A Flexible Semantic KPI Measurement System

Kyriakos Kritikos¹, Dimitris Plexousakis¹, and Robert Woitsch²

¹ ICS-FORTH
Heraklion GR-70013, Greece
{kritikos, dp}@ics.forth.gr

² BOC, Austria
Robert.Woitsch@boc-eu.com

Abstract. Linked Data (LD) technology enables integrating information across disparate sources and can be exploited to perform inferencing for deriving added-value knowledge. As such, it can really support performing different kinds of analysis tasks over business process (BP) execution related information. When moving BPs in the cloud, giving rise to Business Process as a Service (BPaaS) concept, the first main challenge is to collect and link, based on a certain structure, information originating from different systems. To this end, two main ontologies are proposed in this paper to enable this structuring: a KPI and a Dependency one. Then, via exploiting these well-connected ontologies, an innovative Key Performance Indicator (KPI) analysis system is built that offers two main analysis capabilities: KPI assessment and drill-down, where the second can enable finding root causes of KPI violations. This system advances the state-of-the-art by exhibiting the capability, through the LD usage, of the flexible construction and assessment of any KPI kind, allowing experts to better explore the possible KPI space.

1 Introduction

Organisations formulate and realise both internal and external procedures in the form of business processes (BPs) in order to provide support or enable their core business. In this respect, corresponding process-aware information systems (PAIS) are embraced that enable the execution and management of these BPs to facilitate the delivery of core services and products. The traditional BP management lifecycle [13] comprises the four main activities of BP design, allocation, execution and evaluation. The first three activities focus on bridging the well-known business-to-IT gap and supporting BP execution. The evaluation activity supports deriving business intelligence (BI) information via conducting various analysis tasks to facilitate BP improvement, thus closing the BP lifecycle loop.

Key Performance Indicator (KPI) measurement and assessment is a well-studied BP evaluation task in the literature. A KPI specifies a condition over a BP quality metric, thus defining the minimum respective quality level to be sustained. The involved metric supplies all measurement details needed to measure a certain BP quality attribute, which can be categorised into 4 groups: (a) time, (b) quality, (c) customer satisfaction, and (d) financial [1]. As such, an
evaluation expert has the main goal to specify suitable KPIs, possibly spanning all four groups, which can be measured by the BP evaluation system and enable assessing the quality levels of BPs.

In this respect, various KPI measurement systems have been proposed which use different technologies, such as OLAP [3] or SQL query evaluation [2]. While such systems can rapidly perform the KPI assessment, we believe that their main goal should not be the KPI assessment speed but the assistance to evaluation experts in defining the most suitable KPIs for a BP. However, there is a lack of flexible and user-intuitive mechanisms for KPI definition in these systems. Such systems are also usually special-purposed as they are designed to serve only a fix set of KPI metric types, such that introducing a new metric can require re-engineering the underlying system database. Finally, they do not exploit sophisticated integration mechanisms to integrate any information source kind. Such an integration is essential to integrate information not only from internal information sources within a BP management system (e.g., from BP monitoring and execution components) but also external ones that can enable deriving suitable, added-value facts. For instance, a location ontology could certainly assist in deriving non-obvious location relations between KPI assessment facts such that, e.g., the deployment of a BP component in a certain cloud region leads constantly to KPI violations, thus enabling to derive of useful deployment facts.

The latter issue is critical in the context of not only traditional but also BPs that have been moved to the Cloud, i.e., BP as a Service (BPaaS). Such a migration has become a trend nowadays due to the great advantages that cloud computing offers, such as reduced cost and elasticity. As such, this migration must be suitably supported. This support can be realised via a BPaaS management system, able to control the whole BPaaS lifecycle, which comprises different environments, each responsible for a different lifecycle activity [30]. This inevitably well justifies the need to integrate information from many of these environments and their components to support the BPaaS monitoring and evaluation.

To realise the vision of a BPaaS by providing essential, flexible and user-intuitive support to BPaaS evaluation, this paper advocates the use of Linked Data (LD) technology for the following reasons: (a) it allows performing inferencing tasks to deduce added-value analysis information; (b) enables integrating information, even in unforeseen ways, across disparate information sources; (c) LD are represented via ontologies which are closer to human perception.

The information integration task is facilitated via introducing two ontologies: (a) a dependency ontology that captures the dependencies between BPaaS components, across different abstraction levels (BP, software and infrastructure), and their state. This ontology constitutes the major integration point for information coming from different systems, enabling its suitable correlation for supporting KPI analysis; (b) a KPI-based extension of the OWL-Q [16] ontology which enables formally and fully specifying how KPIs can be measured over which BPaaS hierarchy components. As such, via introducing KPI metric hierarchies that span the whole BPaaS hierarchy, the measurability of KPIs is guaranteed.
An innovative KPI measurement system has been built [18] over these two ontologies offering its facilities in the form of a REST service. This system can integrate information from many parts of a BPaaS management system and offers two KPI analysis capabilities: KPI measurement and drill-down. The KPI drill-down capability relies on relating via parent-child relations different KPIs at both business and technical levels which enables performing root cause analysis over a high-level KPI violation. This capability comes into two flavours depending on which KPI elements are related: the KPI itself or its metric. The proposed measurement system supports the on-the-fly KPI metric formula specification and assessment, provided that the relation of the formula to a certain context is given. This highlights the great flexibility in KPI measurement offered that greatly assists in the best possible exploration of the KPI metric space.

The rest of this paper is structured as follows. Section 2 reviews the related work. Section 3 offers background information enabling to better understand the main paper contribution. The two proposed ontologies are analysed in Section 4. Section 5 analyses the proposed system architecture and explains the way KPI analysis is performed. Finally, Section 6 concludes the paper and draws directions for further research.

2 Related Work

Work related to this paper spans KPI & dependency modelling and KPI analysis which is analysed in 3 respective sub-sections.

2.1 KPI Meta-Models

As KPI modelling is a pre-requisite for KPI assessment, a great amount of research was devoted in developing KPI meta-models, languages and ontologies, especially as currently there is no standardised BP language covering the BP context perspective (including goal-based and measurement information aspects) [20].

The related work evaluation in KPI modelling relies on a systematic approach which: (a) considers a comparison criteria set, (b) summarises the comparison based on these criteria in the form of an evaluation table, where rows map to the related work approaches, columns to the criteria and cells to the performance of an approach over a certain criterion, and (c) supplies a discussion over the evaluation results presented.

The following comparison criteria were considered: (a) KPI coverage: how well the notion of a KPI is covered; (b) metric formulas: computation formulas are provided to support the KPI metric measurement; (c) measurability: other aspects complementing metric specification are expressed to cover all measurement details (e.g., units, measured objects); (d) goal coverage: KPIs are connected to goals to enable assessing operational or even tactical goals satisfaction via performing goal analysis; (e) semantics: the meta-model / language should
be semantic or allow semantic annotations to enable formal reasoning and reaching better evaluation accuracy levels; (f) information sources: both internal and external information sources should be exploited; (g) measurement origin: the coverage of measurements and their origin (probes, sensors, or humans); (h) level: the levels covered (BP, SE - service, Inf - Infrastructure).

Table 1: Comparison table [18] over KPI modelling work.

| Work  | KPI Cov. | Metric Formulas | Measur. Goal Cov. | Semantics | Inf. Sources | Meas. Origin | Level |
|-------|----------|-----------------|-------------------|-----------|--------------|-------------|-------|
| [27]  | moderate | yes             | moderate          | no        | no           | internal probes | BP, SE |
| [22]  | low      | no              | low               | yes       | no           | internal     | BP    |
| [23]  | good     | yes             | low               | yes       | no           | internal probes | BP    |
| [9]   | good     | yes             | moderate          | no        | no           | internal probes | BP    |
| [8]   | moderate | no              | low               | yes       | no           | internal probes | BP, Inf |
| [10]  | low      | yes             | low               | no        | no           | internal probes | BP    |
| [24]  | moderate | yes             | good              | yes       | yes          | internal probes | BP    |
| [4]   | low      | no              | low               | no        | no           | internal probes | BP    |
| [21]  | low      | yes             | low               | no        | no           | both probes    | BP, Inf |
| OWL-Q KPI Extension | good | yes | excellent | yes | yes | both | all | BP, SE, Inf |

The table evaluation results [18] show that not only our ontology scores well over all criteria but also exhibits the best performance for almost all of them. Thus, it can be considered as the most prominent. [24] maps to the sole modelling work close to ours. However, that work does not cover all levels and correlate measurements to human sources, it exploits only internal information sources and supplies a moderate KPI coverage. It also does not directly model the notion of a metric but internmixes it with that of an indicator. This is wrong as when the latter notion is re-used in the context of KPI computation formulas, it maps to a metric condition and not to the metric itself which represents all suitable measurement details to enable KPI computation. The metric formula definition in that work, though, is interesting as it involves a restricted natural language form. This might be more user-intuitive but loses on clarity and comprehension when recursive composite metric formulas must be specified. A pure mathematical form might be more suitable, an issue that we currently explore.

2.2 Dependency Meta-Models

Dependency modelling is a pre-requisite for system monitoring and adaptation. Without dependency knowledge, both monitoring can be limited, covering low abstraction levels as propagation to higher levels is prohibited, as well as respective adaptation capabilities.

By following the same analysis approach as in the previous sub-section, the following evaluation criteria have been devised: (a) abstraction level: the levels
(denoted as BE, SE, Inf) in the BPaaS hierarchy covered; (b) formalism: the dependency model formalism used; (c) runtime: the capability to cover a dynamic or just a static system view. Dynamic views enable to record system evolution and enable realising monitoring and adaptation mechanisms; (d) detail level: how well component dependencies are expressed.

The dependency modelling work, apart from ours, encoded in the table [18] is the following: (a) SEE [26], (b) GRU [11], (c) CUI [5], (d) HASS [12], (e) TOSCA\(^3\) and (f) CAMEL\(^4\).

Table 2: Evaluation table [18] over dependency modelling work.

| Work  | Abst. Level | Formal. | Runtime | Detail Level |
|-------|-------------|---------|---------|--------------|
| SEE   | BP, SE      | ontology| no      | good         |
| GRU   | SE, INF     | graph   | yes     | low          |
| CUI   | SE, INF     | graph   | yes     | mod.         |
| HASS  | SE, INF     | graph   | yes     | good         |
| TOSCA | SE, INF     | DSL     | no      | good         |
| CAMEL | SE, INF     | DSL     | yes     | good         |
| Ours  | all         | ontology| yes     | good         |

The evaluation results show that our ontology covers all possible levels, covers runtime information and includes a good detail level for the dependencies captured. Thus, it is better than all other work. Sole competitors are approaches in [25,26] that do not cover all BPaaS hierarchy levels. Moreover, the approach in [26] does not capture runtime information, while [25] does not rely on semantics. Please also consider that: (a) an ontology-based approach is essential for better integrating dependency information from various information sources as well as enabling interesting inferencing over this information; (b) some modelling approaches must go beyond the good dependency detail level that they exhibit.

2.3 KPI Analysis Systems

The KPI analysis frameworks proposed employ techniques that mainly support KPI evaluation but also KPI drill-down in some cases. Most techniques employ relational or semantic dbs or data warehouses in order to appropriately structure the underlying database to support KPI analysis.

By following the same evaluation approach as in previous subsections, the following evaluation criteria have been devised: (a) analysis types: the KPI analysis kinds supported; (b) db type: type of db used to store the information relevant for KPI analysis; (c) evaluation technique: the KPI measurement technique used;

\(^3\) http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html
\(^4\) www.camel-dsl.org
(d) **drill-down technique**: the KPI drill-down technique used; (e) **evaluation flexibility**: the level of flexibility offered in the exploration of the possible metric space; (f) **level**: the BPaaS hierarchy levels covered.

### Table 3: Evaluation table [18] over KPI analysis work.

| Work | Analysis Types | DB Type | Evaluation Technique | Drill-Down Technique | Evaluation Flexibility | Level |
|------|----------------|---------|----------------------|----------------------|------------------------|-------|
| [2]  | all            | relational | SQL queries         | decision trees       | low                    | BP    |
| [28] | all            | relational | formula comp.       | decision trees       | low                    | BP, SE|
| [4]  | evaluation     | semantic   | formula comp.       | -                    | low                    | BP    |
| [29] | evaluation     | semantic   | WSML rules          | -                    | moderate               | BP, SE|
| [3]  | evaluation     | warehouse  | OLAP                | -                    | moderate               | BP    |
| [7]  | all            | semantic   | SPARQL queries      | KPI-based            | moderate               | BP    |
| Our Framework | all      | semantic   | SPARQL queries      | metric/KPI-based     | good                   | all   |

The evaluation results in Table 3 [18] show that semantic dbs are considered in more than half of the systems, indicating that their added-value is recognised: they better link information and enable various forms of reasoning. Almost half of the systems focus only on KPI evaluation, while the rest support KPI drill-down via two main techniques: decision trees and combination of metric & KPI hierarchies. The first technique is suitable for covering measurability gaps (disconnected metric trees). The second is suitable when measurability gaps do not exist and KPI hierarchies are formed, such that we can go down to more technical KPIs and then continue from there by exploring the respective metric hierarchies involved for finding the root causes of the high-level KPI violations considered.

Evaluation techniques greatly vary, from SQL queries, OLAP and event-based metric formula calculation to WSML rules and SPARQL queries. SPARQL queries, however, can be more expressive, even with respect to semantic rules, as they: (a) allow different ways to link the underlying semantic information; (b) have similar grouping and aggregation capabilities with SQL queries; (c) operate on the conceptual level which is closer to actual human perception.

Our system seems to be one step ahead in evaluation flexibility from the work in [3, 7, 29] as it does not only allow to map human-based formulations of metric formulas into SPARQL queries but also to experiment with the metric and condition context. Via combining this with the respective KPI ontology capabilities, it can also support exploiting various information sources, like metrics and service properties to enable better exploring the metric space. As such, our approach is more complete and user-intuitive than the other two systems.

Finally, the evaluation results show that only our system is able to cover all levels. In fact only three out of seven systems recognise the need to cover more than one BPaaS hierarchy level.
3 Background

As OWL-Q is the basis for the KPI ontology proposed, it is shortly analysed in this section. OWL-Q is a prominent non-functional service specification ontology-based language that captures all necessary measurability aspects via the introduction of corresponding OWL-Q facets. Semantic rules also accompany OWL-Q to enable two semantic reasoning types: (a) semantic OWL-Q model validation based on the domain semantics; (b) added-value knowledge generation in the form of term equivalence facts. The 6 main facets of OWL-Q are now analysed by focusing more on those more relevant to this paper’s work.

The core facet captures generic concepts and properties, such as Schedule and name. Category is one important generic concept, enabling to construct hierarchies of categories, i.e., partitions of this and other element types, like quality metrics and attributes. As such, this concept facilitates specifying structured quality models (see KPI categories in Section 1) that can be re-used in the context of non-functional capability and KPI description.

The attribute, unit and value type facets capture respective attribute, unit and value type elements. Attributes (e.g., utilisation) represent properties measurable by metrics. Units can be derived (e.g., bytes/sec), single (e.g., sec) or dimensionless (e.g., percentage). Different value types can be specified spanning both non-numeric (e.g., lists) and numeric (e.g., ranges) constructs. Such value types express the domain of values for metrics. As such, they can be used for measurement or metric condition threshold validation by checking whether such values are included in them.

The metric facet specifies how attributes can be measured via the conceptualisation of the Metric concept. A metric can be raw (e.g., uptime), computed from sensors or measurement directives posed over service instrumentation systems, or composite (e.g., availability), computed from formulas, i.e., function applications over a list of arguments, where an argument can be a metric, attribute, service property or another formula. Any kind of metric can be related to a respective context which details its measurement frequency and window.

The specification facet enables expressing non-functional specifications as sets of respective capabilities/requirements. Each capability/requirement is expressed as a constraint which can be either a logical combination of other constraints (i.e., a CompositeConstraint) or a simple condition imposing a certain threshold over a metric (i.e., a SimpleConstraint). A metric condition is related with two different contexts: the metric and condition ones. The condition context explicated which object (e.g., service or service input) is measured and the way the condition should be evaluated over this object’s instances. In the latter case, it is expressed whether the measurements over all or a certain amount or percentage of object instances should be considered in the condition evaluation.

4 KPI and Dependency Ontologies

To enable conducting any KPI analysis kind, meta-models must be supplied that structure and link respective information on which the analysis relies. As such,
to support BPaaS KPI analysis, 2 main ontologies have been developed: (a) the
dependency ontology covering BPaaS dependency models; (b) the KPI ontology
covering KPI modelling. These ontologies are interconnected in one major point,
the actual BPaaS element measured within the BPaaS hierarchy. In this way,
based on the BPaaS dependency model and interconnections between different
BPaaS elements, measurement propagation hierarchies to cover measurability
gaps can be deduced while root causes for KPI violation can be discovered. In
the following, these two ontologies are analysed in separate sub-sections.

4.1 KPI Ontology
It has been decided to extend OWL-Q to cover the modelling of KPIs as it com-
pletely covers the specification of QoS profiles and SLAs. The OWL-Q extension
developed builds upon OWL-Q constructs only a minimum but sufficient num-
ber of relevant new parts. It does not only involve extending the core OWL-Q
ontology but also the rule set that has been originally specified for OWL-Q.
The latter extension kind involved the development of validation and knowledge
production rules which apply for the KPI domain and enable both the semantic
validation of OWL-Q KPI models (to detect, e.g., that a parent KPI is measured
by a metric which is not a parent of the metric used for the evaluation of a child
KPI) but also the production of new knowledge based on them (e.g., the kind of
KPI violation for a certain KPI measurement based on the measurement value).

The current application of the overall OWL-Q extension over the Cloud-
Socket project\(^5\) use cases has led to the full modelling of all KPIs needed. This
is a major evaluation step which validates the design of this OWL-Q extension.

Figure 1 [18] depicts the KPI OWL-Q extension, where the grey colour de-
notes core OWL-Q concepts, blue metric-related concepts, green specification-
related concepts, and yellow a concept from the Dependency ontology, while red
denotes the KPI extension concepts mapping to a sub-facet of the specification
one.

A KPI represents an indicator signifying whether BP performance is satisfac-
tory, problematic or erroneous. Thus, it maps to 3 possible states, captured by a
warning and violation threshold. The BP performance for positively monotonic
metrics is satisfactory when is above the warning threshold, problematic when
is between the warning and violation ones, and erroneous when is below the
violation threshold. In case of negatively monotonic metrics, the order between
warning and violation thresholds is reversed and the state mapping is symmetric.

A KPI was modelled as a sub-concept of simple constraint which apart from
the existing reference to a metric and (violation) threshold, it carries extra in-
formation spanning a human-oriented description (for human consumption), a
validity period and the warning threshold.

OWL-Q enables the modelling of preference models which can represent the
actual way measurements can be propagated from the lowest to the highest level
by also providing weights to each node in the hierarchy signifying its relative

---

\(^5\) http://www.cloudsocket.eu
Towards Semantic KPI Measurement

importance and contribution to the higher-level quality of the parent node. Such weighting as well as the content of such preference models can be BPaaS or customer-specific and thus might be subjective. This also signifies that weights can be modified as suited at evaluation time to represent the change of (broker/customer) opinion or any kind of initial misjudgement.

In other KPI meta-models in the literature, weights are given to KPIs and not metrics. This is an alternative modelling for representing this nested metric structures. However, such a modelling caters for an ad hoc propagation (e.g., possibly with the on-the-fly grouping of KPIs in order to produce a certain higher-level quality value) of quality and not for a generic one.

While OWL-Q fully covers the specification of metrics, it was extended to address the issue of external information access via incorporating such information in metric formulas. By realistically assuming that all modern information sources are available in the form of REST APIs or database endpoints, this ex-
tension was implemented by introducing the *Query* and *APICall* as sub-concepts of *Argument*, enabling instances of these classes to be directly used as input arguments in metric formulas. A *Query* expresses in an implementation-independent way the information required to connect and query a db spanning: (a) the db connection URL; (b) the query language; (c) the actual query; (d) the db type.

On the other hand, an *APICall* expresses all information needed to call a REST API and retrieve back the result, spanning: (a) the API URL; (b) values to all API input parameters for the call; (c) input information encoding; (d) output format (e.g., XML or JSON); (e) a JSON or XML-like script (e.g., in XPath) to operate over the output returned so as to return a single value focusing on a certain part within the retrieved output (e.g., the post-code for an email address).

In certain cases (e.g., *customer satisfaction* metrics) it is imperative to enable humans to manually provide measurements in the system. In this case, the measurement-to-user linkage should be modelled. When connected to certain aspects like human trust and reliability, such linkage can enable reasoning over measurements and their propagation to establish a so-called trust level over them and a more reliable and accurate way to aggregate them. This was realised in OWL-Q by not only associating a measurement to a specific sensor or directive but also to a human resource that might produce this measurement. Such a human resource could also be part of the BPaaS dependency hierarchy (e.g., the knowledge worker responsible for the execution of a certain BPaaS user task) which then makes this extension another connection point with the Dependency ontology proposed.

The drill-down from higher- to lower-level KPIs to support root-cause analysis is enabled by associating KPIs to each other via a parent-child relation. This relation must conform to the parent-child relation between the metrics of the involved KPIs (i.e., the parent KPI’s metric should be a parent metric of the child KPI metric). For instance, a KPI for service response time could be related to KPIs mapping to the service execution time and network latency.

While such relations enable us to go down until low-level KPIs which could be blamed for a high-level KPI violation, the actual root cause of such a violation may not be clearly identified. Even in this case, the specification of metric hierarchies can suffice to enable going even further down to the actual problem by also observing the objects being involved. The inspection of the lowest level metric values could be subject to automatic analysis tools to produce the respective derivations needed or based on the experience of the analyst which can know what can be the improper measurement values for quite low-level metrics.

In KPI assessment, we are also interested in deriving other information, such as the value trend with respect to the previous assessments, by performing different analysis kinds. For instance, we could evaluate whether the BPaaS performance gets gradually reduced from the very beginning. To this end, to also make a connection to the original OWL-Q concept called *Measurement*, specifying a measurement’s value and its timestamp, a new sub-concept named as *KPIAssessment* was developed, specifying additional information spanning the value trend and the KPI violation kind (warning or fatal) occurred.
Towards Semantic KPI Measurement

By connecting a KPI to a business goal to be satisfied, we have the capability to assess the respective goal’s achievement. In this way, such a linkage can enable performing goal-based analysis in order to reach interesting conclusions related, e.g., to the satisfaction of strategic goals from operational ones.

To this end, OWL-Q was further extended to both specify goals and their linkage to KPIs. First, the concept of Goal, representing any goal kind, along with respective sub-concepts mapping to strategic, tactical, operational, functional and non-functional goals were introduced. Any goal was associated with a name, description and application level, while operational goals were mapped to the BPs used to satisfy them (another connection point with Dependency ontology). Goals were also linked with each other via AND/OR self-relations or contribution relations to enable forming goal hierarchies from strategic to operational goals. The Contribution concept was modelled to express contribution relations by linking a goal with another goal or KPI and being mapped to a certain contribution level.

4.2 Dependency Ontology

Various analysis types over a certain system can be performed only if the evolution of the system dependency model is captured. This model reveals what are the system components, how they are interconnected and what is the interconnection direction. Such a direction indicates the way faults and measurements can be propagated from lower to higher abstraction levels. Following the opposite direction enables conducting root cause analysis, i.e., from a current, issue at a high-level component down to the actual component to blame in a lower-level.

The Dependency ontology proposed captures both deployment and state information about all components in a BPaaS hierarchy. It extensively covers many information aspects, thus becoming suitable for many different BPaaS analysis kinds, including: (a) KPI analysis; (b) (semantic) process mining [6], due to the coverage of (semantic) I/O information for tasks and workflows; (c) best BPaaS deployment analysis [14] as all possible deployment information across all levels is covered; (d) the detection of event patterns [31] leading to a KPI violation.

Figure 2 [18] depicts the Dependency Ontology which follows the well-known type-instance pattern, enabling to capture both the allocation decisions made as well as the whole BPaaS allocation history and evolution. The proposed ontology also captures organisational information. In particular, the Tenant concept was incorporated to model an organisation, also associated with a User set. Different kinds of tenants exist: (a) Brokers that offer a BPaaS, (b) Customers that can purchase a BPaaS, and (c) Providers which offer a cloud service supporting the BPaaS execution. A string type enumeration was also modelled to cover different kinds of customer organisations, like SMEs, start-ups or big companies.

Similar to the new design of OWL-Q [17], the Dependency ontology includes generic data properties which can be mapped to all or a subset of the concepts modelled, such as id properties to be attributed to any concept and endpoint/URL properties to be attributed only to services. As such, from now on,
the analysis focuses on specific data properties that individually characterise aspect-specific concepts and not generic ones.

We follow a top-down analysis for the ontology from the type to the instance level. The top concept at the type level represents a BPaaS which is associated with an owner and an executable Workflow to be run in the Cloud. The executable workflow is related in turn to its main Tasks.

A task can have input and output Variables and is associated with a certain user or role that can be assigned to it. It can be further classified into a ManualTask (performed by human workers), a ScriptTask (performed automatically via a script) and a ServiceTask (performed automatically by calling a Software as a Service (SaaS)).

As stated, a BPaaS corresponds to an allocated executable workflow. This signifies that: (a) many BPaaSes can share the same workflow; (b) from these BPaaSes each BPaaS can be uniquely distinguished based on the specific set of allocations performed on that workflow. Such a distinction is enabled via the
modelling of the Allocation concept which represents an allocation decision that is linked to a certain BPaaS and workflow. Such an allocation maps a service task in the workflow to a SaaS, either an ExternalSaaS or a (internal) Service-Component. In case of a ServiceComponent, the allocation is also associated to an Infrastructure as a Service (IaaS) which supports its deployment. A IaaS is characterised by the following properties: its number of cores and the main memory & storage size.

The top concept at the instance level is BPaaSInstance which represents an instance of a BPaaS associated with the Customer that has purchased it, its actual cost and the DeployedWorkflow. The latter concept represents the BPaaS workflow deployed on behalf of a customer upon successful purchasing of this BPaaS. The instances of this workflow are then associated with: (a) the instances of tasks (TaskInstance) created; (b) its start and end time; (c) its resulting state (“SUCCESS” or “ERROR”); (d) the user that has initiated it; (e) the adaptations performed on it to maintain the service level promised in the corresponding SLA signed with the customer. Instances of tasks of this workflow are associated with similar information that incorporates the user (if exists) executing them and the input/output VariableInstances that they generate. The latter map to the actual Variable concerned and include the respective values produced.

The Dependency ontology models two types of concrete allocations: (a) from a deployed workflow task to a SaaSInstance realising its functionality; (b) from an internal SaaSInstance to the IaaSInstance hosting it (in case of instances of internal service components). Both SaaS and IaaS instances are sub-concepts of ServiceInstance which encompasses their common features including the service’s endpoint and its physical & cloud location. IaaSInstances further encapsulate certain hardware-specific information, such as the id and name of the image involved as well as the respective OS deployed, while they also map to respective IP on which they are available.

The FAO (United Nations Food and Agriculture Organisation) geopolitical ontology\(^6\) has been used to capture physical locations and their hierarchies. In particular, the physical service location is mapped to the concepts of geographical region and self-governing that represent geographical regions (e.g., continents), and all locations mapping to self-governing countries, respectively. The CloudLocation concept was also used to structure arbitrary hierarchies of cloud locations to cover the hierarchy diversity across different cloud providers. The Cloud concept was also modelled in order to specify the cloud that is being offered by a certain Provider for which the corresponding location hierarchy applies.

Concerning BPaaS adaptation, the most usual types as reported in respective literature have been modelled, i.e., service replacement and scaling ones. Any kind of Adaptation is mapped to its start and end time, its final state and the adaptation rule triggered. A ServiceReplacement is additionally associated with the service instance being substituted and the service instance substituting it.

\(^6\) http://aims.fao.org/aos/geopolitical.owl
On the other hand, any kind of scaling maps to the IaaS to be scaled. Two main scaling types have been covered: (a) Horizontal Scalings for which we specify one or more service components hosted by the IaaS to be scaled plus the amount of instances to be generated or removed; (b) Vertical Scalings for which we indicate the respective increase or decrease amount of the corresponding IaaS characteristic(s) scaled.

5 KPI Analysis System

In the following, we analyse the architecture of the KPI analysis system, then we provide some implementation details and finally we describe in detail the algorithms used for the KPI measurement and drill-down mapping to respective components of the system architecture analysed.

5.1 Architecture

The proposed system offers the following main features: (a) it supports multi-tenancy in the context of BPaaS brokers. This means that the system can support multiple BPaaS brokers in the conduction of KPI analysis tasks and also enforces the right access control such that no broker can conduct analysis tasks and see corresponding analysis results pertaining to other brokers; (b) it enables not only to evaluate the current value of KPI metrics but also the browsing and search over their measurement history; (c) it supports the dynamic evaluation of KPI metrics via the more flexible exploration of the possible metric space; (d) it supports the storage of the KPI measurements produced allowing the more efficient querying of the KPI evaluation history rather than its more time-consuming reconstruction; (e) it adopts a high-level language in the specification of the KPI metrics to be evaluated which is closer to the human perception.

Figure 3 depicts the service-oriented architecture of the KPI analysis system which comprises eleven main components and follows the known three-level implementation pattern of UI-business logic-database. In the following, we analyse the functionality of each system component in different paragraphs.

Hybrid Dashboard This component constitutes the main entry point to the system from which respective analysis tasks can be performed and then their results are represented according to suitable visualisation metaphors.

Harvester This composite component is responsible for the harvesting of information from different components of the BPaaS management platform. The information harvested at a particular frequency is further structured and linked according to the two ontologies proposed as well as stored in the Semantic KB. The population is performed periodically to not overwhelm the system but more frequently with respect to the way KPI measurements are assessed. The analysis of the internal architecture of this composite component is out of context of this paper but can be found here [19].
**Conceptual Analytics Service** This component is a REST service offering the two main KPI analysis capabilities at the following URL: http://134.60.64.222/evaluation. As such, it can be exploited by external components to programmatically deliver these capabilities. The KPI measurement comes into two main forms: (a) measurement over certain KPIs which have been already defined for the BPaaS; (b) measurement over dynamically specified KPI metrics - here is where the flexibility in the evaluation is supported as the user can specify dynamically KPI metric formulas as well as play with the metric measurement schedule and window. For the first type of KPI measurement, there are two possibilities as mentioned in the main features of the KPI analysis system: either we directly measure the KPI or we just perform a query over its measurement/evaluation history. The first possibility is more appropriate in the case of the production of new KPI measurements while the second is more efficient in terms of evaluation response time in the context of browsing the evaluation history of a certain KPI.

Similarly, KPI drill-down comes into two forms: (a) KPI drill-down based on KPI parent-child relationships - here the whole KPI hierarchy is exploited in order to go from the violation of a high-level KPI to a low-level KPI which is to be blamed for this violation; (b) KPI drill-down based on the KPI metric parent-child relationships - here we follow the dependencies between KPI metrics in order to find the root causes of KPI violations. Another main differentiation with respect to the other drill-down type is the fact that the analysis can further go down into metrics over which KPIs have not been specified enabling to perform a deeper root cause analysis.

![Fig. 3: Architecture of the KPI Analysis System.](image)

Apart from these two analysis capabilities, the *Conceptual Analytics Service* offers additional functionality which covers
1. the enumeration of all KPIs that can be evaluated for a certain BPaaS - this can enable a UI to visualise represent all these KPIs and assist the user in selecting the right KPI to evaluate for that BPaaS
2. the enumeration of all metrics that can be used for the construction of dynamic metric formulas for on-the-fly evaluation of new, more composite KPI metrics
3. the enumeration of all BPaaS customers for which a certain BPaaS applies - this enables the system to focus the analysis on a certain customer once this customer has been selected in the Hybrid Business Dashboard.
4. the supply of a SPARQL query which can be evaluated in the underlying semantic LD database. This latter functionality is more suitable for an expert which is aware of the two ontologies proposed for a more ad-hoc and specialised exploration of the possible metric space.

Underlying the Conceptual Analytics Engine lie two main components which are responsible in turn for the delivery of the two main KPI analysis functionalities: (a) the Query Creator and the Drill-Down Handler. These two components are now analysed in detail.

Query Creator This component is responsible for the production of a SPARQL query for the evaluation of a certain KPI metric. This query is then issued by the Conceptual Analytics Service on the Semantic KB in order to produce the corresponding KPI measurement as a result that is finally compared against the KPI threshold in order to evaluate whether the KPI is violated or not. Internally, the Query Creator orchestrates the functionality of the following three components.

KPI Handler This component is responsible for the management of the KPIs. In particular, this component extracts the KPI model of a certain BPaaS which is specified in OWL-Q into a list of KPI objects that can then be retrieved by the Query Creator component for their further respective processing and transformation into an SPARQL query. This extraction is supported by the OWL-Q processing library which is analysed in the implementation details subsection.

Resource Accessor This component is responsible for accessing information resources from external information sources. In particular, each time a metric formula needs to be transformed into a SPARQL query, it is checked whether it accesses external information. If this holds, then this component takes care of accessing the DB or invoking the corresponding API exposed by the external information source in order to retrieve the respective information resource required. Subsequently, the retrieved information can be exploited as a constant in the SPARQL query used to evaluate the metric computation formula.

SPARQL Transformer This component is responsible for the actual transformation of the KPI object into a SPARQL query. More details about how this transformation takes place is supplied in Section 5.3.
Drill-Down Handler This component is responsible for the execution of the two main drill-down forms. Internally, it exploits the Query Creator in order to produce respective SPARQL queries when corresponding KPI metrics need to be evaluated. This is needed, for example, when we need to evaluate the KPIs within the hierarchy of a certain high-level KPI. More details about these two forms of KPI drill-down are supplied in Section 5.3.

Semantic KB This component is a semantic Triple Store which enables the management and storage of semantic information, which is structured in our case based on the two ontologies proposed. To address the heterogeneity of different triple store implementations and their exchange, a Semantic KB Service was developed on top of this KB to offer a RESTful interface enabling LD management via methods that facilitate issuing SPARQL queries, inserting as well as updating RDF graphs.

Meta-Model Repository This component includes basic information about BPs like their models and annotations to be exploited for visualisation and information harvesting reasons. In the current implementation prototype, this component is shared between the BPaaS Design and Evaluation environments in the corresponding BPaaS management platform (see CloudSocket project). This witnesses the closeness between the two BP lifecycle activities and the respective level of cooperation established between them.

5.2 Implementation Details

All the components were developed in the Java programming language. The development of the Conceptual Analytics Service relied on the Jersey library\(^7\).

The transformation of a OWL-Q KPI model to KPI in-memory objects relied on the OWL-Q library. This library enables the domain code-based representation of OWL-Q models as well as their loading from and storage to the file system. Such a library can also be exploited for the production of a customised OWL-Q editor in order to provide a more user-intuitive way of editing and managing OWL-Q models with respect to generic OWL editors like Protege\(^8\). Such an editor would also enable the expert user not to possess any knowledge over OWL and ontology modelling for the production of KPI models. This would increase the mass of possible experts that can be involved in the production of KPI models based on OWL-Q.

The Resource Accessor is a Java-based component which encompasses drivers for the accessing of widely-known DBs like MySQL or Postgresql. It also relies on the Jersey library for the invocation of REST APIs for the accessing of external information needed in metric formulas.

The Semantic KB was realised based on the Virtuoso Triple Store\(^9\). This triple store is quite efficient in SPARQL query evaluation, especially as it relies

\(^7\) https://jersey.github.io/
\(^8\) protege.stanford.edu
\(^9\) https://virtuoso.openlinksw.com
on a column-based object database. The service on top of the Semantic KB was realised again via the Jersey as well as the sesame RDF management library\textsuperscript{10}. This enabled us to exploit the sesame driver in Virtuoso for a more advanced and high-level handling of the SPARQL query evaluation.

5.3 KPI Analysis & Drill-Down Algorithms

KPI Analysis Independently of what is the KPI analysis form, a core KPI metric evaluation functionality has to be realised. To this end, a semantic approach for this realisation was selected for two main reasons: (a) evaluation accuracy is higher; (b) semantic linking enables better exploring the actual space of possible KPI metric computation formulas. The latter actually reflects the current KPI evaluation practice where, apart from some KPI metrics that can be fixed in advance (e.g., cross-domain metrics like process duration), the rest of the KPI metrics must be computed based on the knowledge and expertise of the (BP performance) evaluator.

Semantic linking enables a richer connection between different information aspects to facilitate the metric formula construction with respect to other forms of measurement storage and aggregation. On the other hand, other measurement system alternatives, like Time Series Databases, require an a-priori design of the measurement space and do not enable advanced forms of information linking and aggregation. This means that such systems suffer from a certain form of inflexibility while they do not offer a satisfactory level of dynamicity.

The most intuitive way to express metric formulas for a semantic approach that adopts LD technology is via SPARQL queries. However, SPARQL queries require deep knowledge about LD technology and great expertise in SPARQL query modelling which might not be possessed by a BP performance evaluation expert. Such an expert would rather prefer to specify the metric formula in mathematical terms via a simplified language. This observation has led us to adopting the OWL-Q KPI extension. By relying on an user-intuitive OWL-Q editor and the fact that ontologies represent human conceptualisations of a domain, the expert can more naturally specify the metric formula. This obstacle could be further overpassed by introducing a domain-specific language for pure mathematical metric formula expressions which is left as a possible future work direction.

Based on this (language) adoption, the challenge that still remains for a suitable KPI evaluation lies in the capability to transform OWL-Q models, and especially KPI metric formulas, into a SPARQL query specification. This metric formula to SPARQL query transformation comprises some specific hurdles that had to be overcome. First, it is differentiated based on the metric kind. Two metric kinds exist: (a) customer-specific, pertaining to a certain BPaaS instance purchased by a customer of the BPaaS broker at hand; (b) broker-specific, pertaining to the overall performance of the BPaaS offered across all customers. Customer-specific metrics have as their measurement space all the measurements
produced for the customer’s BPaaS instance, while broker-specific metrics have a broader measurement space spanning all measurements over all instances of a BPaaS purchased.

Second, two main factors harden the transformation: (a) it should not only consider the metric itself (i.e., the actual computation) but also the metric and condition context, which indicates that all such information should be linked together to obtain the right set of measurements to be aggregated; (b) the dynamic KPI evaluation kind envisioned, where the expert can experiment with formulas, metric kinds, evaluation (schedule & windows) and history periods, does not allow storing the measurements in the physical storage once they are derived. To this end, the way lower-level KPI metrics can be derived needs to be accounted for when attempting to compute a high-level KPI metric. This means that we might need to go down even to the level of low-level or resource metrics for which measurements are already produced to derive the measurements for a high-level KPI.

By taking into consideration the above two issues, a particular transformation algorithm has been produced which, depending on the input provided, attempts to generate dynamically the SPARQL query to be issued for deriving the respective metric measurement. Listing 1.1 depicts the pseudo-code of this algorithm.

Listing 1.1: Transformation & Drill-Down Algorithms Pseudo-code

```java
ResultSet evalKPI(Metric m, Object object, BPaaS bpaas,
   DateTime start, DateTime end, String custID)
{
    MetricFormula mf = expandFormula(m.formula);
    List<String> vars = getVars(mf);
    String clause = getClause(mf);
    String query = createQuery(vars, clause, object, bpaas,
       start, end, m.metricContext.schedule, custId);
    return runQuery(query);
}

String createQuery(List<String> vars, String clause,
   Object obj, BPaaS bpaas, DateTime start, DateTime end,
   Schedule schedule, String customerId)
{
    String query = insertPrefixes();
    query += applyClause(clause, getBrokerGraph());
    query += createMeasurementTriples(vars, obj.URI);
    query += createInterLink(bpaas, object, customerId);
    query += applyFilters(start, end, vars);
    query += applyGrouping(schedule, vars);
    return query;
}

Tree<ResultSet> kpiDrillDown(KPI kpi, BPaaS bpaas,
   DateTime start, DateTime end,
   String custId)
{
    Tree<ResultSet> result = evalKPI(kpi.metric,
       kpi.object, bpaas, start, end, custId);
    Set<ResultSet> results = measureKPIsInParallel(kpi.children,
       start, end, customerId);
    result.addChildren(results);
    return result;
}

Hashtable<Metric, ResultSet> metricDrillDown(Metric m,
   Object object, BPaaS bpaas, DateTime start,
   DateTime end, String custId)
{
    MetricTree mt = expandFormulaInTree(m.formula);
    Set<MetricNode> metrics = getLeaves(mt);
    Hashtable<Metric, ResultSet> results = new Hashtable<Metric, ResultSet>();
    while (!metrics.isEmpty())
    {
      MetricNode mn = metrics.iterator().next();
      if (mn.isLeaf())
        results.put(mn.metric, evalKPI(mn.metric,
           mn.object, bpaas, start, end, custId));
      else
        results.put(mn.metric, measureKPI(mn.metric,
           mn.object, start, end, customerId));
    }
    return results;
}
```

Towards Semantic KPI Measurement 19
This algorithm (see evaluateKPI method) comprises four main steps: (a) metric formula expansion which includes the recursive substitution of component metrics, for which measurements are not stored in the Semantic KB, with their derivation formulas; (b) SPARQL query variable derivation from those metrics in the expanded formula, i.e., the leaf ones, for which measurements have been stored; (c) production of the (SPARQL) select clause from the expanded formula; (d) production of the whole SPARQL query.

The SPARQL query production (see createQuery method) involves executing the following steps: (i) creation of query prefixes; (ii) application of SELECT & FROM clauses by also taking into account the respective LD graph URI mapping to the individual RDF graph of the broker from which the relevant information for the query evaluation can be obtained; (iii) generation of the triple patterns mapping to the measurements of the leaf metrics / variables in the expanded metric formula; (iv) enforcement of the correlation between measurements according to the object being measured, the customer (if given as input) and the respective BPaaS instances mapping to these measurements; (v) application of the filtering (FILTER clause) over the history period to select measurements produced only on that period; (vi) application of SPARQL GROUP BY clauses based on the KPI metric evaluation period, i.e., its measurement schedule.

To exemplify this OWL-Q-to-SPARQL transformation algorithm and raise its understanding level, we now highlight its application on a specific example of a KPI metric by especially taking a closer look at the corresponding SPARQL query being generated.

Suppose that the average availability metric \( AVG_A \) has to be measured for the whole BPaaS workflow. This metric can be computed via the formula \( \text{MEAN}(\text{RAW}_A) \), where \( \text{RAW}_A \) represents the instance-based availability metric for this workflow. Further suppose that: (i) the average availability metric should be calculated every 1 hour, while the raw availability metric every minute; (ii) the history period is 1 day.

The first transformation algorithm step will expand the above computation formula based on the measurability of the component metrics of average availability. In particular, \( \text{RAW}_A \) is not stored in the Semantic KB, thus it is expanded into its derivation formula

\[
\frac{\text{UPTIME}}{\text{TOTAL\_OBSERVATION\_TIME}}
\]

where \( \text{UPTIME} \) is a raw metric and \( \text{TOTAL\_OBSERVATION\_TIME} \) is a constant. As such, the final expanded formula will take the following form:

\[
\text{MEAN}\left(\frac{\text{UPTIME}}{\text{TOTAL\_OBSERVATION\_TIME}}\right)
\]

Based on this formula, the next two algorithm steps will generate a set of one variable (“\( \text{uptime} \)” ) and the select clause (“SELECT (AVG(?uptime / 60) as...”)
Towards Semantic KPI Measurement

The fourth step will finally generate the actual SPARQL query depicted in Figure 4 [18]. This SPARQL query is now explained by focusing over all the steps involved in the createQuery method and the content generated by them.

As uptime is calculated every second, please observe that 60 is the total observation time constant.

Lines 1-2 supply the prefixes of the two ontologies being exploited mapping to the first query generation step. Lines 3-4 depict the query SELECT & FROM clauses produced from the second step of the query generation sub-algorithm.

Lines 5-9 express a set of triple patterns, generated by the third query generation step, linking the uptime measurement to: (a) the Uptime metric; (b) its actual value used in the formula of Line 3; (c) the actual dateTime where this measurement was produced; (d) the URI of the object measured (i.e., a workflow). These lines guarantee that we operate over Uptime measurements but do not provide suitable connections to other major information aspects, such as which BPaaS is actually concerned.

To this end, Lines 10-13, which map to the fourth query generation step, realise the needed connections from the object measured to both the BPaaS instance involving it and the current BPaaS at hand. Line 10 links the current BPaaS to one of its instances, while Line 11 connects this BPaaS instance with a deployed workflow. Line 12, currently commented, could link this instance to a certain client that has purchased it, in case we are dealing with a customer-specific metric. Finally, Line 13 links the deployed workflow to the actual workflow instance measured. Depending on the kind of object being measured, Lines 11-13 can be differentiated. For instance, if a task instance is to be measured, we need to add another triple pattern linking the workflow instance involved with this task instance.

Line 15, currently commented, mapping to the fifth query generation step, supplies a SPARQL FILTER constraint that can restrict the history period under investigation. In particular, the conjunction of two simple constraints is expressed over the dateTime of the measurement indicating that this dateTime
should be greater or equal to the low bound `dateTime` of the considered period and less than or equal to the upper bound `dateTime` of this period. This line is commented as the whole evaluation history of the KPI metric is explored.

Finally, Line 17, generated by the last query creation step, supplies a grouping statement where the last sub-group directly relates to the evaluation period of the KPI metric (i.e., per hour). This statement groups first the results based on the month, then on the day and finally on the respective hour.

**Drill-Down Algorithms** Two main forms of drill-down are supported by the KPI Analysis System proposed: (a) based on KPI parent-child relationships; (b) based on KPI metric parent-child relationships. It can be indicated that the first form is equivalent to the second one. This is not the actual case due to the fact that while KPI relationships might involve corresponding metric relationships, the latter relationships might not be complete in the sense that the metrics of the child KPIs cannot be solely used for the production of the measurement of the metric of the parent KPI. This indicates that possibly there is a component metric of a parent KPI metric for which a KPI has not been supplied. For instance, suppose that we have a parent KPI metric over BPaaS availability. This metric might be computed from other metrics that span the availability of the SaaS services exploited plus the availability of the workflow engine used to execute this BPaaS. Logically speaking, a BPaaS broker might express KPIs for all SaaS-based availability metrics but not for the workflow engine availability one. As such, the first form of KPI drill-down will be able to show the relationships between the KPIs and their respective evaluations. On the other hand, the second KPI drill-down form will show the relationships between the KPI metrics and their measurements. As such the first form is suitable until the point where KPI relationships clearly show the root causes of problems. However, the second form is a more elaborate one which is also able to bring the analysis until the lowest possible level. These two KPI drill-down forms are now analysed in the following paragraphs.

*KPI Parent-Child Relationship Based Drill-Down* This form of KPI drill-down is handled by the algorithm mapping to the `kpiDrillDown` method in Listing 1.1. In this form, the parent-child relations between KPIs are exploited. The main logic of this algorithm is quite simple: each KPI involved in the hierarchy of a top-level KPI needs to be evaluated. In this respect, the algorithm comprises two main steps: (a) processing of current KPI by producing its SPARQL query via the OWL-Q-to-SPARQL transformation and evaluation of the query in order to produce the respective measurement; (b) processing of the child KPIs of current KPI in parallel and storage of the respective measurements produced into a hierarchical tree-form where parent KPI measurement is described first and then followed by the measurement trees of all its children. Due to the recursive form of this algorithm we will be able in the end to produce the whole hierarchical measurement tree for the top-level KPI which could then enable the respective assessment of all the KPIs involved in the top-level KPI's hierarchy.
KPI Metric Parent-Child Relationship Based Drill-Down This KPI drill-down form is handled by the algorithm mapping to the metricDrillDown method in Listing 1.1. This algorithm exploits the OWL-Q-to-SPARQL transformation one by also considering the whole derivation tree of the current KPI metric at hand. It sequentially executes the following steps: (a) expand recursively the top metric’s derivation list until leaf metric nodes are reached. This leads to producing a metric (derivation) tree which has as leaf nodes metrics for which measurements exist in the Semantic KB; (b) compute the needed intermediate metric (node) values based on the SPARQL-based transformation approach in a bottom-up way. This means that we proceed in a level-by-level basis in the computation of not yet processed metrics for which the child nodes map to metrics whose values have been already computed or map to measurements already stored in the Semantic KB. This procedure the propagates up the produced measurements in the tree until the top level metric is reached. Each time a KPI metric node is visited, its values are produced based on the metric formula involved and the already produced measurements. The produced measurements are stored in the hashtable, from metrics to measurement result sets, to support the higher-level metric computations plus the production of the drill-down results to be returned.

6 Conclusions

This paper has presented a semantic approach, as well as a respective service-oriented multi-tenant system realising it, to KPI measurement which enables the intelligent and dynamic exploration of the whole KPI metric space. This approach relies on the suitable, semantic and human-oriented definition of the KPI metric as well as the expansion of its formula and its transformation into a SPARQL query which is then issued over a semantic KB. Two different KPI drill-down forms are also offered by the proposed KPI analysis system which capitalise over the KPI measurement algorithm and the KPI’s (metric) hierarchy tree. Two ontologies have been also proposed able to support the complete KPI specification and the capturing of BPaaS dependency models. These ontologies are exploited to semantically link information drawn from BPaaS execution to populate the semantic KB. Such a linking then enables various types of BPaaS analysis over the collected information, apart from the currently offered one, such as process mining or best BPaaS deployment discovery.

Future work will pursue the following research directions. First, further validating the proposed ontologies to obtain suitable feedback for optimising them. Second, thoroughly evaluating the KPI analysis system according to both performance and accuracy aspects. Third, realising and injecting additional BPaaS analysis algorithms into the respective KPI analysis system so as to transform it into a full-fledged BPaaS evaluation environment.

Acknowledgements This research has received funding from the European Community’s Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).
References

1. Caplan, R.S., Norton, D.P.: The Balanced Scorecard Measures that Drive Performance. Harvard Business Review 70(1), 281–308 (1992)
2. Castellanos, M., Casati, F., Shan, M.C., Dayal, U.: ibom: A platform for intelligent business operation management. In: ICDE. pp. 1084–1095. IEEE Computer Society, Washington, DC, USA (2005), http://dx.doi.org/10.1109/ICDE.2005.73
3. Chowdhary, P., Bhaskaran, K., Caswell, N.S., Chang, H., Chao, T., Chen, S.K., Dikun, M., Lei, H., Jeng, J.J., Kapoor, S., Lang, C.A., Mihaila, G., Stanoiu, I., Zeng, L.: Model Driven Development for Business Performance Management. IBM Syst. J. 45(3), 587–605 (Jul 2006), http://dx.doi.org/10.1147/sj.453.0587
4. Costello, C., Malloy, O.: Building a Process Performance Model for Business Activity Monitoring. In: Wojtkowski, W., Wojtkowski, G., Lang, M., Conboy, K., Barry, C. (eds.) Information Systems Development - Challenges in Practice, Theory, and Education. pp. 237–248. Springer-Verlag (2008)
5. Cui, Y., Nahorst, K.: QoS-Aware Dependency Management for Component-Based Systems. In: HPDC. p. 127. IEEE Computer Society (2001), http://dx.doi.org/10.1109/HPDC.2001.945183
6. De Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An Outlook on Semantic Business Process Mining and Monitoring. In: OTM. pp. 1244–1255. Springer-Verlag (2007), http://dl.acm.org/citation.cfm?id=1780453.1780517
7. Diamantini, C., Potena, D., Storti, E., Zhang, H.: An Ontology-Based Data Exploration Tool for Key Performance Indicators. In: ODBASE. pp. 727–744. Springer-Verlag, Amantea, Italy (2014), http://link.springer.com/chapter/10.1007/978-3-662-45563-0_45#page-1
8. Frank, U., Heise, D., Kattenstroth, H., Schauer, H.: Designing and utilising business indicator systems within enterprise models: Outline of a method. In: MobIS: Modellierung zwischen SOA und Compliance Management. Saarbrücken, Germany (2008)
9. Friedenstab, J.P., Janiesch, C., Matzner, M., Muller, O.: Extending BPMN for Business Activity Monitoring. In: HICSS. pp. 4158–4167. IEEE Computer Society (2012), http://dx.doi.org/10.1109/HICSS.2012.276
10. González, O., Casallas, R., Deridder, D.: MMC-BPM: A domain-specific language for business processes analysis. In: BIS. vol. 21, pp. 157–168. Springer, Poznan, Poland (2009), http://dx.doi.org/10.1007/978-3-642-01190-0_14
11. Gruschke, B.: Integrated Event Management: Event Correlation Using Dependency Graphs. In: DSOM (1998)
12. Hasselmeier, P.: Managing Dynamic Service Dependencies. In: DSOM. pp. 141–150. INRIA, Nancy, France (2001). http://www.simpleweb.org/ifip/Conferences/DSOM2001/DSOM2001/proceedings/S5-1.pdf
13. Karagiannis, D.: Bpms: Business process management systems. SIGOIS Bull. 16(1), 10–13 (August 1995)
14. Kritikos, K., Magoutis, K., Plexousakis, D.: Towards Knowledge-Based Assisted IaaS Selection. In: CloudCom. IEEE Computer Society, Luxembourg (2016)
15. Kritikos, K., Pernici, B., Plebiani, P., Cappiello, C., Comuzzi, M., Benbernou, S., Brandic, L., Kertész, A., Parkin, M., Carro, M.: A survey on service quality description. ACM Comput. Surv. 46(1), 1 (2013), http://doi.acm.org/10.1145/2522968.2522969
16. Kritikos, K., Plexousakis, D.: Semantic QoS Metric Matching. In: ECOWS. pp. 265–274. IEEE Computer Society (2006)

17. Kritikos, K., Plexousakis, D.: Semantic SLAs for Services with Q-SLA. In: ICWS. pp. 686–689. IEEE Computer Society, San Francisco, CA, USA (2016), http://dx.doi.org/10.1109/ICWS.2016.96

18. Kritikos, K., Plexousakis, D., Woitsch, R.: Towards Semantic KPI Measurement. In: CLOSER. pp. 63–74. SciTePress, Porto, Portugal (2017), https://doi.org/10.5220/000623800630074

19. Kritikos, K., Zegkinis, C., Seybold, D., Griesinger, F.: D3.6 – BPaas Monitoring and Evaluation Prototypes. CloudSocket European Project (June 2017)

20. List, B., Korherr, B.: An evaluation of conceptual business process modelling languages. In: SAC. pp. 1532–1539. ACM, Dijon, France (2006), http://doi.acm.org/10.1145/1141277.1141633

21. Liu, R., Nigam, A., Jeng, J., Shieh, C., Wu, F.Y.: Integrated Modeling of Performance Monitoring with Business Artifacts. In: ICEBE. pp. 64–71. IEEE Computer Society, Shanghai, China (2010), http://dx.doi.org/10.1109/ICEBE.2010.63

22. Motta, G., Pignatelli, G., Florio, M.: Performing Business Process Knowledge Base. In: First Internation Workshop and Summer School on Service Science. Heraklion, Greece (2007)

23. Pierantonio, A., Rosa, G., Silingas, D., Thönsen, B., Woitsch, R.: Metamodeling Architectures for Business Processes in Organizations. In: Proceedings of the Projects Showcase at STAF. CEUR, L’Aquila, Italy (2015)

24. del Río-Ortega, A., Resinas, M., Durán, A., Ruiz-Cortés, A.: Using templates and linguistic patterns to define process performance indicators. Enterp. Inf. Syst. 10(2), 159–192 (Feb 2016). http://dx.doi.org/10.1080/17517575.2013.867543

25. Rossini, A., Kritikos, K., Nikolov, N., Domaschka, J., Griesinger, F., Seybold, D., Romero, D.: D2.1.3 – CloudML Implementation Documentation (Final version). Paasage project deliverable (October 2015)

26. Seedorf, S., Schader, M.: Towards an Enterprise Software Component Ontology. In: AMCIS. Association for Information Systems (2011), http://aisel.aisnet.org/amcis2011_submissions/389

27. Wetzstein, B., Karastoyanova, D., Leymann, F.: Towards Management of SLA-Aware Business Processes Based on Key Performance Indicators. In: BPMDS. Montpellier, France (June 2008)

28. Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., Leymann, F.: Monitoring and Analyzing Influential Factors of Business Process Performance. In: EDOC. pp. 118–127. IEEE Press (2009), http://dl.acm.org/citation.cfm?id=1719357.1719370

29. Wetzstein, B., Ma, Z., Leymann, F.: Towards Measuring Key Performance Indicators of Semantic Business Processes. In: BIS. p. 227. Springer-Verlag (May 2008). http://ww2.informatik.uni-stuttgart.de/cgi-bin/NCSTR/NCSTR_view.pl?id=INPROC-2008-43&engl=1

30. Woitsch, R., Albayrak, M., Köhn, H., Utz, W., Ferrer, A.J., Irazoqui, J., Leonforte, A., Gallo, A., Miñá, V., Pacurar, R., Avasilcai, C., Arama, G., Bocca, R., Griesinger, F., Seybold, D., Domaschka, J., Kritikos, K., Plexousakis, D.: D4.1 – First CloudSocket Architecture. CloudSocket European Project (August 2015)

31. Zeginis, C., Kritikos, K., Plexousakis, D.: Event pattern discovery in multi-cloud service-based applications. Int. J. Syst. Serv.-Oriented Eng. 5(4), 78–103 (Oct 2015), http://dx.doi.org/10.4018/IJSSOE.2015100104