circ data \sqsubseteq measures$$

$$uses \circ refersTo \circ measures \sqsubseteq measures$$

The first two axioms define role $measures$ for base measures, the next two axioms define it for aggregated measures and the last axiom defines it for derived measures. Note that in both aggregated and derived measures role $measures$ is defined recursively based on the measure definitions they aggregate or compose.

Based on role $measuredBy$, the operations for this relationship can be easily formulated as follows:

- The BP elements measured by a set of PPIs P are those elements e that have a role $measuredBy(e, p)$, where $p \in P$. In DL, this can be expressed as the result of $individuals(\exists measuredBy.P)$.
- Similarly, the PPIs that measure a given BPElement e is equivalent to the result of $individuals(\exists measuredBy^{-}.\{e\})$.

5.3.2. Operations for the relationship involved in

Like in relationship *measured by*, the elements involved in a PPI P are determined by its definition, which is expressed in the metamodel by means of a measure definition. Therefore the approach we follow with this relationship is very similar to the one followed with relationship *measured by*.

Specifically, we define two new roles in the KB, namely: role $involvedIn$ whose domain is $BPElement$ and whose range is PPI so that $involvedIn(e, p)$ means that a BPElement e is involved in a PPI p, and role inv whose domain is also $BPElement$, but whose range is $MeasureDefinition$. Finally, the following axiom formalises the relationship between them by stating that the elements involved in a PPI are those involved in its measure definition:

$$inv \circ defines^{-} \sqsubseteq involvedIn$$

Consequently, the definition of role $involvedIn$ can be reduced to the definition of role inv. However, this relationship is more complex than relationship *measured by* and, hence, we cannot define role inv in a generic way, but it is necessary to introduce an axiom that specifies the BP elements involved in a measure definition for each measure definition m in the KB. This axiom may vary depending on the type of the measure definition (i.e. time, count, condition, data, aggregated or derived) as detailed in Table 2. Next we detail each of the possible cases.

Time measure. According to Table 2 the elements involved in a time measure tm are $\exists inv.\{tm\} \equiv ElemStart_{tm} \sqcup ElemEnd_{tm} \sqcup ElemPath_{tm}$, where:

- $ElemStart_{tm}$ is the element where time starts to be measured unless the state in which the measure starts is an end state. In that case, $ElemStart_{tm}$ is empty: $ElemStart_{tm} \equiv \exists appliesTo^{-}.(\exists from^{-}.\{tm\} \sqcap \neg\exists state.EndState)$

- $ElemEnd_{tm}$ is the element used in the condition that specifies when time ends to be measured unless the state specified by the condition is a start state, in which case it is empty: $ElemEnd_{tm} \equiv \exists appliesTo^{-}.(\exists to^{-}.\{tm\} \sqcap \neg\exists state.StartState)$

- $ElemPath_{tm}$ is the BP elements that succeeds the element where time starts to be measured and precedes the element where time ends to be measured: $ElemPath_{tm} \equiv \exists succ^{+}.(\exists appliesTo^{-}.(\exists from^{-}.\{tm\})) \sqcap \exists prec^{+}.(\exists appliesTo^{-}.(\exists to^{-}.\{tm\}))$

Count measure. In this case, the elements involved in a count measure cm are: $\exists inv.\{cm\} \equiv ElemCount_{cm} \sqcup InvolvedXor_{cm}$, where:

- $ElemCount_{cm}$ is the element that is being counted: $ElemCount_{cm} \equiv \exists appliesTo^{-}.(\exists when^{-}.\{cm\})$
- $InvolvedXor_{cm}$ is the set that includes every XOR gateway that precedes the element that is being counted: $InvolvedXor_{cm} \equiv XorGateway \sqcap \exists prec^{+}.(\exists appliesTo^{-}.(\exists when^{\cdot} - \{cm\}))$

Condition measure. The elements involved in a condition measure cm are: $\exists inv.\{cm\} \equiv ElemCond_{cm} \sqcup ElemWriters_{cm}$, where:

- $ElemCond_{cm}$ is the element whose condition is being evaluated: $ElemCond_{cm} \equiv \exists appliesTo^{-}.(\exists meets^{-}.\{cm\})$.
- $ElemWriters_{cm}$ correponds to every activity that can modify, i.e., write the data object whose state is being evaluated if there is any: $ElemWriters_{cm} \equiv \exists dataOutput.(DataObject \sqcap ElemCond_{cm})$

Data measure. The elements involved in a data measure dm are: $\exists inv.\{dm\} \equiv DataMeasured_{dm} \cup ElemWriters_{dm}$, where:

- $DataMeasured_{dm}$ is the data object that is being measured: $DataMeasured_{dm} \equiv \exists data^{-}.(\exists measuresData^{-}.\{dm\})$.
- $ElemWriters_{dm}$ is the activities that could have modified that data object $ElemWriters_{cm} \equiv \exists dataOutput.DataMeasured_{dm}$

Aggregated measure. The elements involved in an aggregated measure am are: $\exists inv.\{am\} \equiv \exists inv.\{bm\} \sqcup InvolvedData_{am}$, where:

- $\exists inv.\{bm\}$ is the elements involved in the base measure that the aggregated measure aggregates (i.e. $bm \in \exists aggregates^{-}.\{am\}$).
- $InvolvedData_{am}$ is the data that the aggregated measure groups by if there is any: $InvolvedData_{am} \equiv \exists data^{-}.(\exists isGroupedBy^{-}.\{am\})$

Derived measure. The elements involved in a derived measure dm are: $\exists inv.\{dm\} \equiv \exists inv.\{d_1\} \sqcup \cdots \sqcup \exists inv.\{d_n\}$, where $\exists inv.\{d_i\}$ is the elements involved in each of the measures used in the mathematical function applied to calculate the value ($\{d_1, \ldots, d_n\} \in \exists isUsedToCalculate.\{dm\}$).

These definitions have two limitations regarding time measures and count measures that are worth mentioning. The first one is the definition regarding time measures considers all of the elements between the first occurrence of the element that starts the time measure and the last occurrence of the element that ends the time measure. However, if the time measure is cyclic, it should only include the elements until the first occurrence of the element that ends the time measure. The consequence is that if the time measure is cyclic, then the definition may include more involved elements than it should (i.e. those between the first and the last occurrence of the element that ends the time measure). However, from a practical point of view, this only affects when the time measure is defined in a loop since in other cases the first and the last occurrence of the element that ends the time measure coincides. The second limitation is that in count measures we consider every preceding XOR gateway without excluding those opening gateways that were already closed with another one (that is, for instance, a splitting gateway with its corresponding merge).

Both limitations could be solved using another formalism to analyse the control flow of the business process and then including the result of the analysis in the KB. For instance, Petri nets could be used to obtain the process fragment between the first occurrence of the element that starts the time measure and the first occurrence of the element that ends it. However, the specific mechanism on how to do so is out of the scope of this paper.

Like relationship *measured by*, operations for relationship *involved in* can be formulated based on role $involvedIn$ as follows:

- Operation $InvolvedBPElement$, which returns the BP elements that are involved in a set of PPIs P can be formulated as $individuals(\exists involvedIn.P)$
- Operation $NotInvolvedBPElement$, which returns the BP elements that are not involved in any PPI of the set of PPIs P can be formulated as $individuals(\neg(\exists involvedIn.P))$
- Operation $InvolvedInAllBPElement$, which returns the BP elements that are involved in all of the PPIs included in a set of PPIs P can be

formulated as $individuals(\exists involvedIn.\{p_1\} \cap \ldots \cap \exists involvedIn.\{p_n\})$, where $p_1, \ldots p_n \in P$.

- Operation $AssociatedPPI$, which returns the PPIs in which a given BP element e is involved can be formulated as $individuals(\exists involvedIn^{-}.\{e\})$.

The last three operations that returns the PPIs associated to the same, a subset or a superset of the elements of a given PPI p must be done in three steps. First, a concept $InvolvedBPElement_q$ that represents all the BP elements that are involved in a PPI q ($InvolvedBPElement_q \equiv \exists involvedIn.\{q\}$) is defined for each PPI q included in the KB. This allows the DL reasoner to classify all these concepts by using DL operation *subsumes*. Then, depending on whether we are defining operation $PPISameElements$, $PPISubsetElements$ or $PPISupersetElements$, this classification is used to obtain the concepts that are equivalent or a subset or a superset to the concept $InvolvedBPElement_p$, respectively. In each case, the result is a set of $InvolvedBPElement_{p_1}, \ldots, InvolvedBPElement_{p_n}$ concepts. The final step is to obtain the PPIs $p_1, \ldots, p_n$ associated to those $InvolvedBPElement$ concepts. These are the resulting PPIs.

6. Implementation

We have developed PPINOT Tool Suite that includes: (1) two complementary tools to define PPIs, namely a graphical editor based on *Oryx* [31] to define them together with their corresponding BPs (cf. Figure 7) and a templates-based editor that uses natural language patterns to allow a textual definition of PPIs; (2) a design-time analyser of PPIs that implements the operations described in Section 4; (3) a PPIs instrumenter that allows to gather the required information to compute their values from Activiti (an open source BPMS, `http://activiti.org`), and (4) a reporter that shows these values[2]. Furthermore, PPINOT Tool Suite is BPMN 2.0 compliant, since PPIs can be defined over BP diagrams (BPDs) previously defined using this specification[3].

[2]In its current version, this reporter provides a simple list of values. We plan to extend it to improve the GUI and provide an enriched report.

[3]Further information related to PPINOT Tool suite, as well as some guidelines and the links to try it, can be found at `http://www.isa.us.es/ppinot`.

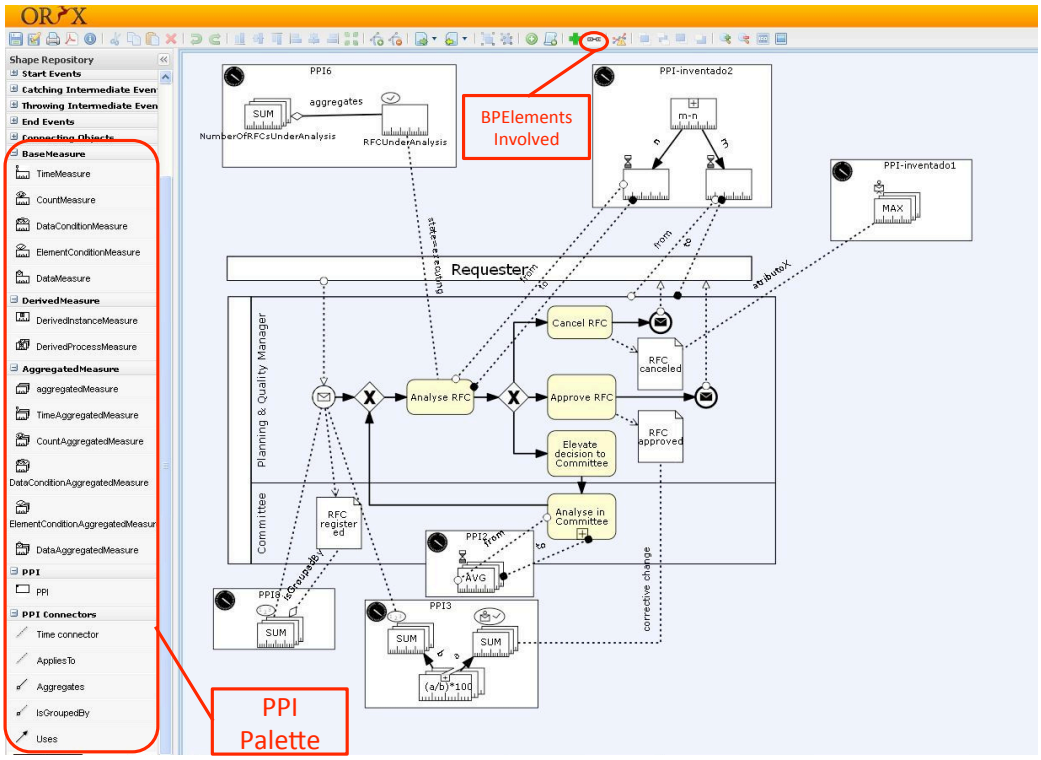

Figure 7: PPINOT screenshot

The PPINOT metamodel is the central part of the PPINOT Tool Suite since all of the tools use it as a common base for their implementation. The PPIs models used in the PPINOT Tool Suite are serialized in XML and links to refer to elements of a BPMN model are provided. Furthermore, this XML can be embedded into the XML serialization of a BPMN 2.0 process model using the extension mechanisms provided by BPMN. Therefore, models defined according to the PPINOT metamodel can be used together with BPMN models seamlessly. These XML files with information about both PPIs and BP models are generated by both editors and used by the PPINOT Instrumenter and the Reporter as inputs to instrument processes and calculate PPIs values, respectively.

Regarding the analyser, it implements the operations described in Section 4 as depicted in Figure 8. Therefore, the interface of the analyser contains those operations together with an additional load operation that must be called before any of the other operations can be invoked.

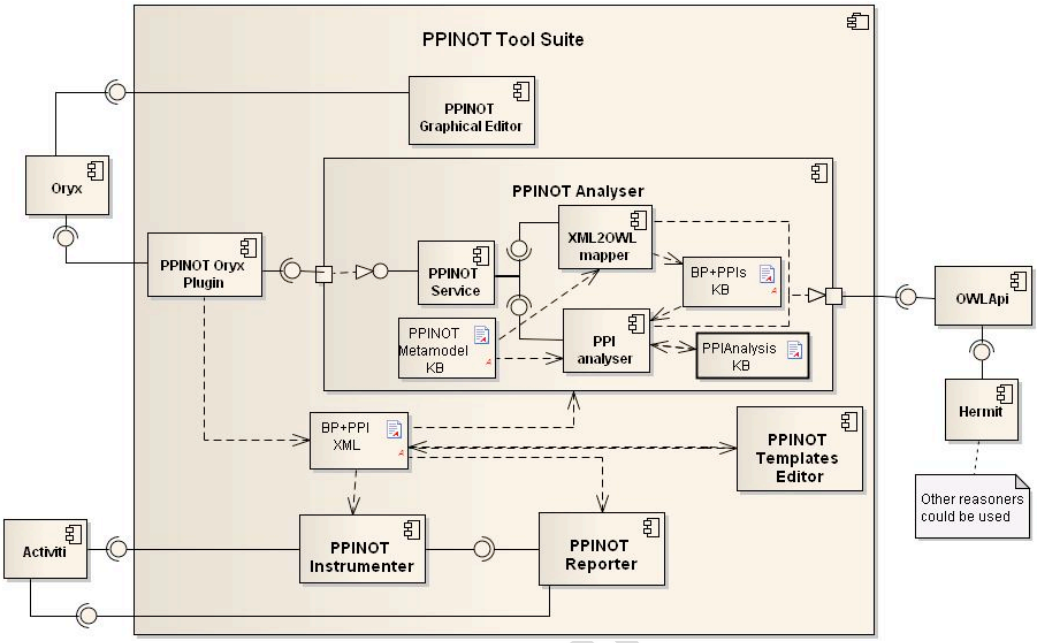

Figure 8: PPINOT component model

The goal of the load operation is to map both the BP model and the PPI model into a DL knowledge base serialized in OWL, which is a W3C recommendation for the representation of ontologies and is based on DLs. It involves the following steps. First, it receives the BP model and the PPI model together as a BPMN 2.0 xml file. Then, it maps them to the ABox of a DL knowledge base, i.e., as instances of the concepts defined in the KB. The TBox of the DL knowledge base, i.e., the concepts, have been previously defined according to the mapping described in Section 5.2. The next step is to add the axioms that are necessary to formalise the operations for the relationship *involved in* into the KB. Finally, it loads the KB serialized as an OWL file into a DL reasoner, in this case *HermiT*.

After this process, the user can invoke any other operation of the analyser, which are all implemented in a very similar way. First, the analyser invokes the appropriate reasoning operation in the DL reasoner as detailed in Section 5.3 and, then, it traces back the results obtained by the DL reasoner to the BP and PPI model and sends them to the user.

PPINOT Analyser has been integrated as a plugin on Oryx so that the user can obtain the design-time analysis information while modelling the

PPIs (cf. Figure 7), although it can be used in other environments as well.

7. Application to a Real Scenario

In order to show the application of the approach presented in this paper to a real scenario, we will use the scenario introduced in Section 2. This scenario takes place in the context of the IT Department of the Andalusian Health Service. This department engaged some years ago in an initiative to adopt the set of good practices proposed by ITIL[4]. This entailed, among others, to define business process models as well as performance indicators for them. After accomplishing this goal, they had two kinds of documents: the business process models, produced by the departments/roles in charge of the modelling and execution of business processes; and the documents containing the definition of their associated PPIs in natural language, produced by the departments/roles in charge of the definition and consecution of goals and its associated indicators. This situation led to several problems, already introduced in Section 1. First, the ambiguity, inherent in natural language, and incompleteness in PPI definitions, in the sense of missing information required to instrument business processes for the PPI values computation. Second, the lack of traceability between the two kinds of documents, that makes it really complicated to maintain the coherence across them, since changes in one document had to be reflected in the other by hand and vice-versa. As a result of the coexistence of these two worlds unable to communicate to each other, inconsistencies between them appeared and it was necessary the human intervention to solve them, which was quite tedious and error-prone.

To illustrate the application of PPINOT, let us focus on the Request For Change (RFC) management process (the one presented in Section 2), from the set of processes defined for the IT Service Management (ITSM) of this organisation. The (simplified) business process model is depicted in Figure 1 and the set of PPIs defined for it are listed in Table 1. Taking these documents as starting point, the RFC management process model was refined and its associated PPIs were defined using PPINOT Graphical Editor. Figures 9 and 10 depict the RFC management process together with the 9 PPIs defined for it[5]. It was possible to define the whole set of PPIs required,

[4]Information Technology Infrastructure Library [32].

[5]Appendix A contains the corresponding specification of these PPI definitions according to the PPINOT metamodel.

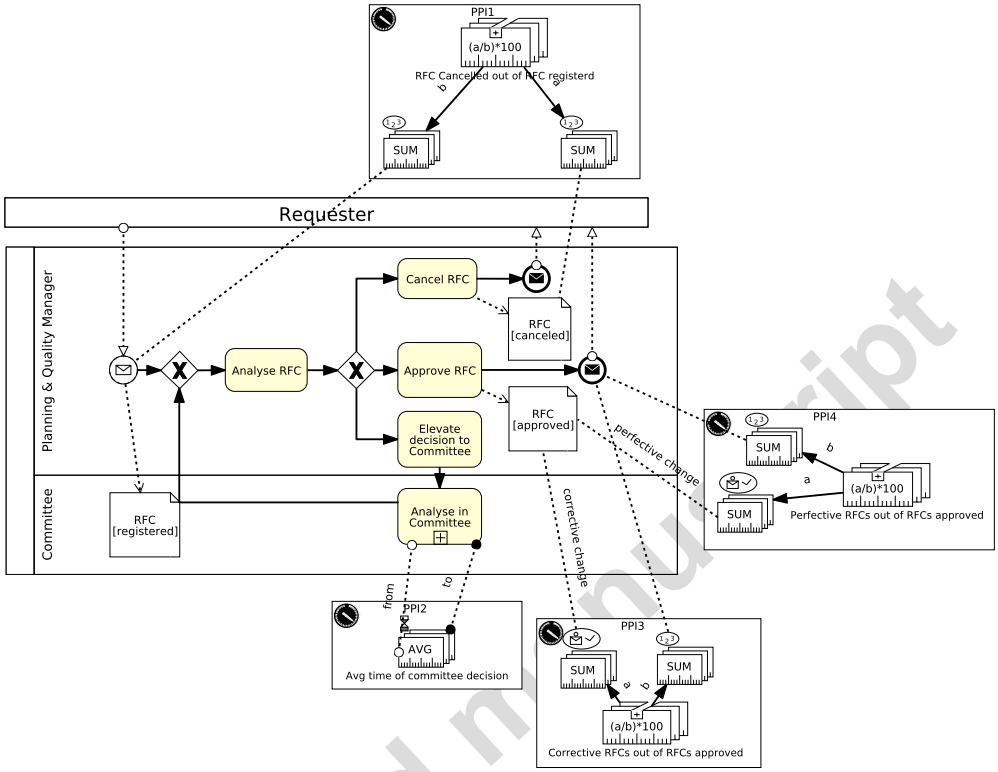

Figure 9: RFC management process together with PPIs graphically defined (PPIs 1 to 4)

what shows the expressiveness of PPINOT metamodel. In particular, within these 9 PPIs, there were 3 *time aggregated measures*, 6 *count aggregated measures*, two of them using *isGroupedBy*, 1 *state condition aggregated measure*, 2 *data property condition aggregated measures*, and 2 *derived multi-instance measures*. These definitions of PPIs eliminate ambiguities and incompleteness, since they are based on the PPINOT metamodel and contain all the information required for their computation.

Furthermore, the availability of the set of analysis operations described in Section 4 allowed them to solve the problem of the lack of traceability. In particular, these operations were mainly used in the context of process evolution. Within this organisation, it was quite frequent to raise the possibility of eliminating or modifying certain activity within a process as part of its continue improvement. In this case, the operation $AssociatedPPIs$

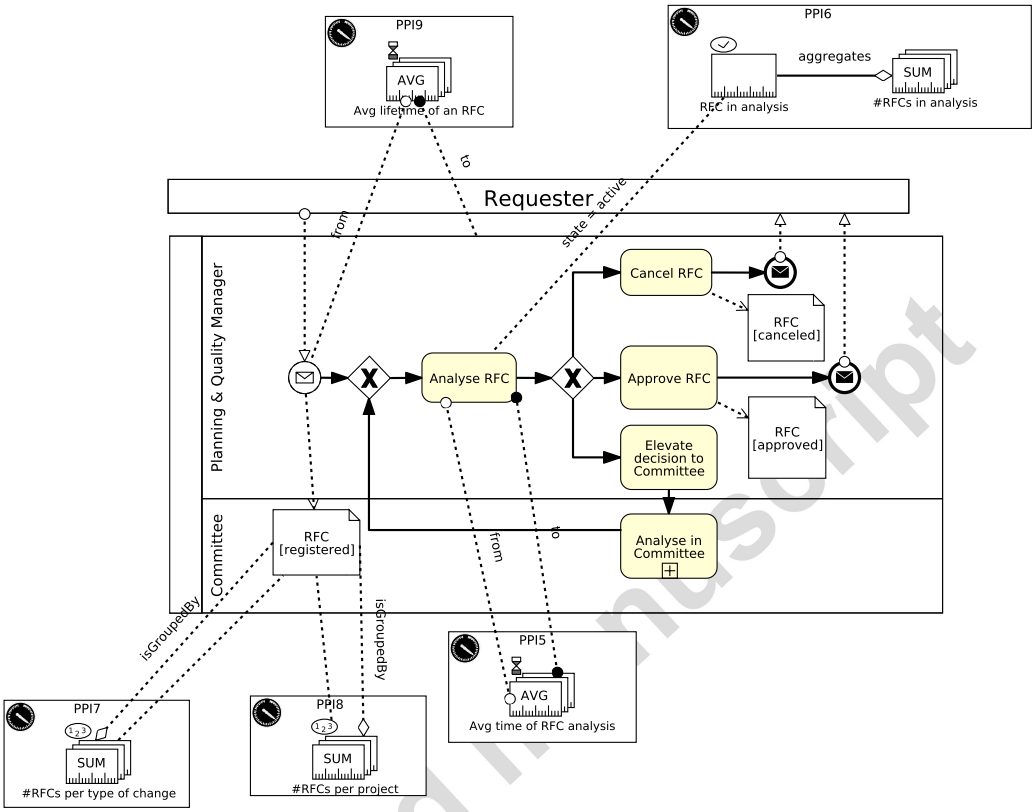

Figure 10: RFC management process together with PPIs graphically defined (PPIs 5 to 9)

was used to obtain the information about which were the PPIs that could have been affected because of the aforementioned change, so that those PPIs could be updated. Besides, the operation *MeasuredBPElements* was also used to maintain an updated list of which were the elements that needed to be measured in the process to obtain the PPI values[6]. As an example, Table 5 depicts the result of applying this *MeasuredBPElements* operation to the 9 PPIs defined for the RFC management process.

In addition, this department has also recently adopted a change in its business policy and they established that new software developments for

[6]The traceability between this list and the concrete way of obtaining the required information from the corresponding information systems was maintained manually.

Name	***MeasuredBPElements***
PPI1	Event *Receive RFC* & dataObject *RFC [cancelled]*
PPI2	Acitvity *Analyse in comittee*
PPI3	Event *Report RFC approved* & dataObject *RFC [approved]*
PPI4	Event *Report RFC approved* & dataObject *RFC [approved]*
PPI5	Activity *Analyse RFC*
PPI6	Activity *Analyse RFC*
PPI7	DataObject *RFC [registered]*
PPI8	dataObject *RFC [registered]*
PPI9	dataObject *RFC [registered]* & pool *RFC Management*

Table 5: Summary of the result of applying the operation $MeasuredBPElements$ to the 9 PPIs of the RFC management process

the Andalusian Health Service must include the definition of the business processes to which they provide support; and, as a consequence of their experience with PPINOT, it is planned to use it for the definition of PPIs for those business processes.

8. Validation

We have validated our proposal by means of several actions. On the one hand, the implementation of PPINOT Tool Suite has served as a proof-of-concept that shows the applicability of our proposal. On the other hand, we have successfully applied PPINOT to manage PPIs in three different real scenarios including the one described in previous section, the IT Department of the Andalusian Health Service, as well as the Information and Communication Service of the University of Seville and a part of the administration of the Andalusian Regional Government. Next we detail how these validation actions have allowed us to validate different aspects of our proposal.

The validation scenarios in which we have applied our proposal have helped us to test and improve the expressiveness of the PPINOT metamodel. In summary, with PPINOT metamodel we have been able to model all of the 54 PPIs for 10 different process models ranging from a Request For Change (RFC) management process to user management process and evaluation and certification management process, and belonging to 3 different

organisations[7]. All of those PPIs were defined on multi-instance measure. In particular, 14 were time aggregated measures; 54 were count aggregated measures, 2 of them using `isGroupedBy`; 1 was state condition aggregated measure, 2 were data condition aggregated measures, 1 was data aggregated measure, 8 were derived multi-instance measures and 5 were aggregated of single-instance measures. From this evaluation we can conclude that the PPINOT metamodel provide a solid basis for defining PPIs in any organisation. Furthermore, one of the main benefits that organisations obtained from the modelling of PPIs using the PPINOT metamodel was that it helped to identify ambiguities and missing information that contained previous definitions of PPIs, chiefly caused because they were expressed in natural language (an example is the PPIs detailed in Table 1). Instead, the PPI definitions obtained with PPINOT are now precise and contain all the information required to compute their values from the corresponding systems if the appropriate mechanisms are implemented.

Regarding the traceability provided by the PPINOT metamodel, the implementation of PPINOT Tool Suite helped assure the completeness of this traceability. On the one hand, the design-time analysis operations implemented by the PPINOT Analyser require full traceability between PPIs and BP elements since their main goal is to identify all of the relationships between them. On the other hand, without such traceability it would have been impossible for PPINOT Instrumenter to automatically instrument the processes in order to obtain the measures that are necessary to calculate PPIs. Furthermore, the traceability provided by the PPINOT metamodel helped solve many issues regarding the definition of PPIs in the three validation scenarios. These issues were caused because in all three organisations the process model and the PPI definitions were two separate documents and, hence, it was really complicated to maintain the coherence across them. In fact, in one of the organisations, the PPIs associated with a process were defined using different terms and even at a different abstraction level than those used in the process. This made it really hard to relate them with the process and, hence, they can hardly be used to improve it.

Finally, the implementation of PPINOT Tool Suite have shown that the PPINOT metamodel is useful to enable the automation of many PPI management tasks. As a matter of fact, PPINOT Tool Suite has been built

[7]Because of privacy reasons these processes and PPIs cannot be included in this paper.

around the PPINOT metamodel so that it provides a common way to represent PPIs that is useful for all of the tools that we have implemented. Both editors create PPI models according to the PPINOT metamodel. This PPI model can be used later on by PPINOT Instrumenter to instrument Activiti by configuring event generators when one of the conditions used by the PPIs in the PPI model take place. PPINOT Reporter receives these events at run-time, calculate the value of the PPIs according to the definition specified in the model and shows these values to the user. Finally, PPINOT Analyser is a proof of our automated design-time analysis approach.

9. Related Work

Performance measurement is a current research field in management science that have gained interest in both academia and business [10]. Many works have been done in the identification and classification of key performance indicators for any company [33] and those relevant for specific domains such as logistics, production, supply chains, etc. (e.g. [34, 35, 36, 37])

Process management is becoming a part of the language and actions of many organisations. Hence, many authors recognize the importance of process orientation of performance management systems [38]. Actually, the Academia of Business Process Management Professionals provides a definition for *process performance measurement*: "the formal planned monitoring of process execution and the tracing of results to determine the effectiveness and efficiency of the process" [39].

There already exists a number of proposals to evaluate the performance of business processes defined in the literature and, in some cases, implemented in products.

Popova et al. present in [10] a framework for modeling performance indicators within a general organisation modeling framework. They define indicators by assigning values to a set of attributes, but they do not point out the way these indicators are calculated. They also define relations between PPIs and the processes, and relationships between PPIs (causality, correlation and aggregation). This work is extended in [40], where they also present formal techniques for the analysis of executions of organizational scenarios. Furthermore, in [18] they establish the connection of performance indicators with goals and discuss analytic issues for consistency and verification checks of the goal structures and between goals and PPIs. Regarding the definition of a scope, they define temporal properties over PPIs (called PI expressions)

in [41], but our proposal is more flexible and allows to define some kind of filters that they do not take into account (*e.g.* a composition of a TimeFilter with a ProcessStateFilter). Moreover, they do not consider derived measures nor offer design-time automated analysis facilities to obtain what we call in this work PPI-BPElements Interaction.

In[42] Mayerl et al. discuss how to derive metric dependency definitions from functional dependencies by applying dependency patterns. To this end, they propose a model that distinguishes between a functional part, where they define dependencies between application, service and process layers (based on concepts of BPEL and WSDL), and another part for metric dependencies, based on concepts of the CIM metrics model [43] and the QoS UML profile described in [44]. They also introduce a mathematical formalism in order to describe dependency functions and the so-called metric characteristics or metrics calculable based on other metric values. Finally they cover the mapping of these models to a monitoring architecture that contains functions to instrument and collect metrics, functions to aggregate and compare metrics with agreed service levels and functions to report SLA compliance and violations. However, they do not delve into the definition of measures, they only set the semantics of some elements to consider when defining measures.

Castellanos et al.'s approach [45] is implemented in the IBOM platform, that allows, among other things, to define business measures and perform intelligent analysis on them to understand causes of undesired values and predict future values. The user can define business measures (through a GUI) to measure characteristics of process instances, processes, resources or of the overall business operations. Specifically, they characterize metrics through four attributes: name (unique), target entity (objet to be measured), data type (numeric, boolean, taxonomy or SLA) and desirable metric values. For the computation logic definition, templates are used. These templates map data and metadata about process executions into numeric and boolean measures. This approach is not focused on business processes but on the whole organisation. Anyway, during the definition of what they call metrics, as far as we can deduce from the paper, they do not take into account some aspects we do, as the scope, the unit of measure, the dimension to be measured. It is not possible to accurately know which is the set of measures than can be defined with this approach.

Momm et al.'s approach [46] consists of a top-down approach for developing an uniform IT support based on SOA in conjunction with the monitoring

aspects required for processing the PPIs. Momm et al build the approach on the principles of the Model Driven Architecture (MDA) to enable the support of different SOA platforms as well as an automated generation of the required instrumentation and monitoring infrastructure. Particularly, they present a metamodel for the specification of the PPI monitoring, an extension of the BPMN metamodel for modeling the required instrumentation for the monitoring, and an outline of methodology for an automated generation of this instrumentation. However, the metamodel for the specification of performance indicators does not consider those related to data or events (PPI6 from our example can not be defined according to this metamodel); and it lacks some properties when defining PPIs like the scope or the function to calculate derived measures. Moreover, the absence of a formal foundation for this PPI definition does not allow an automated analysis of them.

Another work which is close to ours is the one presented by Wetzstein et al. in [9]. This paper introduces a framework for BAM as part of the semantic business process management. The authors describe a KPI ontology using WSML to specify KPIs over semantic business processes. However, our ontology improves this one, since they do not take into account indicators related to data (they can not define PPI6).

The integrated methodology GRAI/GIM [47, 48] explicitly models performance indicators. They establish three parameters or attributes to define performance indicators: name, value domain or dimension and procedure to calculate the value. However this definition is not process-aware and is only made informally and without taking into account the relationships among the performance indicators and between them and the BP elements.

ARIS [49] models key performance indicators and allow for using the Balance Scorecard approach for modelling cause-and-effect relationships and assign KPIs to the strategic objective. Furthermore, in [50], the ARIS Process Performance Manager is described. This tool provides a mechanism to define measurement points over BPs defined using EPCs, as well as the data sources and calculation rules required to calculate PPIs. However this tool, to the best of our knowledge, is not able to define PPIs related to data and is more restrictive regarding the definition of the scope. Furthermore, the definition of PPIs must be done over EPCs, loosing thus the flexibility of our proposal, that can be applied to any business process language.

Pedrinaci et al. [51] describe a Semantic Business Process Monitoring Tool called SENTINEL. This tool can support automated reasoning, though the authors point out that one aspect to be improved is the analysis engines

in order to support deviations. In this paper, they also present a metric ontology to allow the definition and computation of metrics, which take into account many of the aspects we do , for instance the concept of population filter, which is somehow similar to our scope, though they do not delve into the detail of the types of scope that can be defined. However, it is not clear how PPIs can be analysed and queried based on this concept nor it is clear whether it allows an explicit relation between PPIs and the elements of a business process. Furthermore, they deal with runtime analysis, but not design-time.

In Table 6 we establish an explicit comparison between the previously commented approaches for the process performance measurement and PPINOT (our proposal). We highlight those requirements for an appropriate definition of PPIs established in Section 1: Expressiveness (unambiguous and complete definition, taking into account the possibility to define SMART PPIs and the different types of measures and scopes described, feature 1), traceability with BP elements (feature 2), and automated analysis, in the table represented through "Aut An" due to space constraints (feature 3). We do not include the understandability requirement in this comparison since this issue, though commented in the presented implementation and addressed in our other work, is out of the scope of this paper. Furthermore, we also include in the table two additional features we consider important to enable such an appropriate PPI definition and analysis (tooling support, feature 4, and support for BPM standards like BPMN or BPEL, feature 5). We use the following notation: A ✓ sign means that the proposal successfully addresses the issue; a $\sim$ sign indicates that it addresses it partially; N/A means the information is not available; and a blank cell indicates that it does not contemplate the issue. We use ✓* for two features: time measures, when those approaches that addresses this feature do not distinguish between linear and cyclic time measures; and aggregated measures, when those approaches that addresses this feature do not take into account the possibility of grouping by certain property (`isGroupedBy`).

10. Conclusions and Future Work

In this paper we demonstrate that it is possible to provide PPI definitions that are: unambiguous and complete, understandable by non-technical users, traceable with the business process elements and amenable to automated analysis. This is done by presenting PPINOT Metamodel. This mechanism

Proposal	1											2	3		4	5
	1.1	1.2					1.3						3.1	3.2		
		1.2.1	1.2.2	1.2.3	1.2.4	1.2.5	1.2.6	1.3.1	1.3.2	1.3.3						
Popova et al.	✓	✓*	N/A	N/A	N/A	✓*		N/A	✓	∼	✓			∼		
Mayerl et al.		✓*	✓	N/A		✓*	✓				∼					✓
Castellanos et al.		✓*	✓	∼	N/A	✓*	∼				∼				✓	
Momm et al.		✓*	✓			✓*					✓				∼	✓
Wetzstein et al.	✓	✓*	✓	✓		✓*	✓		∼		✓				∼	✓
GRAI/GIM		N/A	N/A	N/A		✓*	N/A									
ARIS	✓	✓	✓	∼		✓	✓	✓	∼	✓	∼				✓	
Pedrinaci et al.	✓	✓*	✓	N/A		✓*	✓	✓	✓	✓	N/A		∼		✓	✓
PPINOT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓

(1.1) SMART PPI
(1.2.1)Time measures
(1.2.2) Count measures
(1.2.3) Condition measures
(1.2.4) Data measures
(1.2.5) Aggregated measures
(1.2.6) Derived measures
(1.3.1) Number of instances scope
(1.3.2) Temporal scope
(1.3.3) Process state scope
(2) Traceability between PPIs and BP models
(3.1) Relationship *measured by*
(3.2) Relationship *involved in*
(4) Tooling support
(5) BPMN standard support

Table 6: Comparison of the analysed approaches and our proposal

to define PPIs is expressive enough to allow the definition of a wide range of PPIs, including all of the PPIs found in the literature and the real scenarios studied; it is also highly flexible since it is independent of the business process model language. Furthermore we have provided an implementation of such metamodel using DL. This formal foundation in the definition of PPIs enables their analysis at design-time in a way that is amenable to automated reasoning. Concretely we have presented a novel analysis operation to automatically derive the dependences between BP elements and PPIs, helping thus to, for instance, assist during the evolution of business processes. In addition we have developed a software prototype comprising a graphic editor, an implementation of the aformentioned analysis operation and a mechanism to extract the information required to calculate PPIs' values from the open source BPMS activiti. The applicability of this proposal has been studied by applying it to several real scenarios.

A number of directions for future work are considered. The extension of the PPINOT metamodel in order to support the specification of PPIs related to (human) resources and of the analysis mechanism will be necessary in order to extract information related to the resources assigned to the BP activities (workload for instance). New analysis operations are being investigated to predict conflicts between PPIs and future behaviour. Another aspect to work in is the research of the concrete activities related to the PPI lifecycle that must be performed along the BPM lifecycle (identifying which of them must be accomplished in each phase of the BPM lifecycle). In this sense, more investigation is needed in the integration of the tool with the execution aspects of the supporting information systems in the context of performance evaluation (although as previousy mentioned, some first steps using the Activiti BPM Platform has been already made).

Acknowledgments

We would like to thank to the Quality Office of the Information Technology Department of the Andalusian Health Service for kindly providing us their internal business process models and its PPIs. We also would like to thank J. Mendling, A. Sharpanskykh, V. Popova, D. Ruiz, C. Pedrinaci, M. Weske, E. Cardoso and Christian Janiesch for their helpful comments in earlier versions of this paper.

Appendix A. PPINOT specification of PPIs for the case study

In this appendix we present the specification of the 9 PPIs contained in Table 1 of our case study - RFC management process (Section 2). We have used an adapted version of the HUTN (Human-Usable Textual Notation) notation presented by the OMG in [52].

```
PPI{
    identifier: PPI1
    name: RFCs cancelled from RFCs registered
    relatedTo: RFCManagement
    goals: Improve customer satisfaction
    definition: DerivedMultiInstanceMeasure{
        function: (a/b)*100
        uses: a.refersTo: AggregatedMeasure {
            scale: float
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: CountMeasure{
                when: TimeInstantCondition{
                    changesToState: cancelled
                    appliesTo: dataObject RFC
                }
            }
        }
        uses: b.refersTo: AggregatedMeasure {
            scale: float
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: CountMeasure{
                when: TimeInstantCondition{
                    changesToState: registered
                    appliesTo: dataObject RFC
                }
            }
        }
    }
    target.upperBound: 4
    scope: ComposedFilter{
        And[
            ProcessStateFilter.processState: finished
            Timefilter.periodicity: Weekly.dayOfWeek: Friday
        ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
}

PPI{
    identifier: PPI2
    name: Average time of committee decision
    relatedTo: RFCManagement
    goals: Reduce RFC time-to-response
    definition:AggregatedMeasure{
        scale: int
        unitOfMeasure: day
```

```
        aggregationFunction: average
        aggregates: TimeMeasure{
            from: TimeInstantCondition{
                changesToState: active
                appliesTo: activity Analyse in Committee
            }
            to: TimeInstantCondition{
                changesToState: completed
                appliesTo: activity Analyse in Committee
            }
        }
    }
    target.upperBound: one day
    scope: ComposedFilter{
        And[
            ProcessStateFilter.processState: finished
            Timefilter.periodicity: Weekly.dayOfWeek: Friday
        ]
    }
    responsible: Planning and Quality Manager
    informed:CIO
}

PPI{
    identifier: PPI3
    name: Corrective RFCs from approved RFCs
    relatedTo: RFCManagement
    goals: Improve customer satisfaction
    definition: DerivedMultiInstanceMeasure{
        function: (a/b)*100
        uses: a.refersTo: AggregatedMeasure {
            scale: int
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: ConditionMeasure{
                meets: DataPropertyCondition{
                    restriction: type of change = corrective
                    statesConsidered: approved
                    appliesTo: dataObject RFC
                }
            }
        }
        uses: b.refersTo: AggregatedMeasure {
            scale: int
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: CounttMeasure{
                when: TimeInstantCondition{
                    chagesToState: triggered
                    appliesTo: event report RFC approved
                }
            }
        }
    }
    target: SimpleTarget.upperBound: 2
    scope: ComposedFilter{
        And[
```

```
            ProcessStateFilter.processState: finished
            Timefilter.periodicity: Weekly.dayOfWeek: Friday
        ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
    comments: values up to 5 \% are reasonable
}

PPI{
    identifier: PPI4
    name: Perfective RFCs from approved RFCs
    relatedTo: RFCManagement
    goals: Improve customer satisfaction
    definition: DerivedMultiInstanceMeasure{
        function: (a/b)*100
        uses: a.refersTo: AggregatedMeasure {
            scale: int
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: ConditionMeasure{
                meets: DataPropertyCondition{
                    restriction: type of change = perfective
                    statesConsidered: registered
                    appliesTo: dataObject RFC
                }
            }
        }
        uses: b.refersTo: AggregatedMeasure {
            scale: int
            unitOfMeasure: RFC
            aggregationFunction: sum
            aggregates: CounttMeasure{
                when: TimeInstantCondition{
                    chagesToState: triggered
                    appliesTo: event report RFC approved
                }
            }
        }
    }
    target: SimpleTarget.upperBound: 4
    scope: ComposedFilter{
        And[
        ProcessStateFilter.processState: finished
        Timefilter.periodicity: Weekly.dayOfWeek: Friday
        ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
}

PPI{
    identifier: PPI5
    name: Average time of RFC analysis
    relatedTo: RFCManagement
    goals: Reduce RFC time-to-response
    definition:AggregatedMeasure{
```

```
        scale: int
        unitOfMeasure: day
        aggregationFunction: average
        aggregates: TimeMeasure{
            from: TimeInstantCondition{
                changesToState: active
                appliesTo: activity Analyse RFC
            }
            to: TimeInstantCondition{
                changesToState: completed
                appliesTo: activity Analyse RFC
            }
        }
    }
    target: simpleTarget.upperBound: two days
    scope: ComposedFilter{
    And[
            LastInstancesFilter.numberOrInstances: 100
            Timefilter.periodicity: Weekly.dayOfWeek: friday
        ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
}

PPI{
    identifier: PPI6
    name: Number of RFCs in analysis
    relatedTo: RFCManagement
    goals: Improve customer satisfaction. Reduce RFC time-to-response
    definition: AggregatedMeasure{
        scale: int
        unitOfMeasure: RFC
        aggregationFunction: sum
        aggregates: ConditionMeasure{
            meets: StateCondition{
                state: active
                appliesTo: activity analyse RFC
            }
        }
    }
    target: 2 RFCs
    scope: ComposedFilter{
    And[
            ProcessStateFilter.processState: active
            Timefilter.periodicity: Weekly.dayOfWeek: friday
        ]
    }

    responsible: Planning and Quality Manager
    informed: CIO
}

PPI{
    identifier: PPI7
    name: Number of RFCs per type of change
    relatedTo: RFCManagement
```

```
    goals:
    definition: AggregatedMeasure{
       scale: map
       unitOfMeasure: RFC
       aggregationFunction: sum
       aggregates: CountMeasure{
          when: TimeInstantCondition{
             changesToState: registered
             appliesTo: dataObject RFC
          }
       }
       isGroupedBy: type of change
    }
    target: ComposedTarget[
       corrective-20 RFCs
       evolutive-30 RFCs
       perfective-20 RFCs
    ]
    scope: ComposedFilter{
       And[
          ProcessStateFilter.processState: finished
          Timefilter.periodicity: Monthly.dayOfMonth: 25
       ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
    comments: the ideal situation is that corrective RFCs tend to zero
}

PPI{
    identifier: PPI8
    name: Number of RFCs per project
    relatedTo: RFCManagement
    goals:
    AggregatedMeasure{
       scale: map
       unitOfMeasure: RFC
       aggregationFunction: sum
       aggregates: CountMeasure{
          when: TimeInstantCondition{
             changesToState: registered
             appliesTo: dataObject RFC
          }
       }
       isGroupedBy: project
    }
    target: ComposedTarget[
       RR.HH-50 RFCs
       Diraya-60 RFCs
       Pharma-1 RFCs
    ]
    scope: ComposedFilter{
       And[
          ProcessStateFilter.processState: finished
          Timefilter.periodicity: Monthly.dayOfMonth: 25
       ]
    }
```

```
    responsible: Planning and Quality Manager
    informed: CIO
}

PPI{
    identifier: PPI9
    name: Average lifetime of an RFC
    relatedTo: RFCManagement
    goals: Reduce RFC time-to-response
    definition:AggregatedMeasure{
        scale: float
        unitOfMeasure: day
        aggregationFunction: average
        samplingFrequency:
        aggregates: TimeMeasure{
            from: TimeInstantCondition{
                changesToState: triggered
                appliesTo: event Receive RFC
            }
            to: TimeInstanceCondition{
                changesToState: completed
                appliesTo:pool RFC management
            }
        }
    }
    target: 3
    scope: ComposedFilter{
        And[
            ProcessStateFilter.processState: finished
            Timefilter.periodicity: Monthly.dayOfMonth: 25
        ]
    }
    responsible: Planning and Quality Manager
    informed: CIO
}
```

Appendix B. OCL Constraints

This section provides the complete set of OCL invariants for the PPINOT metamodel presented in Section 3

OCL Constraint 1 `AggregatedMeasures` can only aggregate single-instance measures, that is a `BaseMeasure` or a `DerivedSingleInstanceMeasure`.

```
context AggregatedMeasure inv:
self.aggregates.oclIsTypeOf(BaseMeasure) or
self.aggregates.oclIsTypeOf(DerivedSingleInstanceMeasure)
```

OCL Constraint 2 `DerivedSingleInstanceMeasures` are calculated by using single-instance measures in their mathematical functions, i.e.,

they can only be calculated by using `BaseMeasures` and/or `DerivedSingleInstanceMeasures`.

```
context DerivedSingleInstanceMeasure inv:
self.uses->forall (v: Variable |
v.refersTo.oclIsTypeOf(BaseMeasure) or
v.refersTo.oclIsTypeOf(DerivedSingleInstanceMeasure))
```

OCL Constraint 3 `DerivedMultiInstanceMeasures` are calculated by using multi-instance measures in their mathematical functions, i.e., they can only be calculated by using `AggregatedMeasures` and/or `DerivedMultiInstanceMeasures`.

```
context DerivedMultiInstanceMeasure inv:
self.uses->forall (v: Variable |
v.refersTo.oclIsTypeOf(AggregatedMeasure) or
v.refersTo.oclIsTypeOf(DerivedMultiInstanceMeasure))
```

OCL Constraint 4 `AggregatedMeasures` that aggregates `ConditionMeasures` can only use the aggregation function sum.

```
context AggregatedMeasure inv:
self.aggregates.oclIsTypeOf(ConditionMeasure) implies
self.aggregationFunction = AggregationFunction::sum
```

OCL Constraint 5 The BPElements to which a `DataPropertyCondition` can be applied in BPMN are: dataObjects.

```
context DataPropertyCondition inv:
self.appliesTo.oclIsTypeOf(DataObject)
```

Appendix C. OCL Constraints for BPMN binding

The following constraints are defined for the case where BPMN is used to define business processes, defining thus the specific bp elements and their states.

OCL Constraint 6 The BPElements to which a `TimeInstantCondition` can be applied in BPMN are: activity, pool, event and dataObjects.

```
context TimeInstantCondition inv:
self.appliesTo.oclIsTypeOf(Activity) or
self.appliesTo.oclIsTypeOf(Pool) or
self.appliesTo.oclIsTypeOf(Event) or
self.appliesTo.oclIsTypeOf(DataObject)
```

OCL Constraints 7-8 Depending on the type of bp elements, the possible states that are considered for `tTimeInstantConditions` are specified in the two following constraints. The states of dataObjects, as stated in Section 3, are defined by the user in the BP diagram.

```
context TimeInstantCondition inv:
self.appliesTo.oclIsTypeOf(Event) implies self.changesToState = ‘‘triggered’’

context TimeInstantCondition inv:
self.appliesTo.oclIsTypeOf(Activity) implies
(self.changesToState = ‘‘ready’’  or
self.changesToState = ‘‘active’’  or
self.changesToState = ‘‘withdrawn’’  or
self.changesToState = ‘‘completing’’ or
self.changesToState = ‘‘completed’’ or
self.changesToState = ‘‘failing’’ or
self.changesToState = ‘‘failed’’ or
self.changesToState = ‘‘terminating’’ or
self.changesToState = ‘‘terminated’’ or
self.changesToState = ‘‘compensating’’ or
self.changesToState = ‘‘compensated’’)
```

OCL Constraint 9 The BPElements to which a `StateCondition` can be applied in BPMN are: activity, pool, event and dataObjects.

```
context StateCondition inv:
self.appliesTo.oclIsTypeOf(Activity) or
self.appliesTo.oclIsTypeOf(Pool) or
self.appliesTo.oclIsTypeOf(Event) or
self.appliesTo.oclIsTypeOf(DataObject)
```

OCL Constraints 10-11 Depending on the type of bp elements, the possible states that are considered for t`StateConditions` are specified in the two following constraints. The states of dataObjects, as stated in Section 3, are defined by the user in the BP diagram.

```
context StateCondition inv:
self.appliesTo.oclIsTypeOf(Event) implies self.state = ‘‘triggered’’

context StateCondition inv:
self.appliesTo.oclIsTypeOf(Activity) implies
(self.sToState = ‘‘ready’’  or
self.state = ‘‘active’’  or
self.state = ‘‘withdrawn’’  or
self.state = ‘‘completing’’ or
self.state = ‘‘completed’’ or
self.state = ‘‘failing’’ or
self.state = ‘‘failed’’ or
self.state = ‘‘terminating’’ or
self.state = ‘‘terminated’’ or
self.state = ‘‘compensating’’ or
self.state = ‘‘compensated’’)
```


[1] W. M. van der Aalst, A. H. ter Hofstede, M. Weske, Business process management: A survey, in: Business Process management, volume 2678, Springer, 2003, pp. 1–12.

[2] G. D. Alexander Grosskopf, M. Weske, The Process. Busisness Process Modeling using BPMN, Megan-kiffer Press, Mar 2009.

[3] G. Chase, A. Rosenberg, R. Omar, J. Taylor, M. Rosing, Applying Real-World BPM in an SAP Environment, SAP Press, Galileo Press, Incorporated, 2011.

[4] F. Franceschini, M. Galetto, D. Maisano, Management by Measurement: Designing Key Indicators and Performance Measurement Systems, Springer Verlag, 2007.

[5] A. del Río-Ortega, M. Resinas, A. Ruiz-Cortés, Defining process performance indicators: An ontological approach, in: Proceedings of the 18th International Conference on Cooperative Information Systems (CoopIS). OTM 2010, Part I, pp. 555–572.

[6] A. del Río-Ortega, M. Resinas, A. Ruiz-Cortés, PPI Definition and Automated Design-Time Analysis, Technical Report, Applied Software Engineering Research Group, 2012.

[7] Object Management Group (OMG), Business process modeling notation (BPMN) version 1.2, 2009.

[8] OASIS, Web services business process execution language (BPEL) version 2.0, oasisstandard. published via internet, 2007.

[9] B. Wetzstein, Z. Ma, F. Leymann, Towards measuring key performance indicators of semantic business processes, in: BIS, pp. 227–238.

[10] V. Popova, A. Sharpanskykh, Modeling organizational performance indicators, Inf. Syst. 35 (2010) 505–527.

[11] A. del Río-Ortega, M. Resinas, A. Durán, A. Ruiz-Cortés, Defining process performance indicators by using templates and patterns, in: Business Process Management (BPM), pp. 223–228.

[12] H. Dresner, Business activity monitoring: BAM architecture, 2003.

[13] W. van den Heuvel, Survey on Business Process Management, Technical Report, 2008.

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

[15] W. M. P. van der Aalst, M. Pesic, M. Song, Beyond process mining: From the past to present and future, in: B. Pernici (Ed.), CAiSE, volume 6051 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 38–52.

[16] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, M.-C. Shan, Business process intelligence, Computers in Industry 53 (2004) 321–343.

[17] A. Shahin, M. A. Mahbod, Prioritization of key performance indicators: An integration of analytical hierarchy process and goal setting, International Journal of Productivity and Performance Management 56 (2007) 226 – 240.

[18] V. Popova, A. Sharpanskykh, Formal modelling of organisational goals based on performance indicators, Data Knowl. Eng. 70 (2011) 335–364.

[19] J. M. García, D. Ruiz, A. Ruiz-Cortés, A model of user preferences for semantic services discovery and ranking, in: L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), ESWC (2), volume 6089 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 1–14.

[20] C. Cabanillas, M. Resinas, A. Ruiz-Cortés, Defining and analysing resource assignments in business processes with ral, in: G. Kappel, Z. Maamar, H. R. M. Nezhad (Eds.), ICSOC, volume 7084 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 477–486.

[21] C. Cabanillas, M. Resinas, A. R. Cortés, Ral: A high-level user-oriented resource assignment language for business processes, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops (1), volume 99 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 50–61.

[22] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Inf. Syst. 35 (2010) 615–636.

[23] D. Harel, B. Rumpe, Meaningful modeling: What's the semantics of "semantics"?, IEEE Computer 37 (2004) 64–72.

[24] J. Rivera, E. Guerra, J. de Lara, A. Vallecillo, Analyzing rule-based behavioral semantics of visual modeling languages with Maude, in: Software Language Engineering 2008, volume 5452/2009 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 54–73.

[25] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[26] Racer Systems, RACER: Renamed ABox and Concept Expression Reasoner, http://www.racer-systems.com, 2011. [Online; accessed 30-July-2012].

[27] Knowledge Representation and Reasoning Group, Hermit OWL reasoner. the new kid on the OWL block, http://hermit-reasoner.com, 2011. [Online; accessed 30-July-2012].

[28] Clark & Parsia, Pellet OWL 2 reasoner for java, http://clarkparsia.com/pellet, 2011. [Online; accessed 30-July-2012].

[29] D. Nardi, R. J. Brachman, An introduction to description logics, in: Description Logic Handbook, pp. 1–40.

[30] M. Bhatt, W. Rahayu, S. P. Soni, C. Wouters, Ontology driven semantic profiling and retrieval in medical information systems, Web Semantics: Science, Services and Agents on the World Wide Web 7 (2009) 317 – 331.

[31] G. Decker, H. Overdick, M. Weske, Oryx - an open modeling platform for the bpm community, in: BPM, pp. 382–385.

[32] O. O. of Government Commerce), Information technology infrastructure library (ITIL) v3, Collection of books, 2007.

[33] R. S. Kaplan, D. P. Norton, The balanced scorecard: Measures that drive performance, Harvard Business Review January-February 1992 (1992) 71–79.

[34] P. Brewer, T. Speh, Using the balance scorecard to measure supply chain performance, Journal of Business Logistics 21 (2000) 75–93.

[35] F. Chan, Performance measurement in a supply chain, International Journal of Advanced Manufacturing Technology 21 (2003) 534–548.

[36] E. Krauth, H. Moonen, V. Popova, M. C. Schut, Performance measurement and control in logistics service providing, in: ICEIS (2), pp. 239–247.

[37] G. Vaidyanathan, A framework for evaluating third-party logisics, ACM 48 (2005) 89–94.

[38] M. Benner, M. L. Tushman, Exploitation, exploration and process management: the productivity dilemma revisited, Academy and Management Review 28 (2003) 238–256.

[39] A. of Business Process Management Professionals (ABPMP), Guide to the business process management common body of knowledge, 2009.

[40] V. Popova, A. Sharpanskykh, Formal analysis of executions of organizational scenarios based on process-oriented specifications, Applied Intelligence (2009).

[41] V. Popova, A. Sharpanskykh, Formal goal-based modeling of organizations, in: MSVVEIS, pp. 19–28.

[42] C. Mayerl, K. Hner, J.-U. Gaspar, C. Momm, S. Abeck, Definition of metric dependencies for monitoring the impact of quality of services on quality of processes, in: Second IEEE/IFIP International Workshop on Business-driven IT Management (Munich), pp. 1–10.

[43] Distributed Management Task Force (DMTF), Common information model (CIM) metrics model, 2003.

[44] Object Management Group (OMG), UMLTM profile for modeling quality of service and fault tolerance characteristics and mechanisms specification, 2006.

[45] M. Castellanos, F. Casati, M.-C. Shan, U. Dayal, ibom: a platform for intelligent business operation management, in: Proceedings. 21st International Conference on Data Engineering, 2005., Hewlett-Packard Laboratories, pp. 1084– 1095.

[46] C. Momm, R. Malec, S. Abeck, Towards a model-driven development of monitored processes, in: Wirtschaftsinformatik (2), pp. 319–336.

[47] D. Chen, B. Vallespir, G. Doumeingts, Grai integrated methodology and its mapping onto generic enterprise reference architecture and methology, Computers in Industry 33 (1997) 387–394.

[48] G. Doumeingts, B. Vallespir, D. Chen, Handbook on Architectures of Information Systems, Springer-Verlag, pp. 313–338.

[49] R. Davis, E. Brabnder, Aris design platform, Springer, 2007.

[50] A. Scheer, W. Jost, H. Heß, A. Kronz, Corporate Performance Management: Aris in Practice, Springer, 2006.

[51] C. Pedrinaci, D. Lambert, B. Wetzstein, T. van Lessen, L. Cekov, M. Dimitrov, Sentinel: a semantic business process monitoring tool, in: OBI, p. 1.

[52] Object Management Group (OMG), Human-usable textual notation (HUTN) specification, 2004.

We developed a numerical model to find the output solution of an APM fiber laser of two-coupled linear cavities.
Numerical results are analyzed for different cavities lengths.
Experimentally we found a relation between the pulse repetition frequency and the cavities lengths.
Dependence of the time pulse width with the repetition frequency was confirmed for an reflectivity value.
We found a very good correlation between experimental and simulations results.