Towards Semantic KPI Measurement

Kyriakos Kritikos[1], Dimitris Plexousakis[1] and Robert Woitch[2]
[1]*Information Systems Laboratory, ICS-FORTH, Greece*
[2]*BOC, Austria*

Keywords: KPI, Semantics, Ontologies, Quality, QoS, Linked Data, Analysis, SPARQL

Abstract: Linked Data (LD) represent a great mechanism towards integrating information across disparate sources. The respective technology can also be exploited to perform inferencing for deriving added-value knowledge. As such, LD technology can really assist in performing various analysis tasks over information related to business process execution. In the context of Business Process as a Service (BPaaS), the first real challenge is to collect and link information originating from different systems by following a certain structure. As such, this paper proposes two main ontologies that serve this purpose: a KPI and a Dependency one. Based on these well-connected ontologies, an innovative Key Performance Indicator (KPI) analysis system is then built which exhibits two main analysis capabilities: KPI assessment and drill-down, where the second can be exploited to find root causes of KPI violations. Compared to other KPI analysis systems, LD usage enables the flexible construction and assessment of any KPI kind allowing experts to better explore the possible KPI space.

1 INTRODUCTION

Business processes (BPs) enable organisations to formulate and realise internal and external procedures which provide support or enable their core business. Respective information systems and IT technology then enables the execution and management of these BPs to enable core service and product delivery. The flexible BP management and optimisation is enabled via a lifecycle comprising the four main activities of design, allocation, execution and evaluation. The first three activities focus on bridging the well-known business-to-IT gap and enabling the BP execution. The last activity facilitates deriving business intelligence information via performing various analysis tasks which can facilitate BP improvement, thus closing the aforementioned lifecycle.

A well-studied BP evaluation task concerns Key Performance Indicators (KPIs) measurement and assessment. KPIs map to certain indicators related the BP quality. They usually include a metric and a threshold imposed on it, thus defining the minimum respective performance level to be sustained. The metric provides all measurement details needed to measure different BP quality attributes, which can be categorised into 4 groups: (a) time, (b) quality, (c) customer satisfaction, and (d) financial (Caplan and Norton, 1992). As such, the main goal of an evaluation expert would be to specify suitable KPIs, possibly spanning all four categories, which can be measured by the BP evaluation system and enable assessing the quality levels of BPs.

In this respect, various KPI measurement systems were proposed in the past, relying on different technologies, such as OLAP (Chowdhary et al., 2006) or SQL query evaluation (Castellanos et al., 2005). While KPI assessment can be performed extremely fast in these systems, we believe that the main goal of a KPI measurement system should not be the KPI assessment speed but to provide assistance to evaluation experts in defining the most suitable KPIs for a BP. As such, there is a lack of flexible and user-intuitive mechanisms via which KPIs can be defined in these systems. Moreover, such systems are usually special-purposed as they are designed to serve certain KPI metric types, such that the introduction of a new metric can require re-engineering the underlying system database. Finally, they do not employ sophisticated information integration mechanisms to integrate any kind of information source, even external ones.

The latter issue is critical in the context of BP as a Service (BPaaS), i.e., BPs that are moved to the cloud. Such a migration is becoming a trend nowadays due to the great advantages that cloud computing brings about, such as reduced cost and elasticity. To this end, support for this migration is greatly needed. This sup-

Kritikos, K., Plexousakis, D. and Woitsch, R.
Towards Semantic KPI Measurement.
DOI: 10.5220/0006238000910102
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 63-74
ISBN: 978-989-758-243-1

port can be realised in the form of a BPaaS management system, able to manage the whole lifecycle of a BPaaS.

As indicated in (Woitsch et al., 2015), an architecture to support the BPaaS management is quite sophisticated, involving different environments and components that can be hosted in different virtual machines (VMs). As such, to enable a BP's measurement, the system would have to collect and integrate information coming from many of these components. Even external information might be required, out of the control of the BP management platform, as in the case of platform as a service (PaaS) services.

To realise the vision of a BPaaS, enabling the flexible BP allocation and execution in the cloud, as well as address the aforementioned drawbacks, this paper endorses the usage of Linked Data (LD) technology to support the KPI analysis of BPaaS. This technology is selected based on the following reasons: (a) allows performing inferencing tasks to deduce added-value analysis information; (b) enables integrating information across disparate information sources, even in unforeseen ways; (c) LD are expressed via ontologies which are closer to human conceptualisation.

The information integration task is assisted by introducing two ontologies: (a) a dependency ontology capturing the dependencies between BPaaS components, across different abstraction levels (BP, software and infrastructure), and their state; (b) a KPI ontology as an extension of OWL-Q (Kritikos and Plexousakis, 2006) enabling the complete KPI specification. The first ontology constitutes the major integration point for information coming from different systems, enabling its suitable correlation for supporting KPI analysis. The second ontology enables formally 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.

By building on these two ontologies, an innovative KPI measurement system has been developed, able to integrate information from many parts of a BPaaS management system and offer two KPI analysis capabilities: KPI measurement and drill-down. The second capability relies on connecting different KPIs at both business and technical levels which enables performing root cause analysis over a high-level KPI violation. The proposed system enables the on-the-fly KPI metric formula specification and assessment, provided that the formula's correlation to a certain context is given. This showcases 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 necessary to better understand the main paper contribution. Section 4 analyses the two ontologies proposed. Section 5 provides the proposed system architecture and exemplifies the way KPI analysis is performed. Finally, Section 6 concludes the paper and draws directions for further research.

2 RELATED WORK

Based on the main paper contributions, related work spans KPI & dependency modelling and KPI analysis. Thus, its analysis is split in 3 sub-sections.

2.1 KPI Meta-Models

As KPI modelling is a pre-requisite for KPI assessment, a great amount of research work was devoted in producing KPI meta-models, languages and ontologies, especially as currently there is a lack of standardised BP languages that cover appropriately the BP context perspective (including goal-based and measurement information aspects) [evaluation].

To evaluate the related work in KPI modelling, we rely on a systematic approach which considers a set of comparison criteria, it summarises the comparison 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 then includes a discussion over the presented evaluation results.

The comparison criteria considered were the following: (a) *KPI coverage*: how well the notion of a KPI is covered; (b) *metric formulas*: computation formulas are provided supporting the KPI metric measurement; (c) *measurability*: other aspects complementing metric specification are needed to cover all measurement details (e.g., units, measured objects); (d) *goal coverage*: connecting KPIs to goals enables to assess whether operational or even tactical goals are satisfied by performing goal analysis; (e) *semantics*: if the meta-model / language is semantic or allows semantic annotations. Semantics enables formal reasoning and reaching better evaluation accuracy levels; (f) *information sources*: ability of the language to exploit both internal and external information sources; (g) *measurement origin*: the language ability to cover measurements and explicate their origin (probes, sensors, or humans); (h) *level*: the levels covered (BP, SE - service, Inf - Infrastructure).

Based on the table evaluation results, we can see that only our ontology scores well over all criteria, it has better performance for almost all of them and can

Table 1: Comparison table over KPI modelling work.

Work	KPI Cov.	Metric Formulas	Measur.	Goal Cov.	Semantics	Inf. Sources	Meas. Origin	Level
(Wetzstein et al., 2008a)	moderate	yes	moderate	no	no	internal	probes	BP, SE
(Motta et al., 2007)	low	no	low	yes	no	internal	-	BP, SE
(Pierantonio et al., 2015)	good	yes	low	yes	no	internal	-	BP
(Friedenstab et al., 2012)	good	yes	moderate	no	no	internal	probes	BP
(Frank et al., 2008)	moderate	no	low	yes	no	internal	probes	BP, Inf
(González et al., 2009)	low	yes	low	no	no	internal	probes	BP
(del Río-Ortega et al., 2016)	moderate	yes	good	yes	yes	internal	probes	BP
(Costello and Malloy, 2008)	low	no	low	no	yes	internal	probes	BP
(Liu et al., 2010)	low	yes	low	no	no	both	probes	BP, Inf
OWL-Q KPI Extension	good	yes	excellent	yes	yes	both	all	BP, SE, Inf

be considered as the most prominent. The only modelling work close to ours is the one in (del Río-Ortega et al., 2016). However, that work does not cover all levels, does not correlate measurements to human sources, exploits only internal information sources and provides a moderate KPI coverage. In addition, it does not directly model the notion of a metric but intermixes it with that of an indicator. In our opinion, this is wrong, especially when the latter notion is re-used in the context of KPI computation formulas, as it maps to a metric condition and not to the metric itself which represents all appropriate measurement details to enable the KPI computation. Please note, though, that the metric formula definition in that approach is quite interesting as it involves a kind of restricted natural language form. This might be more user-intuitive but clarity and comprehensiveness might be lost when recursive composite metric formulas need to be specified. A more mathematical form might have been more appropriate. This is actually an issue that we currently explore.

2.2 Dependency Meta-Models

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

By following the analysis approach in the previous section, we have come up with the following evaluation criteria: (a) *abstraction level*: which levels (denoted as BE, SE, Inf) in the BPaaS hierarchy are covered; (b) *formalism*: the dependency model formalism used; (c) *runtime*: whether the dependency model covers a dynamic or just a static system view. Dynamic views enable to cover the system evolution and provide support for realising monitoring and adaptation mechanisms; (d) *detail level*: how well the component dependencies are specified.

The respective work, apart from ours, encoded in the table is the following: (a) SEE (Seedorf and Schader, 2011), (b) GRU (Gruschke, 1998), (c) CUI (Cui and Nahrstedt, 2001), (d) HASS (Hasselmeyer, 2001), (e) TOSCA[1] and (f) CAMEL[2].

Table 2: Evaluation table 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 comparison table results clearly show that our ontology covers all possible levels, does capture runtime information and includes a good information level for the dependencies captured. It is thus better than all other work. Sole competitors are the approaches in (Seedorf and Schader, 2011; Rossini et al., 2015) which do not cover all BPaaS hierarchy levels. Moreover, the approach in (Seedorf and Schader, 2011) does not capture runtime information, while CAMEL does not rely on semantics. We should state that: (a) an ontology-based approach is essential to allow a better integration of dependency information from different information sources as well as interesting inferencing over this information; (b) the good dependency detail level in some modelling approaches constitutes a place for improvement.

2.3 KPI Analysis Systems

Various KPI analysis frameworks have been proposed employing techniques that mainly support KPI evaluation while in some cases KPI drill-down is also sup-

[1]`http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html`

[2]`www.camel-dsl.org`

ported. Most techniques focus on appropriately structuring the underlying database to support KPI analysis. In this sense, they employ relational or semantic dbs or data warehouses.

By following the same approach as in previous subsections, we have compiled the next evaluation criteria: (a) *analysis types*: which KPI analysis kinds are supported; (b) *db type*: type of db used to store the information needed for KPI analysis; (c) *evaluation technique*: the technique used to measure KPIs; (d) *drill-down technique*: the technique used for KPI drill-down; (e) *evaluation flexibility*: system flexibility in the exploration of the possible metric space; (f) *level*: the BPaaS hierarchy levels supported.

From the evaluation results of Table 3, we see that semantic dbs are do considered in more than half of the systems, signifying that their added-value is being recognised in terms of better linking information and enabling various forms of reasoning. We also see that almost half of the systems focus only on KPI evaluation. The approaches supporting KPI drill-down exploit two main techniques: decision trees and combination of metric & KPI hierarchies. The first technique is suitable when there are measurability gaps (disconnected metric trees) to be filled-in. The second is suitable when connections between KPIs and metrics exist such that we can go down to more technical KPIs and then continue from there by exploring the respective metric hierarchies involved.

A great variation in evaluation techniques can be seen, from SQL queries, OLAP and event-based metric formula calculation to WSML rules and SPARQL queries. We believe, however, that SPARQL queries 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) they work on the conceptual level which is more close to actual human conception.

Concerning evaluation flexibility, our system seems to be one step ahead from the work in (Wetzstein et al., 2008b; Chowdhary et al., 2006; Diamantini et al., 2014) as it does not only allow to map human-based formulations of metric formulas into SPARQL queries but also to play around with the metric and condition context. Combined also by the respective KPI ontology capabilities, it can also support exploiting various information sources, like metrics, service properties and external ones, thus enabling a better exploration of the metric space. In this respect, our approach is more complete and user-intuitive with respect to the other two systems.

Finally, We can see from the evaluation results that only our system is able to cover all levels. In fact only three our of seven systems do recognise the necessity to cover more than one level in the BPaaS hierarchy.

3 BACKGROUND

This section shortly analyses OWL-Q, as it is the basis for the KPI ontology proposed. OWL-Q is a prominent (Kritikos et al., 2013) non-functional service specification ontology that captures all possible measurability aspects. Each aspect is covered by a respective OWL-Q facet. OWL-Q is also accompanied by semantic rules enabling 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. OWL-Q currently comprises 6 main facets which are now analysed by focusing more on those facets that are more relevant to this paper's work.

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

The attribute, unit and value type facets enable specifying respective attribute, unit and value type elements. Attributes (e.g., *utilisation*) represent properties that can be measured by metrics. Units can be derived (e.g., *bytes/sec*), single (e.g., *sec*) or dimensionless (e.g., *percentage*). Value types represent the domain of values for metrics. As such, they can be used to validate whether measurements or thresholds in metric conditions are correct by checking whether they are included in them.

The metric facet enables describing how attributes can be measured via the conceptualisation of the *Metric* concept. Metrics can be raw (e.g., uptime) or composite (e.g., availability). Raw metrics are computed from sensors or measurement directives posed over service instrumentation systems. Composite metrics are 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 metric can be related to respective contexts detailing its measurement frequency and window.

The specification facet enables describing non-functional specifications as sets of respective capabilities. Each capability is expressed as a constraint which can be either a logical combination of other constraints (i.e., a *CompositeConstraint*) or a simple

Table 3: Evaluation table over KPI analysis work.

Work	Analysis Types	DB Type	Evaluation Technique	Drill-Down Technique	Evaluation Flexibility	Level
(Castellanos et al., 2005)	all	relational	SQL queries	decision trees	low	BP
(Wetzstein et al., 2009)	all	relational	formula comp.	decision trees	low	BP, SE
(Costello and Malloy, 2008)	evaluation	semantic	formula comp.	-	low	BP
(Wetzstein et al., 2008b)	evaluation	semantic	WSML rules	-	moderate	BP, SE
(Chowdhary et al., 2006)	evaluation	warehouse	OLAP	-	moderate	BP
(Diamantini et al., 2014)	all	semantic	SPARQL queries	KPI-based	moderate	BP
Our Framework	all	semantic	SPARQL queries	metric/KPI-based	good	all

condition over a metric (i.e., a *SimpleConstraint*. A metric condition is associated to two different contexts: the metric and condition ones. The latter explicates which object (e.g., service or service input) is measured and the way the condition should be evaluated over this object's instances.

4 KPI AND DEPENDENCY ONTOLOGIES

To enable performing any KPI analysis kind, there is a need to provide meta-models that structure and link respective information on which the analysis relies. As such, in the context of BPaaS KPI analysis, we have developed two main ontologies: (a) the dependency ontology covering BPaaS dependency models; (b) the KPI ontology covering KPI modelling. These ontologies are interlinked in one main connection point, the actual BPaaS element measured within the BPaaS hierarchy. Then, based on the BPaaS dependency model and the interconnections between different BPaaS elements, measurement propagation hierarchies to cover measurability gaps and the discovery of root causes for KPI violations can be enabled.

4.1 KPI Ontology

As OWL-Q completely covers the specification of QoS profiles and SLAs, it was decided to extend it to cover the KPI specification. This OWL-Q extension builds upon OWL-Q constructs only a minimum but sufficient number of relevant new parts. The current application of this extension over the CloudSocket project[3] use cases shows that OWL-Q can model all KPIs needed. This major evaluation step validates the design of this OWL-Q extension.

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

A *KPI* represents an indicator of whether BP performance is satisfactory, problematic or erroneous mapping to 3 states, captured by a warning and violation threshold. 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 one. For a negatively monotonic metric, the order between warning and violation thresholds is reversed and the state mapping is symmetric.

A KPI has been modelled as a sub-class of simple constraint that carries extra information. As a simple constraint already includes a reference to a metric and (violation) threshold, this extra information spans a human-oriented description (for human consumption), a validity period and the warning threshold.

While OWL-Q more or less fully covers the conceptualisation of a metric, it was extended to address the major issue of external information access via incorporating such information in metric formulas. By considering that all modern information sources are available in form of REST APIs or database endpoints, this extension was realised by introducing the *Query* and *APICall* as sub-classes of *Argument*. As such, instances of these classes can be directly used as input arguments in metric formulas. A *Query* specifies in an implementation-independent way the required information to connect and query a db spanning: (a) the db's connection URL; (b) the query language; (c) the actual query; (d) the db type.

An *APICall* includes all information needed to call the API and retrieve back the result, spanning: (a) the API URL; (b) values to all 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.

It might be imperative in certain cases (e.g., *customer satisfaction* metrics) to enable humans to manually provide measurements in the system such that the measurement-to-user linkage should be modelled. When connected to certain aspects like human trust and reliability, such linkage can enable reasoning over

[3] `http://www.cloudsocket.eu`

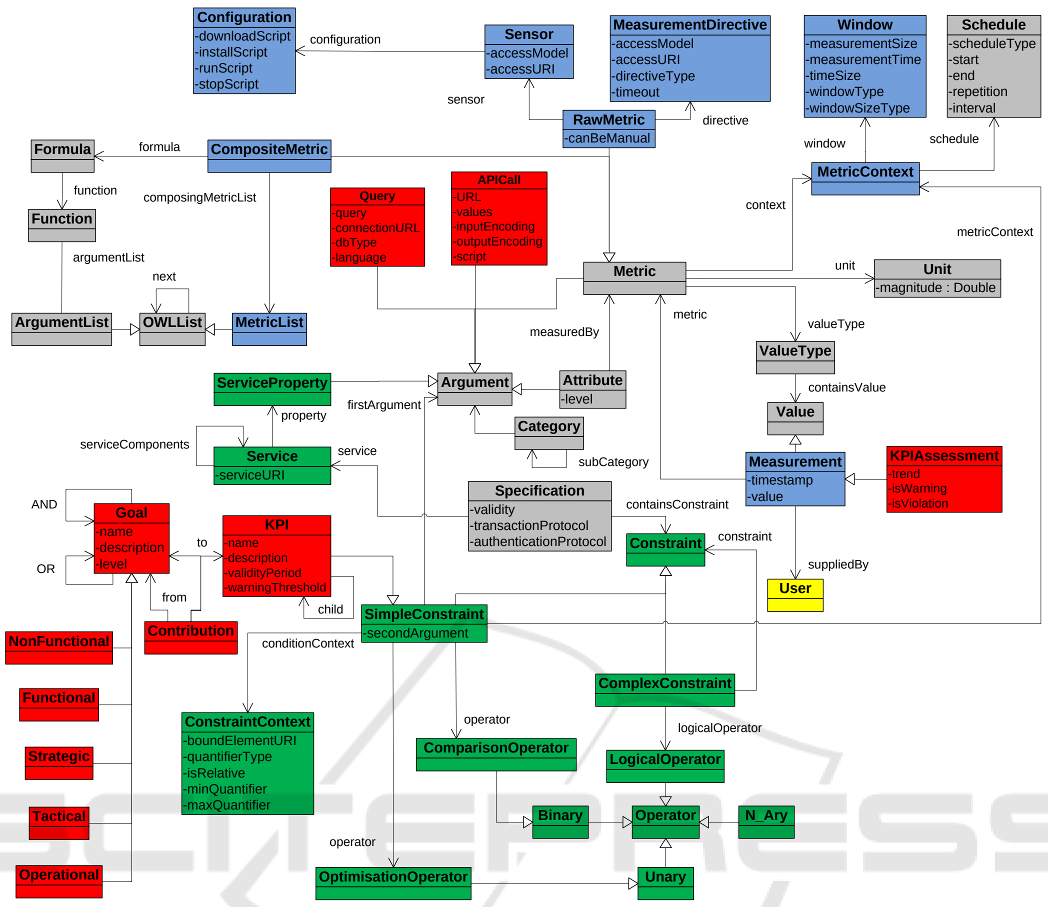


Figure 1: OWL-Q KPI Extension.

measurements and their propagation to establish a so-called trust level over them and a more suitable way to aggregate them. This was accommodated 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.

To enable a drill-down from higher- to lower-level KPIs to support root-cause analysis, we associate KPIs to each other via a *child* relation. This relation must conform to the respective 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 corresponding network latency.

During KPI assessment, we are interested in checking also other information, such as the value trend with respect to the previous assessments, by performing different analysis kinds. For instance, we could assess whether the BPaaS performance gets gradually reduced from the very beginning. As such, to also make a connection to the original OWL-Q concept called *Measurement*, specifying a measurement's value and its timestamp, a new sub-class was created, called *KPIAssessment*, including information parts focusing on covering the value trend and the KPI violation kind (warning or fatal) occurred.

A KPI should be connected to a business goal that must be satisfied to be used as an instrument to assess the respective goal's achievement. Such a linkage can also enable performing goal-based analysis in order to reach interesting conclusions related, e.g., to the satisfaction of strategic goals from operational ones.

As such, OWL-Q was further extended to specify goals and their linking to KPIs. First, the *Goal* concept, representing any goal kind, as well as respective sub-concepts mapping to strategic, tactical, operational, functional and non-functional goals were introduced. Any goal was given a name, description and application level, while operational goals were associated to the processes used to satisfy them (another connection point with Dependency ontology). Goals were also linked via AND/OR self-relations or contribution relations to enable forming goal hierarchies from strategic to operational goals. Contribution relations were modelled via the *Contribution* class which links a goal with another goal or KPI and is mapped to a specific level of contribution.

4.2 Dependency Ontology

To perform various analysis types over a certain system, it is critical to model the evolution of the system dependency model. Such a model reveals what are the system components and how they are interconnected along with the interconnection direction. Such a direction can indicate the way faults and measurements can be propagated from lower to higher abstraction levels. The opposite direction enables performing 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 proposed Dependency ontology covers both deployment and state information about all components in a BPaaS hierarchy. It extensively captures many information aspects making it suitable for many different kinds of BPaaS analysis, including: (a) KPI analysis; (b) (semantic) process mining (De Medeiros et al., 2007), as (semantic) I/O information for tasks and workflows is covered; (c) best BPaaS deployment analysis (Kritikos et al., 2016) as all possible deployment information across all levels is covered.

The Dependency Ontology, depicted in Figure 2, follows the well-known type-instance pattern, enabling the capturing of both the allocation decisions made as well as the whole BPaaS allocation history and evolution. The proposed ontology also covers organisational information. In particular, the *Tenant* concept was introduced to model an organisation which is also associated to a *User* set. A tenant can be a *Broker*, offering a BPaaS, a *Customer*, purchasing a BPaaS, or a *Provider*, offering a cloud service supporting the BPaaS execution. Customer organisations can be SMEs, start-ups or big companies (string type enumeration denotes them).

The ontology analysis follows a top-down approach from the type to the instance level. At the type level, the top concept represents the *BPaaS* which is associated to an *owner* and an executable *Workflow* to be run in the cloud. The latter is related to its main *Task*s. Tasks can have input and output *Variable*s and are related to a specific user or role that can be assigned to them. A *Task* can be further classified into a *ManualTask* (performed by human workers), *ScriptTask* (performed automatically via a script) and *ServiceTask* (performed automatically by calling a Software as a Service (SaaS))

A BPaaS corresponds to an allocated executable workflow. This means that: (a) a workflow can be shared by many BPaaS; (b) within one BPaaS, a specific set of allocations can be performed over a workflow. This resulted in modelling of the *Allocation* concept to represent an allocation decision and link it to a certain BPaaS and workflow.

Each allocation maps a service task to a *SaaS*, either an *ExternalSaaS* or a (internal) *ServiceComponent*. In case of a *ServiceComponent*, the allocation is also related to an Infrastructure as a Service (IaaS). A *IaaS* is characterised by the number of cores and the main memory & storage size properties.

At the instance level, *BPaaSInstance* represents an actual instance of a BPaaS associated with the *Customer* that has purchased it, its actual cost and the *DeployedWorkflow*. The latter represents the BPaaS workflow deployed in the context of a customer upon successful purchasing. The instances of this workflow are then associated to: (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 keep up with the SLOs promised. Instances of tasks of this workflow are associated to similar information which includes the user (if exists) executing them and their generated input/output *VariableInstance*s. The latter map to the actual *Variable* concerned and possess the respective values generated.

Two types of concrete allocations are modelled: (a) from a deployed workflow task to a *SaaSInstance* realising its functionality; (b) from internal *SaaSInstance* to the *IaaSInstance* hosting it. Both a SaaS and IaaS instance are sub-classes of *ServiceInstance*, encompassing their common features mapping to: the service's endpoint and its physical & cloud location. Physical locations are captured via the FAO (United Nations Food and Agriculture Organisation) geopolitical ontology[4]. *CloudLocation*s are used to structure arbitrary hierarchies of cloud locations to cover the hierarchy diversity across different cloud providers.

The most usual BPaaS adaptation types across the literature have been modelled: service replacement and scaling ones. Any *Adaptation* is associated to its start and end time, its final state and the adaptation rule triggered. A *ServiceReplacement* is associated to the service instance being substituted and the service instance substituting it.

Any scaling maps to the IaaS to be scaled. 2 main scaling kinds are covered: (a) *HorizontalScaling* where 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) *VerticalScaling* where we indicate the increase or decrease amount of respective IaaS characteristic(s).

[4]`http://aims.fao.org/aos/geopolitical.owl`

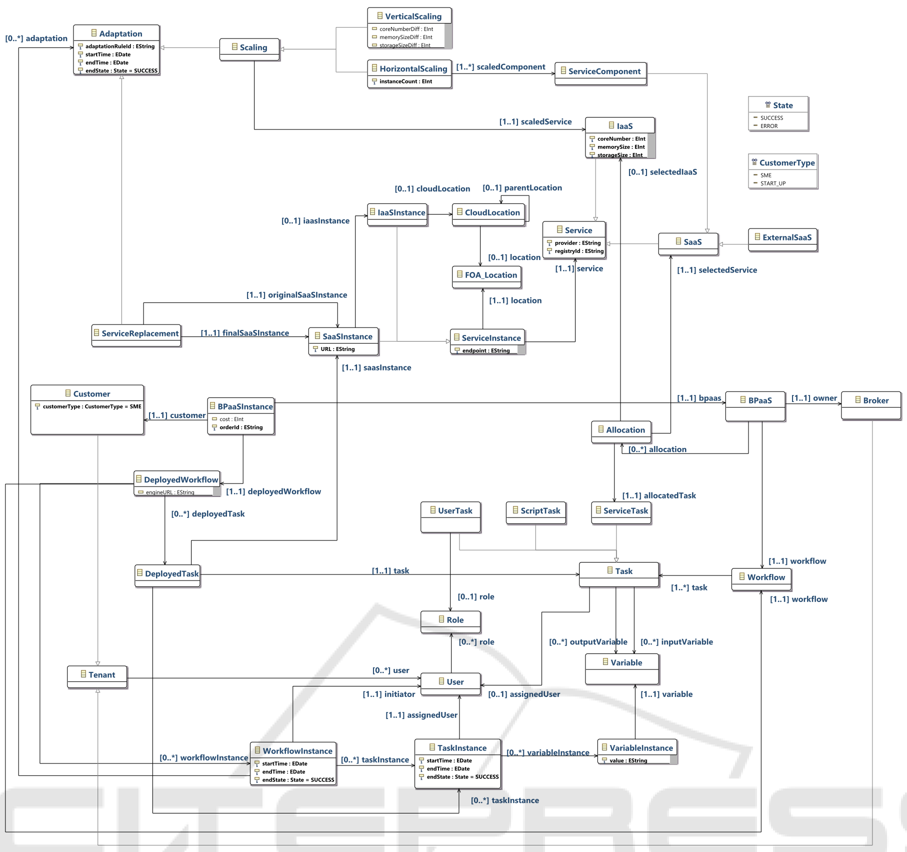


Figure 2: Dependency ontology UML class diagram.

5 KPI ANALYSIS SYSTEM

5.1 Architecture

The architecture of the KPI analysis system, depicted in Figure 3, follows a Service-Oriented Architecture and the known three-level implementation pattern of UI-business logic-database. This system comprises ten main components. The *Hybrid Dashboard* is the main entry point to the system from which respective analysis tasks can be performed and then represented according to suitable visualisation metaphors.

The *Conceptual Analytics Service* is a REST service offering the two main KPI analysis capabilities. As such, it can be exploited by external components to programmatically deliver these capabilities.

The *Conceptual Analytics Engine* orchestrates the way the KPI analysis and drill-down can be performed by invoking three main components: (a) the *Metric Specification Extractor* which extracts the KPI metric definition from the *Semantic KB*; (b) the *Metric Formula Extender* which expands the extracted

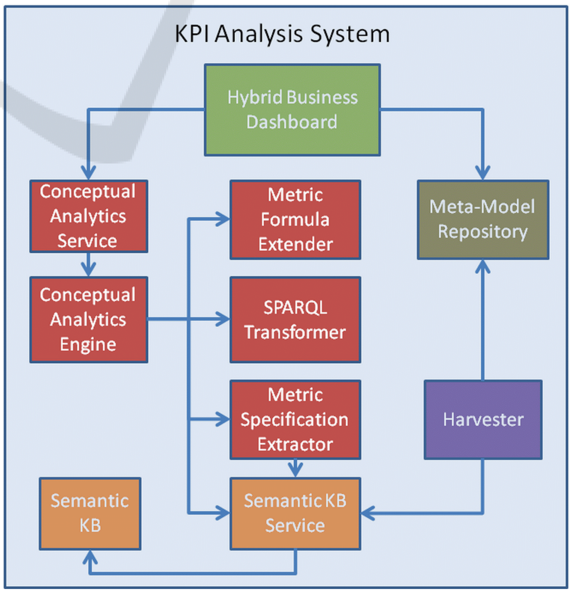


Figure 3: Architecture of the KPI Analysis System.

metric's formula based on the metric's derivation hierarchy; (c) the *SPARQL Transformer* which transforms the expanded metric formula into a SPARQL[5] query to be assessed over the *Semantic KB*.

The *Semantic KB* is a semantic Triple Store enabling the management and storage of semantic information, structured 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.

The *Harvester* populates the *Semantic KB* by using the *Semantic KB Service*. This component first obtains the needed information from disparate information sources within the BPaaS management prototype and then semantically enhances, links and formulates it based on the two proposed ontologies proposed. The population is performed periodically to not overwhelm the system but more frequently with respect to the way KPI measurements are assessed.

Finally, the *Meta-Model Repository* 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. This witnesses the closeness between the two BP lifecycle activities and the respective level of cooperation that has to be established between them.

5.2 KPI Analysis

No matter what is the KPI analysis form, the core KPI metric evaluation functionality needs to be realised. As such, we have selected a semantic approach for this realisation for two main reasons: (a) the evaluation accuracy is higher; (b) semantic linking enables exploring the actual space of possible KPI metric computation formulas. In fact, the latter reflects the current KPI evaluation practice where there might be some KPI metrics that can be fixed in advance (e.g., cross-domain metrics) while the rest of the metrics must be computed based on the knowledge and expertise of the (BP performance) evaluator.

In contrast to other forms of measurement storage and aggregation, the semantic linking enables the rich connection between different information aspects to facilitate the metric formula construction. On the other hand, measurement system alternatives, such as Time Series Data Bases, require an apriori design of the measurement space and do not allow advanced forms of information linking and aggregation.

[5]https://www.w3.org/TR/rdf-sparql-query/

By relying on a semantic approach and adopting LD technology, the most intuitive way to express metric formulas is via SPARQL queries. The main difficulty lies on the fact that 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. This expert would rather prefer to mathematically specify the metric formula in a simplified language. This observation has led to the need to transform the KPI metric specification, as specified in the KPI ontology, into a SPARQL query specification. 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.

This metric formula to SPARQL query transformation included a set of specific hurdles that had to be overcome. First, it relies on the metric kind. We distinguish between two metric kinds: (a) customer-specific, pertaining to a certain BPaaS instance purchased by the customer; (b) broker-specific, pertaining to an overall performance of the BPaaS offered. Customer-specific metrics have as a 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.

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

By considering the above two issues, a particular transformation algorithm has been developed which, depending on the input provided, attempts to construct dynamically the SPARQL query to be issued for deriving the respective metric measurement. The pseudo-code of this algorithm is shown in Listing 1.

Listing 1: Transformation & Drill-Down Algorithms Pseudo-code.

```
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;
}

Hashtable<Metric,ResultSet> drillDown(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()){
  for (MetricNode mn: metrics{
   if (mn.isLeaf()) results.union(evalKPI(mn.metric,
     object,bpaas,start,end,custId));
   else results.union(measureKPI(mn,object,start,
    end,custId,results));
  }
  metrics = getParents(metrics);
 }
 return results;
}
```

The algorithm (see `evaluateKPI` method) comprises five steps: (a) metric formula expansion which involves the recursive substitution of component metrics, for which measurements are not stored in the *Semantic KB*, with the derivation formula of these metrics; (b) derivation of query variables from those metrics, i.e., the leaf metrics, in the expanded formula for which measurements have been stored; (c) production of the (SPARQL) select clause from the expanded formula; (d) production of the whole SPARQL query; (e) evaluation of the SPARQL query over the *Semantic KB*.

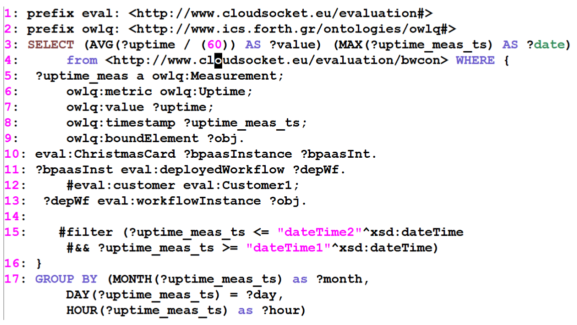
The SPARQL query production (see `createQuery` method) includes the execution of the following steps: (i) the creation of the query prefixes; (ii) the application of the `SELECT` & `FROM` clauses by also considering the respective LD graph URI mapping to the individual RDF graph of the broker from which the relevant information for the query evaluation can be obtained; (iii) the creation of triples mapping to the measurements of the leaf metrics / variables; (iv) the enforcement of the intelinking between measurements according to the object being measured, the customer (if given as input) and the respective BPaaS instances of the measurements; (v) the application of the filtering (`FILTER` clause) over the history period to consider measurements produced only on that period; (vi) the application of SPARQL `GROUP BY` clauses based on the KPI metric evaluation period, i.e., its schedule.

In order to exemplify the transformation algorithm and raise its understanding level, we focus on highlighting its application on a specific example of a KPI metric and we especially take a closer look at the respective SPARQL query being generated.

Suppose that we need to measure the *average availability* metric AVG_A for the whole BPaaS workflow which can be computed from the formula MEAN(RAW_A), where RAW_A represents the instance-based availability metric for this workflow. Moreover, further assume that: (i) the availability metric should be calculated every 1 hour, while the raw availability one every minute; (ii) the history period is 1 day.

The first step of the transformation algorithm will expand this formula based on the measurability of its component metrics. In particular, as RAW_A is not stored in the *Semantic KB*, it is further expanded into its derivation formula $\frac{UPTIME}{TOTAL\_OBSERVATION\_TIME}$, where $UPTIME$ is a raw metric and $TOTAL\_OBSERVATION\_TIME$ is a constant. In this sense, the final expanded formula will become: MEAN$\left(\frac{UPTIME}{TOTAL_OBSERVATION_TIME}\right)$. From this formula, the next two algorithm steps will produce a set of one variable ("?uptime") and the select clause ("SELECT (AVG(?uptime / 60) as ?value) (MAX(?uptime_ms_ts) as ?date)"). Please note that as uptime is calculated every second, the total observation time constant is 60.

The fourth step will focus on generating the actual SPARQL query which is depicted in Figure 4. This figure is now explained by focusing over all the steps involved in the `createQuery` method and the content generated by them.



```
1: prefix eval: <http://www.cloudsocket.eu/evaluation#>
2: prefix owlq: <http://www.ics.forth.gr/ontologies/owlq#>
3: SELECT (AVG(?uptime / (60)) AS ?value) (MAX(?uptime_meas_ts) AS ?date)
4:      from <http://www.cloudsocket.eu/evaluation/bwcon> WHERE {
5:  ?uptime_meas a owlq:Measurement;
6:      owlq:metric owlq:Uptime;
7:      owlq:value ?uptime;
8:      owlq:timestamp ?uptime_meas_ts;
9:      owlq:boundElement ?obj.
10: eval:ChristmasCard ?bpaasInstance ?bpaasInt.
11: ?bpaasInst eval:deployedWorkflow ?depWf.
12:     #eval:customer eval:Customer1;
13:  ?depWf eval:workflowInstance ?obj.
14:
15:    #filter (?uptime_meas_ts <= "dateTime2"^xsd:dateTime
        #&& ?uptime_meas_ts >= "dateTime1"^xsd:dateTime)
16: }
17: GROUP BY (MONTH(?uptime_meas_ts) as ?month,
        DAY(?uptime_meas_ts) = ?day,
        HOUR(?uptime_meas_ts) as ?hour)
```

Figure 4: Constructed SPARQL query for the example.

Lines 1-2 indicate the prefixes of the two ontologies being exploited mapping to the first query generation step. Line 3-4 map to the second step which completely specifies the query `SELECT` & `FROM` clauses.

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

As such, Lines 10-13, mapping to the fourth query creation step, realise the needed connections from the object measured to both the BPaaS instance involving it and the BPaaS under investigation. Line 10 connects the current BPaaS to one of its instances, while Line 11 links this BPaaS instance to a deployed workflow. Line 12, currently commented, would link this instance to a specific client that has purchased it, in case we deal with a customer-specific metric. Finally, Line 13 maps the deployed workflow to the actual workflow instance measured. Lines 11-13 can be differentiated based on the kind of object being measured. For instance, if a task instance is measured, then we need to add another triple pattern indicating that the workflow instance includes this task instance.

Line 15, currently commented, maps to the fifth query creation step and provides a SPARQL `FILTER` constraint that can restrain the history period under investigation. In particular, the dateTime of the measurement is mapped to two simple constraints 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. The commendation of this line signifies that the whole evaluation history of the KPI metric is explored.

Finally, Line 17, generated by the last query creation step, provides a grouping statement where the last sub-group maps to the evaluation period of the KPI metric (per hour). This statement groups first the results according to the month, then according to the day and then according to the respective hour.

The derivation of KPI drill-down knowledge is handled by another algorithm which maps to the `drillDown` method in Listing 1. As it can be seen, this algorithm exploits the transformation one by also considering the whole derivation tree of the current KPI metric at hand. The drill-down algorithm steps are the following: (a) start with the top metric's derivation list and expand on it recursively until leaf metric nodes are reached. This leads to the production of a metric (derivation) tree which has as nodes either formula or metric nodes. The latter nodes map only to metrics for which measurements exist in the *Semantic KB*; (c) compute the needed metric values according to the SPARQL-based transformation approach in a bottom-up way; The latter approach is used only for the leaf metric nodes in the metric derivation tree. Then, the produced measurements are propagated up in the tree by considering the respective formula and metric nodes visited in a level-by-level manner. Each time a KPI metric node is visited, its values are produced according to the metric formula involved and the already produced measurements and they are stored in the hashtable, from metrics to measurement result sets, to be returned.

6 CONCLUSIONS

This paper has presented a semantic approach to KPI measurement which enables the clever and dynamic exploration of the KPI metric space. This approach relies on the appropriate definition of the KPI metric, the expansion of its formula and its transformation into a SPARQL query that is then issued over a semantic KB. A KPI drill-down capability is also offered by the proposed KPI analysis system which capitalises over the KPI measurement algorithm and the KPI's hierarchy tree. This paper has also proposed specific ontologies focusing on the complete KPI definition and the capturing of BPaaS dependency models. Both ontologies can be exploited to semantically link information originating from the BPaaS execution so as to populate the semantic KB and enable various types of BPaaS analysis over it, apart from the currently offered one, such as process mining or best BPaaS deployment discovery.

The following research directions are planned. First, further validating the two proposed ontologies from use cases of the CloudSocket project to obtain suitable feedback for optimising them. Second, evaluating the KPI analysis system based on both performance and accuracy aspects. Third, realising additional BPaaS analysis algorithms into the respective KPI analysis system 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

Caplan, R. S. and Norton, D. P. (1992). The Balanced Scorecard Measures that Drive Performance. *Harvard Business Review*, 70(1):281–308.

Castellanos, M., Casati, F., Shan, M.-C., and Dayal, U. (2005). ibom: A platform for intelligent business operation management. In *ICDE*, pages 1084–1095, Washington, DC, USA. IEEE Computer Society.

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., Stanoi, I., and Zeng, L. (2006). Model Driven Development for Business Performance Management. *IBM Syst. J.*, 45(3):587–605.

Costello, C. and Malloy, O. (2008). Building a Process Performance Model for Business Activity Monitoring. In Wojtkowski, W., Wojtkowski, G., Lang, M., Conboy, K., and Barry, C., editors, *Information Systems Development - Challenges in Practice, Theory, and Education*, pages 237–248. Springer-Verlag.

Cui, Y. and Nahrstedt, K. (2001). QoS-Aware Dependency Management for Component-Based Systems. In *HPDC*, page 127. IEEE Computer Society.

De Medeiros, A. K. A., Pedrinaci, C., van der Aalst, W. M. P., Domingue, J., Song, M., Rozinat, A., Norton, B., and Cabral, L. (2007). An Outlook on Semantic Business Process Mining and Monitoring. In *OTM*, pages 1244–1255. Springer-Verlag.

del Río-Ortega, A., Resinas, M., Durán, A., and Ruiz-Cortés, A. (2016). Using templates and linguistic patterns to define process performance indicators. *Enterp. Inf. Syst.*, 10(2):159–192.

Diamantini, C., Potena, D., Storti, E., and Zhang, H. (2014). An Ontology-Based Data Exploration Tool for Key Performance Indicators. In *ODBASE*, pages 727–744, Amantea,Italy. Springer-Verlag.

Frank, U., Heise, D., Kattenstroth, H., and Schauer, H. (2008). Designing and utilising business indicator systems within enterprise models: Outline of a method. In *MobIS: Modellierung zwischen SOA und Compliance Management*, Saarbröcken, Germany.

Friedenstab, J.-P., Janiesch, C., Matzner, M., and Muller, O. (2012). Extending BPMN for Business Activity Monitoring. In *HICSS*, pages 4158–4167. IEEE Computer Society.

González, O., Casallas, R., and Deridder, D. (2009). MMC-BPM: A domain-specific language for business processes analysis. In *BIS*, volume 21, pages 157–168, Poznan, Poland. Springer.

Gruschke, B. (1998). Integrated Event Management: Event Correlation Using Dependency Graphs. In *DSOM*.

Hasselmeyer, P. (2001). Managing Dynamic Service Dependencies. In *DSOM*, pages 141–150, Nancy, France. INRIA.

Kritikos, K., Magoutis, K., and Plexousakis, D. (2016). Towards Knowledge-Based Assisted IaaS Selection. In *CloudCom*, Luxembourg. IEEE Computer Society.

Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benbernou, S., Brandic, I., Kertész, A., Parkin, M., and Carro, M. (2013). A survey on service quality description. *ACM Comput. Surv.*, 46(1):1.

Kritikos, K. and Plexousakis, D. (2006). Semantic QoS Metric Matching. In *ECOWS*, pages 265–274. IEEE Computer Society.

Liu, R., Nigam, A., Jeng, J., Shieh, C., and Wu, F. Y. (2010). Integrated Modeling of Performance Monitoring with Business Artifacts. In *ICEBE*, pages 64–71, Shanghai, China. IEEE Computer Society.

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

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

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

Seedorf, S. and Schader, M. (2011). Towards an Enterprise Software Component Ontology. In *AMCIS*. Association for Information Systems.

Wetzstein, B., Karastoyanova, D., and Leymann, F. (2008a). Towards Management of SLA-Aware Business Processes Based on Key Performance Indicators. In *BP-MDS*, Montpellier, France.

Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., and Leymann, F. (2009). Monitoring and Analyzing Influential Factors of Business Process Performance. In *EDOC*, pages 118–127. IEEE Press.

Wetzstein, B., Ma, Z., and Leymann, F. (2008b). Towards Measuring Key Performance Indicators of Semantic Business Processes. In *BIS*, page 227238. Springer-Verlag.

Woitsch, R., Albayrak, M., Köhn, H., Utz, W., Ferrer, A. J., Iranzo, J., Leonforte, A., Gallo, A., Mihnea, V., Pacurar, R., Avasilcai, C., Arama, G., Boca, R., Griesinger, F., Seybold, D., Domaschka, J., Kritikos, K., and Plexousakis, D. (2015). *D4.1 – First CloudSocket Architecture*. CloudSocket European Project.